ALIBABA CLOUD

阿里云

智能媒体管理

用户指南

文档版本：20220704

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用须知 08
2.基础概念 09
3.服务地域 10
4.使用流程 11
5.文档预览 13
5.1. 概述 13
5.2. 文档预览V1/V2版本对比说明 13
5.3. 文档预览V1 15
5.3.1. 快速入门 15
5.3.2. 文档格式转换 25
5.3.3. JavaScript API预览高级功能配置 28
5.4. 文档预览V2 36
5.4.1. 快速入门 36
5.4.2. 支持的文档格式 40
5.4.3. 自定义配置 41
5.4.3.1. 事件监听 41
5.4.3.2. 高级用法 45
5.4.3.3. 文字相关 46
5.4.3.4. PDF相关 48
5.4.3.5. 演示相关 49
5.4.3.6. 表格相关 52
6.文档在线协作 53
6.1. 快速入门 53
6.2. 事件 55
6.3. 兼容性 59
6.4. 基础用法 61

6.5. 高级用法 72
6.5.1. 调用流程 72
6.5.2. 释放对象 74
6.5.3. 公共 76
6.5.4. 文字 77
6.5.4.1. 页面 77
6.5.4.2. 内容控件 92
6.5.4.3. 书签 97
6.5.4.4. 修订 103
6.5.4.5. 评论 109
6.5.4.6. 缩放 111
6.5.4.7. 选区 112
6.5.4.8. 水印 118
6.5.4.9. 文本 128
6.5.4.10. 表格 139
6.5.4.11. 公文域 158
6.5.4.12. 超链接 175
6.5.4.13. 图形 176
6.5.4.14. 会议 192
6.5.4.15. 目录 194
6.5.5. 表格 199
6.5.5.1. 页面 199
6.5.5.2. 活动工作簿 209
6.5.5.3. 活动工作表 214
6.5.5.4. 名称管理器 225
6.5.5.5. 图形 229
6.5.5.6. 区域 240
6.5.5.7. 会议 275

6.5.6. 演示 282
6.5.6.1. 页码 282
6.5.6.2. 页面 284
6.5.6.3. 页面设置 293
6.5.6.4. 幻灯片 295
6.5.6.5. 会议 308
6.5.7. PDF 319
6.5.7.1. 页码 319
6.5.7.2. 页面 321
6.5.7.3. 缩放 328
6.5.7.4. 滚动 329
6.6. 页面定制 331
6.6.1. 页面定制概述 331
6.6.2. 隐藏或显示组件 333
6.6.3. 禁用或启用组件 333
6.6.4. 控制按钮单击 334
6.6.5. 搜索指定内容 335
6.6.6. 可定制列表 336
6.6.7. 定制元素 344
6.7. JS-SDK版本 357
7.图片管理 360
7.1. 概述 360
7.2. 图片分析 361
7.2.1. 内容识别 361
7.2.2. 二维码识别 362
7.2.3. 图片裁剪建议 362
7.2.4. 图片质量评分 363
7.3. 人脸人体 363

7.3.1. 人脸检测 363
7.3.2. 人脸搜索 363
7.3.3. 人脸比对 364
7.3.4. 人脸聚类 364
7.3.5. 人体检测 365
7.4. 图片盲水印 365
7.5. 标签搜索 367
8.通过OSS使用智能媒体管理 368
9.授权管理 369
9.1. 配置用户权限 369
9.2. 基于RAM Policy的权限控制 370
10.用户限额（Quota） 373

1.使用须知

为了保证正常运行智能媒体管理服务，在使用前请务必认真阅读注意事项。

RAM用户访问

采用RAM用户访问智能媒体管理服务，推荐为该RAM用户授予AliyunIMMFullAccess权限，从而拥有登录智能媒体管理控制台管理和调用API的能力。

文档格式转换

智能媒体管理支持将48种常见的文件格式转换为JPG、PNG、PDF、TXT、VECTOR 5种目标文档格式。

支持的输入文件类型包含如下格式：

- 文字文件：DOC、DOT、WPS、WPT、DOCX、DOTX、DOCM、DOTM
- 演示文件：PPTX、PPT、POT、POTX、PPS、PPSX、DPS、DPT、PPTM、POTM、PPSM
- 表格文件：XLS、XLT、ET、ETT、XLSX、XLTX、CSV、XLSB、XLSM、XLTM
- 其他格式文件：PDF、LRC、C、CPP、H、ASM、S、JAVA、ASP、BAT、BAS、PRG、CMD、RTF、TXT、LOG、XML、HTM、HTML

支持的输出文件类型包含如下格式：

- JPG格式：按文件样式每页生成一张JPG图片。
- PNG格式：按文件样式每页生成一张PNG图片。
- PDF格式：每个文件生成一个PDF文件。
- TXT格式：按文件样式每页生成一个TXT文件。
- VECTOR向量格式：按文件样式每页生成一个VECTOR向量格式文件。

图片AI功能

- 图片格式支持输入JPG、PNG、BMP格式的图片。除二维码检测功能外，其他功能还支持输入GIF格式的图片，GIF格式的图片均以首帧静态图作为检测依据。
- 当输入WEBP、TIFF、HEIC格式的图片时，智能媒体管理会自动尝试使用OSS提供的图片转换能力进行格式转换，然后进行图片AI处理。

  > **注意** 此过程将可能产生一定转换费用。更多信息，请参见数据处理计费说明。

- 输入图片的最大为100 MB。二维码检测功能输入图片的最小为40 px*40 px。
- 对图片中的人脸数目没有限制，默认返回所有识别出来的人脸；检测时间和图片尺寸有关，图片越大耗时越长。
- 对于过大的图片，请求可能超时，请减小图片尺寸后再次尝试。

2.基础概念

通过本文您可以了解智能媒体管理的基本概念，例如项目、数据源等。

项目

开发者使用智能媒体管理服务的基本单元。例如开发者开发网盘APP时，在与存储数据的OSS Bucket相同的地域创建项目，并对项目授予访问OSS Bucket的权限，可以便捷的使用智能媒体管理的文档预览、图片AI功能等。

智能媒体管理包括如下项目类型：

- 文档标准型：支持格式转换、文档预览功能。
- 图片标准型：支持内容识别、二维码检测、人脸检测等图片AI功能。

数据源

为智能媒体管理提供数据来源，当前只支持OSS作为数据源。

数据源可以为整个Bucket或者Bucket的前缀，通过在RAM控制台配置角色Policy时实现控制，再通过智能媒体管理服务创建项目时指定角色，完成对数据源的访问控制。

请求类型

- 同步请求

  客户端发出请求后阻塞等待结果返回，通常在几百毫秒内返回。例如图片实例中的人脸检测、内容识别等。

- 异步请求

  为了解决同步任务执行时间太长，导致客户端阻塞太久的问题，例如超过10秒客户端可能会链接断开，因此智能媒体管理引入了异步请求机制。

  客户端发出请求后，系统会快速返回ID，基于此ID，可以查询异步请求的状态。例如大文件的格式转换请求可能几十秒，甚至分钟级才能完成。

任务

单个数据处理异步请求的执行实体，调用异步请求API返回的ID，即任务ID。例如执行单个文档的异步转换，它的执行实体就是一个任务。

任务创建后，支持Get查询任务状态，Delete删除任务，List遍历任务请求。

3.服务地域

通过本文您可以了解智能媒体管理目前已开放的服务地域（Region），以及服务地域的地域ID和接入地址。

> **说明**
>
> 更多服务地域正在开放中，如果有紧急开服需求，请提交工单联系我们。

服务地域（Region）	地域ID	接入地址
华东1（杭州）	cn-hangzhou	imm.cn-hangzhou.aliyuncs.com
华东2（上海）	cn-shanghai	imm.cn-shanghai.aliyuncs.com
华北2（北京）	cn-beijing	imm.cn-beijing.aliyuncs.com
华北3（张家口）	cn-zhangjiakou	imm.cn-zhangjiakou.aliyuncs.com
华南1（深圳）	cn-shenzhen	imm.cn-shenzhen.aliyuncs.com

4.使用流程

本文介绍了文档管理场景、图片社交场景和家庭设备场景下产品的使用流程。

前提条件

- 已创建RAM用户并授予RAM用户AliyunIMMFullAccess权限。具体操作，请参见配置用户权限。
- 已为RAM用户创建AccessKey。具体操作，请参见为RAM用户创建访问密钥。

> **说明** AccessKey包括AccessKey ID和AccessKey Secret，用于调用API。

- 已开通智能媒体管理服务。具体操作，请参见开通产品。
- 已开通对象存储OSS并上传文档到OSS。具体操作，请参见开始使用OSS。

文档管理场景

使用文档格式转换功能，可以将48种文档格式转换为JPG、PNG、PDF、TXT和VECTOR 5种格式，实现在应用中更好的展示文档。使用步骤如下：

1. 创建项目。
   - 通过智能媒体管理控制台创建项目。
2. 通过同步请求或者异步请求进行文档格式转换。

   同步请求需要在5秒的时间内完成文档转换，如果同步请求转换失败，请使用异步请求。

   转换成功后，可以在OSS的指定目录查看对应的输出结果。

   - 同步请求

     调用ConvertOfficeFormat接口进行格式转换。

   - 异步请求

     调用CreateOfficeConversionTask接口进行格式转换。

     创建格式转换任务后，您可以通过调用GetOfficeConversionTask接口获取任务状态或者使用消息队列获取转换结果的通知（Notification）。

智能媒体管理提供文档预览V1和文档预览V2两个版本的文档预览，请根据实际场景选择。

> **注意** 如果预览服务的域名和OSS Bucket的域名不同，需要设置OSS Bucket的跨域资源共享，允许预览服务器访问。具体操作，请参见设置跨域资源共享。

- 文档预览V1：将输入文档转换为VECTOR向量格式后，通过对接智能媒体管理提供的前端渲染引擎，实现更易用、功能更强大、定制化的文档预览效果。更多信息，请参见快速入门。

  > **说明** VECTOR格式的文档支持复制文字，缩放不失真，自定义广告等功能。

- 文档预览V2：获取文档的预览地址和AccessToken后，无需指定iframe元素，JS文件会自动在自定块元素下生成iframe，并通过JS文件设置AccessToken即可快捷的实现文档预览。更多信息，请参见快速入门。

文档预览V1的使用步骤如下：

1. 创建项目。
   - 通过智能媒体管理控制台创建项目。

2. 通过异步请求进行文档格式转换。

   调用CreateOfficeConversionTask接口进行格式转换。

   创建格式转换任务后，您可以通过调用GetOfficeConversionTask接口获取任务状态或者使用消息队列获取转换结果的通知（Notification）。

3. 对接预览引擎，将生成到OSS的文件作为参数传递给渲染引擎。

文档预览V2的使用步骤如下：

1. 创建项目。
   - 通过智能媒体管理控制台创建项目。
2. 在服务端分别封装GetOfficePreviewURL和RefreshOfficePreviewToken接口，获取预览地址和AccessToken。
3. 无需指定iframe元素，JS文件会自动在自定块元素下生成iframe，并通过JS文件设置AccessToken即可快捷的实现文档预览。

图片社交分析场景

本场景通常使用图片AI功能，通过调用DetectImageTags接口返回标签信息或者调用DetectImageFaces接口返回人脸信息，然后基于AI处理图片得到的元数据，使用智能媒体管理提供的元数据索引能力，可以帮助您的应用快速构建标签搜索能力，例如在图片社区应用中加入搜索栏，查询具有猫的图片。使用步骤如下：

1. 创建项目。
   - 通过智能媒体管理控制台创建项目。
2. 调用CreateSet接口创建媒体集，并获取媒体集ID（即SetId）。
3. 调用IndexImage接口将图片索引到媒体集中。

   当需要索引多张图片时，您可以指定应用多次调用该接口。

4. 调用ListSetTags接口获取媒体集中的标签列表。
5. 调用FindImages接口按指定标签搜索媒体集中的图片。

家庭设备数据存储场景

本场景通常使用图片人脸AI功能，使用您的应用视频截帧生成的多张图片，调用图片人脸聚类功能，实现对家庭成员的识别。使用步骤如下：

1. 创建项目。
   - 通过智能媒体管理控制台创建项目。
2. 调用CreateSet接口创建媒体集，并获取媒体集ID（即SetId）。
3. 调用IndexImage接口将图片索引到媒体集中。

   当需要索引多张图片时，您可以指定应用多次调用该接口。

4. 调用CreateGroupFacesJob接口对媒体集中的人脸进行聚类操作。

   调用CreateGroupFacesJob接口后，您可以继续调用IndexImage接口将图片中的人脸索引到媒体集，然后再次调用CreateGroupFacesJob接口进行增量聚类的处理。

5. 调用ListFaceGroups接口获取一个媒体集中的人脸分组列表。

5.文档预览

5.1. 概述

通过文档标准型项目，可以进行文档处理，例如文档格式转换和文档预览，常用于网页转码、网盘文档预览、邮箱附件预览、知识管理的文档格式转换等场景。

格式转换

智能媒体管理支持将48种常见的文档格式转换为JPG、PNG、PDF、TXT、VECTOR 5种目标文档格式。更多信息，请参见文档格式转换。

文档预览

智能媒体管理提供两个版本的文档预览，请根据实际选择合适的文档预览版本。

- 文档预览V1：将输入文档转换为VECTOR向量格式后，通过对接智能媒体管理提供的前端渲染引擎，实现更易用、功能更强大、定制化的文档预览效果。更多信息，请参见快速入门。

  > **说明** VECTOR格式的文档支持复制文字，缩放不失真，自定义广告等功能。

- 文档预览V2：获取文档的预览地址和AccessToken后，无需指定iframe元素，JS文件会自动在自定块元素下生成iframe，并通过JS文件设置AccessToken即可快捷的实现文档预览。更多信息，请参见快速入门。

计费方式

文档标准型项目支持按QPS计费和按次计费两种计费方式。

当按QPS计费时，1 QPS的费用为60元，当按次计费时，1次API调用的费用为0.08元。因此如果应用每天的请求超过750（60/0.08 = 750）次，则适合选择按QPS计费。更多信息，请参见计费说明。

- 按QPS计费

  按每天的处理能力计费，QPS表示每秒的并发处理能力，该值为1表示每秒钟只能处理1个文档请求。

  > **说明** 当文档转换请求为异步请求时，异步请求的队列深度和QPS值保持一致，因此当QPS为1时，如果已有请求在执行，提交新的请求会返回错误。

- 按次计费

  按实际的API调用次数计费，每小时结算一次。

  为了控制瞬时提交大量API带来的影响，此计费模式下每次只能提交一个任务，只有等一个任务结束后才能提交另一个任务。

5.2. 文档预览V1/V2版本对比说明

智能媒体管理提供两版文档预览方式，包括文档预览V1和文档预览V2。本文介绍两版预览方式的主要区别（目前文档预览V1已经停止更新和维护，推荐使用V2版本的文档预览）。

预览原理

版本	说明
文档预览V1	将输入文档转换为VECTOR向量格式后，通过对接智能媒体管理提供的前端渲染引擎，实现更易用、定制化的文档预览效果。更多信息，请参见预览原理。
文档预览V2	获取文档地址和AccessToken后，无需指定iframe元素，JS文件会自动在自定块元素下生成iframe，并通过JS文件设置AccessToken即可快捷的实现文档预览。更多信息，请参见预览原理。

预览方式

版本	说明
文档预览V1	智能媒体管理的预览引擎支持URL参数预览和JavaScript API预览两种方式，URL参数预览更加简单，JavaScript API预览有更多的配置，请根据应用场景选择合适的预览方式。更多信息，请参见预览方式对比。
文档预览V2	调用GetWebofficeURL接口，获取协作地址。通过JS-SDK将返回的预览地址挂载到HTML块状元素中并设置AccessToken，JS文件会自动在自定块元素下生成iframe，并通过JS文件设AccessToken即可快捷的实现文档预览。更多信息，请参见预览流程。

支持功能

对比项	文档预览V1	文档预览V2	说明
页面定制	不支持	支持	您可以通过文档预览V2提供的页面定制功能为文档添加、隐藏或启用组件。更多信息，请参见页面定制概述。
添加水印	支持（前端）	支持	• 文档预览V1：您可以通过JavaScript API预览方式为文档添加水印。更多信息，请参见JavaScript API预览。 • 文档预览V2：您可以通过调用GetWebofficeURL接口并配置参数 `Watermark` 为文档添加水印。
用户信息	不支持	支持	您可以通过调用文档预览V2的GetWebofficeURL接口并配置参数 `User` 配置用户信息。

对比项	文档预览V1	文档预览V2	说明
用户权限	不支持	支持	您可以通过调用文档预览V2的GetWebofficeURL接口并配置参数 `Permission` 配置用户权限。

文档配置

版本	说明
文档预览V1	文档预览时支持复制文档中的文字，缩放文档内容不失真，自定义广告等配置。更多信息，请参见快速入门。
文档预览V2	文档预览时支持自定义配置，包括事件监听、多种文档相关配置等。更多信息，请参见自定义配置。

5.3. 文档预览V1

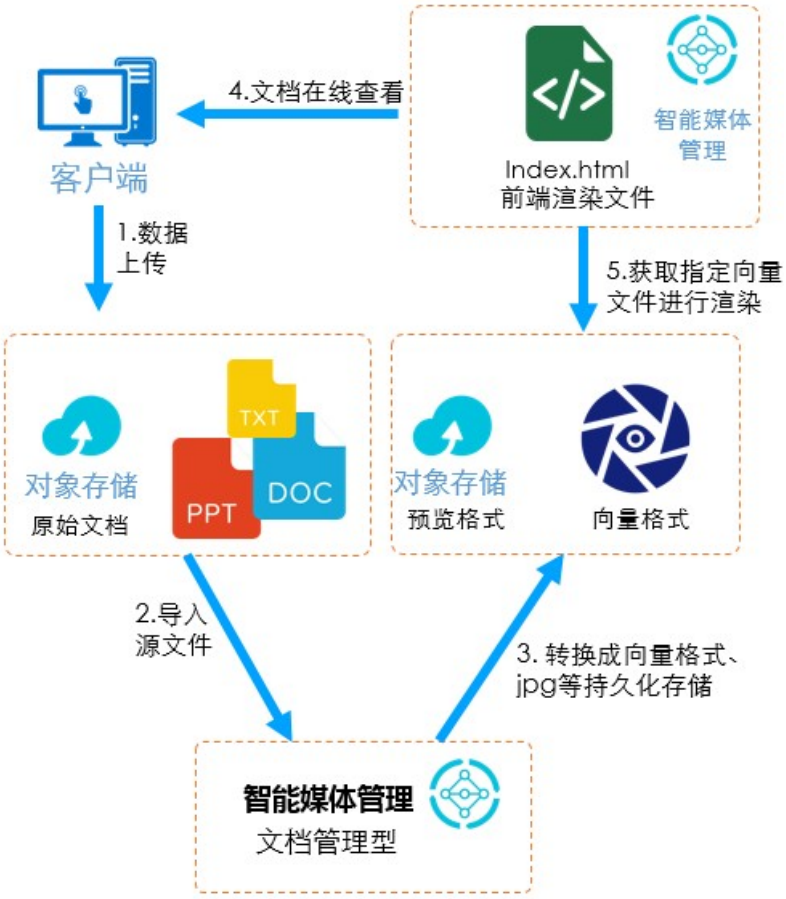
5.3.1. 快速入门

文档预览V1将输入文档转换为VECTOR向量格式后，通过对接智能媒体管理提供的前端渲染引擎，实现更易用、功能更强大、定制化的文档预览效果。文档预览时支持复制文档中的文字，缩放文档内容不失真，自定义广告等功能。

> **说明** 将输入文档转换为JPG、PNG等格式时，您可以直接搭建一个在线文档预览系统进行文档预览。

预览原理

将输入文档转换为VECTOR向量格式后，通过对接智能媒体管理提供的前端渲染引擎实现文档预览。



准备工作

1. 设置跨域访问规则。

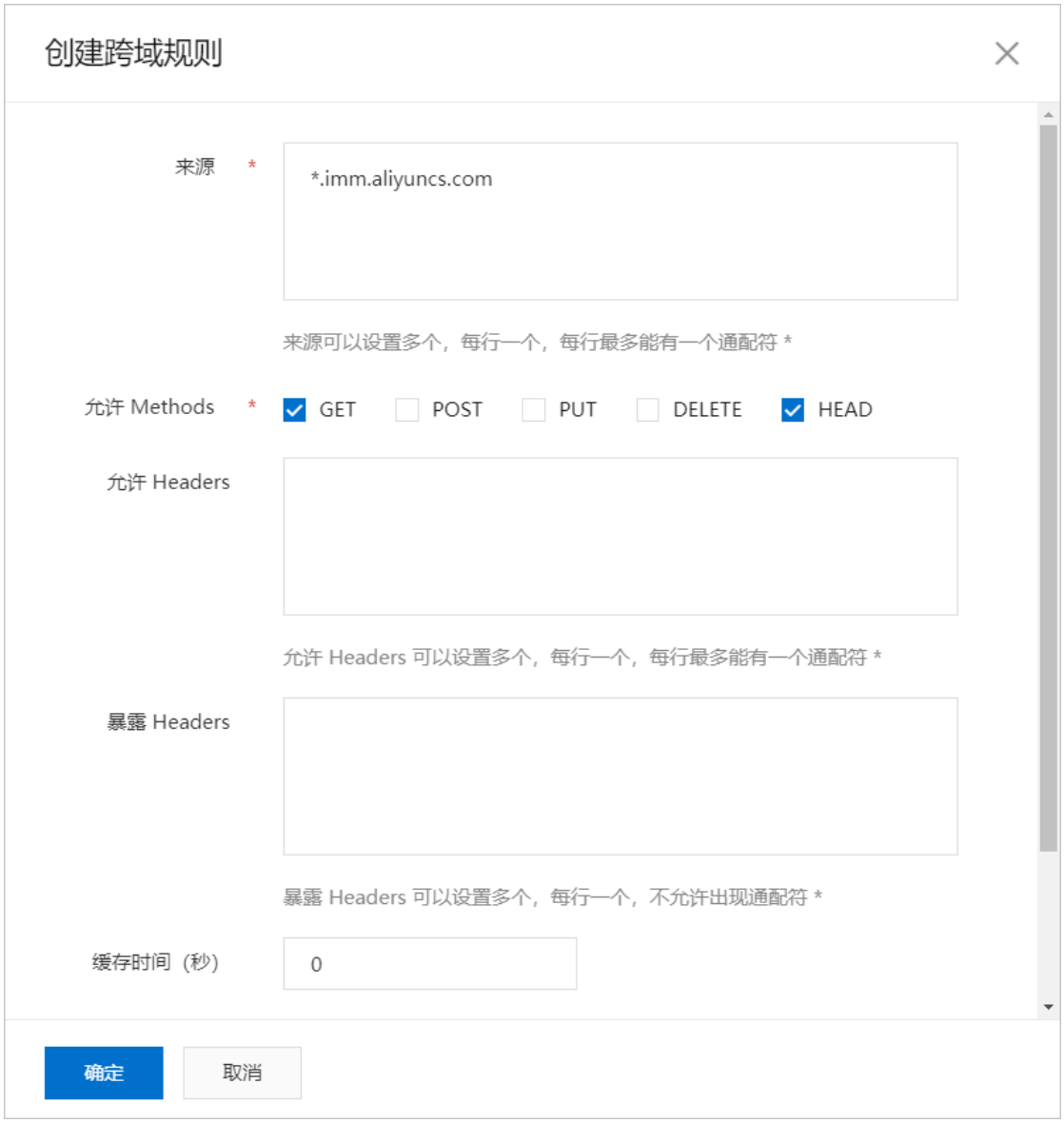
   由于访问的OSS Bucket域名与预览引擎的域名不同，因此请通过OSS管理控制台将预览引擎域名添加到存储转换后文档的OSS Bucket的跨域访问列表中。更多信息，请参见设置跨域资源共享。设置跨域访问规则的具体操作步骤如下：

   i. 登录OSS管理控制台后，单击目标Bucket名称，选择**权限管理 > 跨域设置**。

   ii. 在**跨域设置**区域单击**设置**。

iii. 单击**创建规则**，配置跨域规则。

配置跨域规则时，来源需要配置为***.imm.aliyuncs.com**，允许Methods选择**GET**和**HEAD**，其他参数请根据需要填写。



iv. 单击**确定**。

2. 获取预览文件的临时访问权限。

为了数据安全性，预览引擎只支持从私有的OSS Bucket中读取文档。当OSS Bucket设置为私有时，此时需要通过OSS STS获取该文件临时的访问权限，获取临时访问凭证后传递给预览引擎，预览引擎通过凭证获取相关权限，实现文档预览。

临时访问凭证包括临时访问密钥（AccessKey ID和AccessKey Secret）和安全令牌（SecurityToken），关于获取临时访问权限的具体操作，请参见使用STS临时访问凭证访问OSS。

> **说明** 由于预览引擎需要从用户的OSS Bucket读取转换后的文档，受用户OSS Bucket的读取权限影响。OSS权限的更多信息，请参见了解OSS读写权限控制。

3. 上传原始文档到对象存储OSS中。具体操作，请参见上传文件。

4. 调用智能媒体管理的CreateOfficeConversionTask接口，将原始文档转换为VECTOR向量格式输出到指定的OSS目录。

预览方式对比

智能媒体管理的预览引擎支持URL参数预览和JavaScript API预览两种方式，两种预览方式的对比如下，请根据应用场景选择合适的预览方式，推荐使用JavaScript API预览方式。

> **说明** 预览引擎支持的浏览器包括Internet Explorer 10+、Microsoft Edge、Google Chrome、Firefox、Safari和Opera。

预览方式	描述
URL参数预览	通过URL QueryString将必要的参数传递给预览引擎，预览引擎获取相关参数后在页面中展示文档。 此方式操作快捷方便，但是相关临时授权凭证会暴露在iframe的URL地址中。当用户的OSS Bucket为私有读并且签名有效期内时，只要把对应的URL从iframe src属性中复制出来即可分享给别人，对安全性要求较高的业务不建议使用此方式。
JavaScript API预览	通过postMessage将必要参数发送给预览引擎页并完成初始化工作。 JavaScript API预览具有更高的安全性，并且支持全屏预览、文字和演示页码跳转控制、文字和演示预览界面UI自定义等高级功能配置。更多信息，请参见JavaScript API预览高级功能配置。 此方式采用隐式传递参数，安全性较高，但是需要部分额外的前端编程工作。由于iframe的URL地址中不包含签名，因此即使复制了URL地址也无法预览。 **说明** postMessage浏览器兼容的更多信息，请参见postMessage浏览器兼容性。

URL参数预览

您可以通过完整的URL在浏览器中直接预览文档，也可以将智能媒体管理提供的预览引擎嵌入到页面或者Web App中进行文档预览。

- 通过完整的URL预览文档

  完整的预览URL格式如下所示：

  ```
  https://preview.imm.aliyuncs.com/index.html?url=[url]&accessKeyId=[accessKeyId]&accessKeySecret=[accessKeySecret]&stsToken=[stsToken]&bucket=[bucket]&region=[region]
  ```

  其中 `https://preview.imm.aliyuncs.com/index.html` 为智能媒体管理服务提供的预览引擎地址，问号（`?`）后的参数为预览文档的信息，如果未提供必填参数，则会预览失败。

  - 参数

参数	类型	是否必填	描述

参数	类型	是否必填	描述
url	String	是	文档转换后的向量文件所在OSS目录。 如果提交文档转换任务的TgtUri为oss://your-bucket/paxos.pptx/output，则URL的格式为https://your-bucket.oss-{region}.aliyuncs.com/paxos.pptx/output。 注意 填写时无需在OSS目录结尾加/。
accessKeyId	String	是	从STS获取的访问OSS的访问密钥标识（AccessKeyId）。
accessKeySecret	String	是	从STS获取的访问OSS的访问密钥（AccessKeySecret）。
stsToken	String	是	从STS获取访问OSS的安全令牌（SecurityToken）后，对SecurityToken进行编码得到的结果。编码的具体操作，请参见签名机制。
region	String	是	OSS Bucket所在地域，例如oss-cn-shanghai。 注意 请注意添加OSS前缀。
bucket	String	是	文档转换后的向量文件所在OSS Bucket的名称，例如your-bucket。
endpoint	String	否	使用与OSS Bucket绑定的用户域名进行预览，使用此参数时无需传递bucket和region参数。
pageIndex	Int	否	从指定的页码开始预览。
serverTime	Int	否	服务器当前时间，用于校准本地OSS签名时的时间，避免本地与服务器时差大于15分钟导致签名失败。单位为秒。
expires	Int	否	文档预览的有效时间。默认值为1800，单位为秒。

- 示例

```
https://preview.imm.aliyuncs.com/index.html
?url=https://yourid-dev-imm.oss-cn-shanghai.aliyuncs.com/paxos.pptx/output   //文档转换后的向量文件所在OSS目录。请注意无需在output后加/。
&accessKeyId=STS.AAAA    //从STS获取的AccessKey ID。请注意参数的access小写。
&accessKeySecret=BBBB    //从STS获取的AccessKey Secret。请注意参数的access小写。
&stsToken=CCCC    //CCCC是编码从STS获取的SecurityToken后得到的结果，从STS获取的SecurityTokeen不能直接使用。
&region=oss-cn-shanghai    //OSS Bucket所在地域，请注意添加OSS前缀。
&bucket=bucket-name    //文档转换后的向量文件所在OSS Bucket的名称。
&...
```

- 将预览引擎嵌入到页面或者Web App中预览文档

  将智能媒体管理提供的预览引擎 `preview.imm.aliyuncs.com/index.html?url=[TgtUri]` 通过HTML的<iframe>标签嵌入到页面或者Web App中进行文档预览。其中 `TgtUri` 为预览文件在OSS的http（s）路径，需要通过QueryString参数传入，如下图所示。

JavaScript API预览

为便于描述，以下称阿里云的iframe页面为子页面，嵌入该iframe的接入方页面称为父页面。

1. 子页面发送preview.ready事件。
2. 父页面接收preview.ready事件后，发送preview.init事件以及传递初始化参数。

   父页面向子页面发送消息，通过sendMessage完成，调用格式为 `sendMessage(action, data)` 。

参数	类型	是否必填	描述
action	String	是	事件名称，包括preview.init和setConfig。 ○ preview.init：父页面传递初始化子页面参数。 ○ setConfig：父页面设置子页面相关的渲染参数。

参数	类型	是否必填	描述

data	Object	否	事件参数。
			○ 关于preview.init事件中的参数说明，请参见参数说明。
			○ setConfig事件使用setData传递参数变量，函数中的变量以window.iframeData形式在子页面中访问，可多次调用。支持功能的更多信息，请参见JavaScript API预览高级功能配置。
			`sendMessage("setData", {`
			`//需要在子页面中用到的数据。`
			`});`
			`//注意sendMessage封装了postMessage，详细操作请参见示例代码。`

3. 子页面接收preview.init事件及其参数，并根据参数初始化渲染引擎。

- 参数

  preview.init事件中父页面传递初始化子页面参数，详细参数说明请参见下表。

参数	类型	是否必填	描述
url	String	是	文档转换后的向量文件所在OSS目录。 如果提交文档转换任务的TgtUri为oss://your-bucket/paxos.pptx/output，则URL的格式为https://your-bucket.oss-{region}.aliyuncs.com/paxos.pptx/output。 **注意** 填写时无需在OSS目录结尾加/。
region	String	是	OSS Bucket所在地域，例如oss-cn-shanghai。 **注意** 请注意添加OSS前缀。
bucket	String	是	文档转换后的向量文件所在OSS Bucket的名称，例如your-bucket。
accessKeyId	String	是	从STS获取的访问OSS的访问密钥标识（AccessKey ID）。
accessKeySecret	String	是	从STS获取的访问OSS的访问密钥（AccessKey Secret）。

参数	类型	是否必填	描述
stsToken	String	是	从STS获取的访问OSS的安全令牌（SecurityToken）。
endpoint	String	否	使用与OSS Bucket绑定的用户域名进行预览，使用此参数时无需传递bucket和region参数。
copy	Int	否	是否允许文字拷贝，可选值如下： ○ 0（默认值）：禁止文字拷贝。 ○ 1：允许文字拷贝。
wmType	Int	否	水印类型，可选值如下： ○ 0（默认值）：关闭水印。 ○ 1：文字水印。 ○ 2：图片水印。
wmValue	String	否	水印值。 ○ 当wmType设置为1时，填写水印内容。 ○ 当wmType设置为2时，填写图片水印地址。
wmColor	String	否	水印颜色的RGBA值。默认值为rgba(192, 192, 192, 0.6)。
wmRotate	Float	否	水印角度。默认值为-Math.PI/4。
wmFont	String	否	水印字体。默认值为bold 20px Serif。
wmHeight	Int	否	水印高度。默认值为170，单位为px。
wmWidth	Int	否	水印宽度。默认值为195，单位为px。

- 示例

```
<html>
  <head>
    <meta charset="UTF-8">
    <!-- 建议禁用外框浏览器自带的缩放 -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1
.0, minimum-scale=1.0,user-scalable=no">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title></title>
    <style>
      * {
        box-sizing: border-box;
      }
      html, body {
        padding: 0;
        margin: 0;
        height: 100%;
```

```
      /* 防止双击缩放 */
      touch-action: manipulation;
    }
    .main {
      display: flex;
      flex-direction: column;
      height: 100%;
    }
    #aliyunPreview {
      flex: 1;
    }
  </style>
  <script type="text/javascript" charset="utf-8">
    function json2str(obj) {
      return JSON.stringify(obj, function(key, val) {
        if (typeof val === 'function') {
          val = val.toString();
        }
        return val;
      });
    };
    window.sendMessage = function(action, data) {
      var iframe = document.getElementById('aliyunPreview');
      iframe.contentWindow.postMessage(json2str({ action: action, data: data }), '*');
    };
    window.addEventListener('message', function(e) {
      try {
        var res = JSON.parse(e.data);
      } catch(err) {
        return;
      }
      switch (res.action) {
        case 'preview.ready':
          window.sendMessage('preview.init', {
            url: '',
            region: '',
            bucket: '',
            accessKeyId: '',
            accessKeySecret: '',
            stsToken: ''
          });
          break;
      }
    }, false);
    //禁止双指缩放手势。
    document.addEventListener('gesturestart', function (e) {
      e.preventDefault();
    });
  </script>
</head>
<body>
  <iframe
    allowfullscreen
    id="aliyunPreview"
```

```
      frameborder="0"
      src="https://preview.imm.aliyuncs.com/index.html"
    ></iframe>
  </body>
</html>
```

错误码参考

错误码	描述
-1002	无效的AccessKey ID。
-1003	无权限访问该文档。
-1004	无效的STS Token。
-1005	STS Token已过期。
-1006	签名错误。
-1007	OSS Bucket未配置跨域。
-1008	Refers配置错误。
-1100	其他原因造成的错误。

常见问题处理

如下方案仅为建议的通用解决方案，不同的应用场景请根据实际分析和处理问题。

常见问题	解决方案
iframe内部预览界面被遮挡问题	• 禁止iframe滚动，设置iframe属性scrolling="no"（预览界面实现了局部滚动，可以放心把该属性设置为no）。 • 通过flex布局来给iframe弹性高度或者通过js计算设置iframe一个具体高度值，不能用百分比。

常见问题	解决方案

父级窗口缩放导致预览界面变形问题	遇到此问题则需要父级窗口禁用浏览器自带的缩放功能。
	● Android平台
	通过设置viewport的user-scalable属性来禁止父级窗口缩放，示例代码如下：
	`<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,minimum-scale=1.0,user-scalable=no">`
	● iOS平台
	禁用双击缩放手势，示例代码如下：
	`body { touch-action: manipulation; }`
	禁止双指缩放手势，示例代码如下：
	`document.addEventListener('gesturestart', function(e){ e.preventDefault(); });`

PPT预览全屏按钮在某些浏览器中消失	只需在iframe上添加allowfullscreen属性即可修复，添加此属性后PPT文件可支持全屏预览。
IE下通过location.href | window.open打开预览URL时，如果URL中包含复制功能参数&copy=1，则会被当做HTML字符实体解析为版权符号©=1，导致预览不成功	由于IE的经典bug导致此问题，此时有如下两种解决方案： ● 方式一：使用IE浏览器时在跳转前把&转义为HTML字符实体&。 ● 方式二：将&copy=1的位置调整为紧跟URL的第一个参数，例如?copy=1&...即可解决，推荐使用此方式。

5.3.2. 文档格式转换

使用文档格式转换功能，您可以将支持的输入文档类型转换为指定的输出文档类型，便于上层应用更好的使用文档内容。

功能

将支持的输入文档类型转换为指定的输入文档类型，格式转换的输出文档保存在请求中指定的OSS路径中。

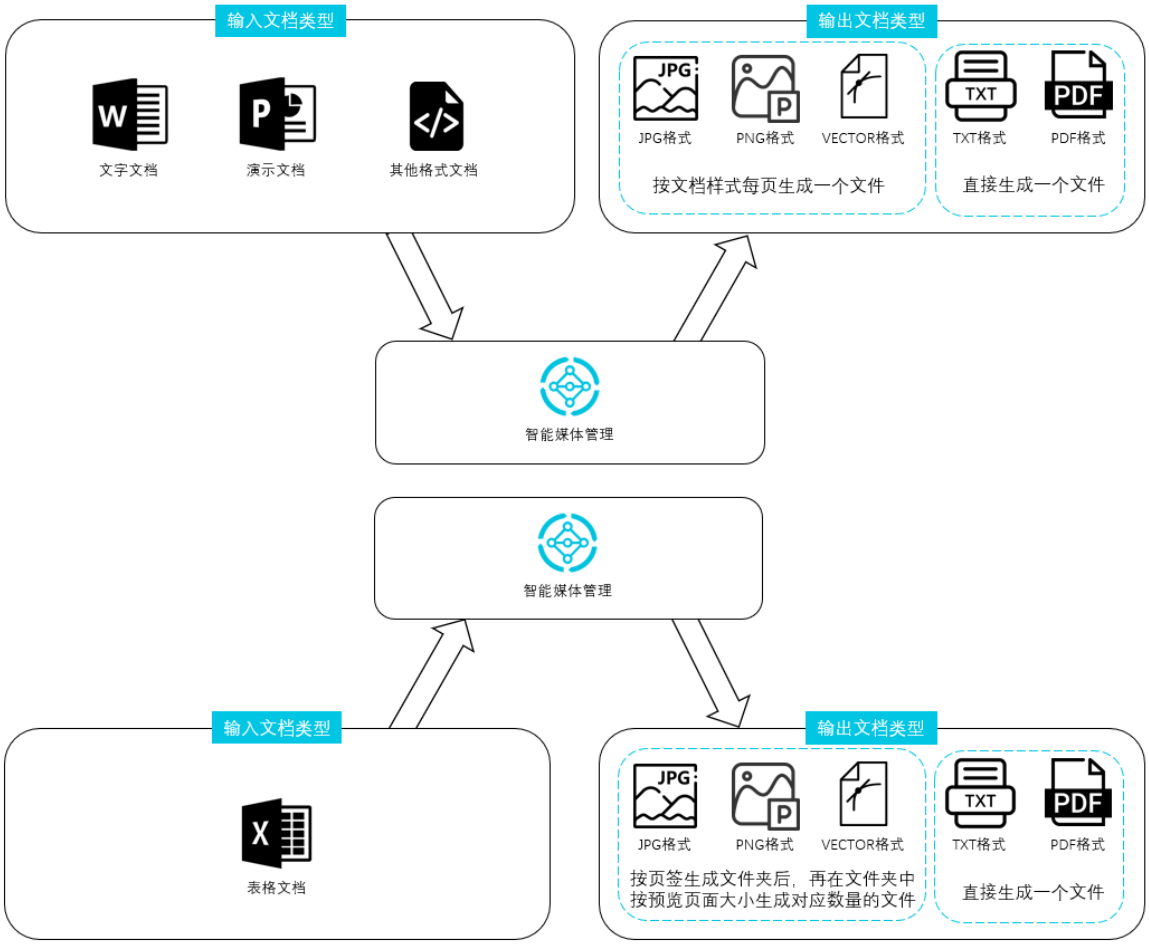
- 目前支持的输入文档类型包含如下格式：
  - 文字文档：DOC、DOT、WPS、WPT、DOCX、DOTX、DOCM、DOTM
  - 演示文档：PPTX、PPT、POT、POTX、PPS、PPSX、DPS、DPT、PPTM、POTM、PPSM
  - 表格文档：XLS、XLT、ET、ETT、XLSX、XLTX、CSV、XLSB、XLSM、XLTM

- 其他格式文档：PDF、LRC、C、CPP、H、ASM、S、JAVA、ASP、BAT、BAS、PRG、CMD、RTF、TXT、LOG、XML、HTM、HTML

- 目前支持的输出文档类型包含JPG、PNG、PDF、TXT、VECTOR格式。

根据请求中指定的输出文档类型会有不同的输出内容，输出内容示例的更多信息，请参见输出内容示例。

- 对于JPG、PNG和VECTOR格式的输出文档类型，根据输入文档类型是否为表格文档，输出内容不同。
  - 当输入文档类型为表格文档时，按表格中的页签生成对应数量的文件夹后，再按预览页面大小生成对应数量的文件。
  - 当输入文档类型为非表格文档（例如文字文档、演示文档）时，按文档样式每页生成一个文件。
- 对于PDF和TXT格式的输出文档类型，无论输入文档类型是否为表格文档，直接生成一个文件。



使用

文档格式转换所需的时间依赖于文档大小、文档页数、文档字数等因素，典型情况下在秒级可以完成请求，但是大文件、多页数、多字数的文档可能需要几十秒的时间。为了降低请求的应用等待时间，文档格式转换目前提供了异步请求接口。

您可以通过同步请求或者异步请求方式进行文档格式转换。

- 文档格式转换的同步请求接口为ConvertOfficeFormat。
- 文档格式转换的异步请求接口为CreateOfficeConversionTask。

输出内容示例

如下以请求中指定输出到OSS路径*/bucket1/imm-format-convert-tgt/session123/*为例说明转换为不同输出文档类型时的输出内容。

输出文档类型为JPG格式

- 当输入文档类型为表格文档时，输出内容示例如下所示。其中s1和s2为根据表格文档中的页签名称生成的文件夹。
  - /bucket1/imm-format-convert-tgt/session123/s1/1.jpg
  - /bucket1/imm-format-convert-tgt/session123/s1/2.jpg
  - /bucket1/imm-format-convert-tgt/session123/s1/[...].jpg
  - /bucket1/imm-format-convert-tgt/session123/s2/1.jpg
  - /bucket1/imm-format-convert-tgt/session123/s2/2.jpg
  - /bucket1/imm-format-convert-tgt/session123/s2/[...].jpg
- 当输入文档类型为非表格文档时，输出内容示例如下所示。
  - /bucket1/imm-format-convert-tgt/session123/1.jpg
  - /bucket1/imm-format-convert-tgt/session123/2.jpg

输出文档类型为PNG格式

- 当输入文档类型为表格文档时，输出内容示例如下所示。其中s1和s2为根据表格文档中的页签名称生成的文件夹。
  - /bucket1/imm-format-convert-tgt/session123/s1/1.png
  - /bucket1/imm-format-convert-tgt/session123/s1/2.png
  - /bucket1/imm-format-convert-tgt/session123/s1/[...].png
  - /bucket1/imm-format-convert-tgt/session123/s2/1.png
  - /bucket1/imm-format-convert-tgt/session123/s2/2.png
  - /bucket1/imm-format-convert-tgt/session123/s2/[...].png
- 当输入文档类型为非表格文档时，输出内容示例如下所示。
  - /bucket1/imm-format-convert-tgt/session123/1.png
  - /bucket1/imm-format-convert-tgt/session123/2.png

输出文档类型为PDF格式，无论输入文档类型是否为表格文档，输出内容均相同，输出内容示例为/bucket1/imm-format-convert-tgt/session123/1.pdf。

输出文档类型为TXT格式，无论输入文档类型是否为表格文档，输出内容均相同，输出内容示例为/bucket1/imm-format-convert-tgt/session123/1.txt。

输出文档类型为VECTOR格式

- 当输入文档类型为表格文档时，输出内容示例如下所示。其中meta.json文档是元数据，s1/meta.json是sheet1的元数据，s1/fp1.json、s1/fp2.json等文档是sheet1每页的渲染信息。
  - /bucket1/imm-format-convert-tgt/session123/doc/meta.json
  - /bucket1/imm-format-convert-tgt/session123/doc/s1/meta.json
  - /bucket1/imm-format-convert-tgt/session123/doc/s1/fp1.json
  - /bucket1/imm-format-convert-tgt/session123/doc/s1/fp2.json
  - /bucket1/imm-format-convert-tgt/session123/doc/s1/fp[...].json
  - /bucket1/imm-format-convert-tgt/session123/doc/s2/meta.json
  - /bucket1/imm-format-convert-tgt/session123/doc/s2/fp1.json

- /bucket1/imm-format-convert-tgt/session123/doc/s2/fp2.json
- /bucket1/imm-format-convert-tgt/session123/doc/s2/fp[...].json

- 当输入文档类型为非表格文档时，输出内容示例如下所示。其中meta.json文档是元数据，fp1.json、fp2.json等文档是每页的渲染信息，l/1、l/2等文档是渲染使用的具体内容。
  - /bucket1/imm-format-convert-tgt/session123/doc/meta.json
  - /bucket1/imm-format-convert-tgt/session123/doc/fp1.json
  - /bucket1/imm-format-convert-tgt/session123/doc/fp2.json
  - /bucket1/imm-format-convert-tgt/session123/doc/fp[...].json
  - /bucket1/imm-format-convert-tgt/session123/doc/l/1
  - /bucket1/imm-format-convert-tgt/session123/doc/l/2
  - /bucket1/imm-format-convert-tgt/session123/doc/l/[...]

5.3.3. JavaScript API预览高级功能配置

当使用JavaScript API预览方式时，您可以实现全屏预览、页码跳转控制、预览界面UI自定义、加密文档处理、签名刷新、重试、广告配置等功能以及获取到预览文档渲染的相关信息，方便查看相关数据。

全屏预览

在嵌套预览引擎的父页面中的iframe上添加allowfullscreen属性，即可支持全屏预览。

```
<iframe id="preview" src=[PreviewURL] allowfullscreen frameborder="0" scrolling="no"></ifra
me>
```

启用全屏预览方法：

```
function reqFullScreen(){
      var el = $('.preview')[0];
      var fn = el.requestFullScreen || el.webkitRequestFullScreen || el.mozRequestFullScree
n || el.msRequestFullScreen;
      fn.call(el);
}
```

控制文字或演示文档页码跳转

```
//跳转到某一页。
sendMessage('logic.toPage', {
    pageIndex: 0 //跳转页码，从0开始。
});
//跳转到上一页。
sendMessage('logic.prevPage');
//跳转到下一页。
sendMessage('logic.nextPage');
```

自定义文字或演示预览界面UI

```
sendMessage('setConfig', {
```

```
sendMessage('setConfig', {
  writerCustomStyle: function(isMobile) {
    if (!isMobile) {
      return {
        //预览容器上边距。
        containerMarginTop: 30,
        //预览容器下边距。
        containerMarginBottom: 30,
        //容器背景色。
        containerBackground: '#000000',
        //页与页之间的距离。
        pageSpacing: 20,
        //每页的阴影样式。
        pageShadow: '0px 0px 6px 0px rgba(0, 0, 0, 0.3)',
        //每页的边框样式。
        pageBorder: 'none',
        //放大缩小按钮容器样式。
        scale: {/* style注意这里的样式写法跟css文件样式写法一样，比如vertical-align:middle */},
        //放大缩小按钮样式。
        scaleBtn: {/* style注意这里的样式写法跟css文件样式写法一样，比如vertical-align:middle */
},
        //缩小按钮样式。
        scaleShrink: {/* style注意这里的样式写法跟css文件样式写法一样，比如vertical-align:middle
*/},
        //缩小按钮字符。
        scaleShrinkText: '',
        //缩小按钮不可点击样式。
        scaleShrinkDisable: {/* style注意这里的样式写法跟css文件样式写法一样，比如vertical-align
:middle */},
        //放大按钮样式。
        scaleMagnify: {/* style注意这里的样式写法跟css文件样式写法一样，比如vertical-align:middl
e */},
        //放大按钮字符。
        scaleMagnifyText: '',
        //放大按钮不可点击样式。
        scaleMagnifyDisable: {/* style注意这里的样式写法跟css文件样式写法一样，比如vertical-alig
n:middle * /},
        //缩放比例文字。
        scaleText: {/* style注意此处的样式写法与css文件样式写法相同，例如vertical-align:middle *
/},
        //缩放组件元素排序。
        scaleSort: ['shrink', 'text', 'magnify'] //shrink表示缩小，text表示文字，magnify表示放
大。
      }
    } else {
      return {
        pageBorder: '1px solid #d2d5d8'
      };
    }
  }
});
//演示预览自定义样式。
sendMessage('setConfig', {
  powerpointCustomStyle: function(isMobile) {
    if (!isMobile) {
```

```
    if (visible) {
      return {
        //隐藏翻页按钮。
        paginationDisplay: false,
        //隐藏全屏按钮。
        fullScreenButtonDisplay: false,
        //预览容器上边距。
        containerMarginTop: 30,
        //预览容器下边距。
        containerMarginBottom: 30,
        //容器背景色。
        containerBackground: '#000000'
      };
    } else {
      return {};
    }
  }
});
```

自定义错误提示界面

在iframe外框监听错误消息，自行处理报错的UI逻辑。

```
sendMessage("setConfig", {
  enableMessageUI: false
});
window.addEventListener('message', function(e) {
  var res = JSON.parse(e.data);
    switch(res.action) {
    case 'message.error':
      toast(i18n(res.data.result));
      break;
    }
}, false);
```

设置加密文件解密流程

监听message.error事件，处理展示密码输入相关UI逻辑，然后用户输入密码后，携带密码再次提交转换任务，服务端根据提交的密码执行解密流程，生成一个新的预览地址，浏览器重新设置iframe的src属性为新的预览地址。如果解密失败，子页面会重新抛出aliyunPasswordInvalid错误，重新输入密码重复以上流程即可。

```
sendMessage("setConfig", {
    enableCheckPasswordLogic: false
});
```

```
var isFirstTime = false;
window.addEventListener('message', function(e) {
  var res = JSON.parse(e.data);
    switch(res.action) {
    case 'message.error':
      if (res.data.result === 'aliyunPasswordInvalid') {
        if (isFirstTime) {
          //显示密码框。
          //your code。
          isFirstTime = false;
        } else {
          //提示密码错误。
          //your code。
        }
    }
      break;
    }
}, false);
```

刷新签名

当用户本地临时访问凭证过期时，无法继续预览文档，此时需要刷新签名。首先监听子页面抛出的logic.refreshToken事件，然后获取相关token信息后通过发送logic.setToken消息设置相关token信息，刷新iframe页面签名。

```
window.addEventListener('message', function(e) {
  var res = JSON.parse(e.data);
    switch(res.action) {
    case 'logic.refreshToken':
        //重新获取token的逻辑请自行实现。
      $.ajax(...).then(function(data){
        sendMessage('logic.setToken', {
          region: data.region,
          accessKeyId: data.accessKeyId,
          accessKeySecret: data.accessKeySecret,
          stsToken: data.stsToken,
          bucket: data.bucket
        });
      });
    break;
    ...
  }
}, false);
```

设置重试机制

如果浏览器预览出错，可以设置重试逻辑，尝试重新获取OSS预览文档信息，提高预览的成功率。

```
sendMessage('setConfig', {
  couldRetry: function(tryCount) {
    //tryCount当前已经重试的次数。
    return new Promise(function(resolve) {
      if (tryCount > 35) {
        //最多重试次数。
        resolve(false);
      } else {
        var tryDelay = 0;
        if (tryCount < 2) {
          // 0 - 1
          tryDelay = 250;
        } else if (tryCount <= 11) {
          // 2 - 11
          tryDelay = 500;
        } else {
          // 12 - 35
          tryDelay = 1000;
        }
        setTimeout(function () {
          resolve(true); //执行重试。
        }, tryDelay);
      }
    });
  }
});
```

自定义广告

注意 如果data为function，则函数中引用的变量必须在此之前通过setData传递到子页面。

```
//文字Word（移动端）。
sendMessage('setConfig', {
  wordAd: function(pageIndex, total) {
  //pageIndex表示当前页数，total表示总页数，接入方可以通过如下参数设计广告显示逻辑。
    return {
      type: 'iframe', //iframe或者img。
      src: iframeData.xxx, //函数中所有变量必须通过setData传递，并通过iframeData来使用。
      href: iframeData.xxx, //跳转地址。
      width: 1090, //宽度。
      height: 200, //高度。
      closeBtn: true //是否显示关闭广告按键。
    };
  }
});
//表格Excel（移动端和WEB端）。
sendMessage('setConfig', {
  etAd: function() {
    return {
      type: 'iframe',
      src: 'http://xxxx',
      href: '',
      width: 1090,
      height: 200,
      closeBtn: true
    }
  }
});
//演示PowerPonit（WEB端）。
sendMessage('setConfig', {
  wppAd: function() {
      return {
      type: 'iframe',
      src: 'http://xxxx',
      href: '',
      width: 1090,
      height: 200,
      closeBtn: true
    }
  }
});
```

校准本地签名时间

OSS SDK签名时默认会使用本地时间，如果用户本地时间与服务器时间相差超过15分钟则会导致签名验证失败，此时可手动设置强制本地使用服务器时间签名，而获取服务器时间的逻辑则由用户自行处理。

```
sendMessage("setConfig", {
  timestamp: {
    serverTime: 1540881763,  //服务器当前时间。单位为秒。
    expires: 3600 //签名有效期。默认值为1800，单位为秒。
  }
});
```

获取预览信息

在iframe外层监听message，可以得到预览页面抛出的信息，信息体为JSON格式字符串，例如{"action":"page.readPage", "data":{"pageIndex":5}}，其中action为事件名，data为抛出的数据。

```
window.addEventListener('message', function(e) {
  var res = JSON.parse(e.data);
  // res.action
  // res.data
});
```

记录当前浏览页码

监听action为page.readPage时，会返回当前浏览页数data.pageIndex，接入方自行存储该值后，下次打开相同页面只需将其赋值于iframe url中的pageIndex参数即可跳转到上次浏览的页码，以实现从历史记录继续浏览。

```
{
  "action": "page.readPage",
  "data": {
    "pageIndex": 0
  }
}
```

获取meta.json信息

```
{
  action: "preview.meta", //当前页面执行状态。
  data: {
    version: "xxx", //数据版本号。
    epc: "xxx", //预览渲染的页数。
    pc: "xxx",  //文档实际的页数。
    pro: "xxx",  //文件类型。
    ...
  }
}
```

错误事件说明

当发生错误时，父级窗口会接收到如下数据结构JSON字符串消息。

```
{
  action: 'message.error',
  data: {
    result: 'aliyunOpenFileFail',
    data: {
    //错误细节。
    }
  }
}
```

其中action的值固定为message.error，result的值为具体错误类型的说明，详细说明请参见下表。

错误类型	说明
aliyunOpenFileFail	打开文档失败。
aliyunUnsupportFile	不支持的文件类型。
aliyunRequstFail	前端请求静态资源失败，包括签名错误等，具体信息会有data数据返回。
aliyunQueryParamInvalid	页面必传参数缺失。
aliyunRequestTimeout	资源加载超时
aliyunRenderEngineTooOld	js渲染引擎版本过旧。
aliyunPasswordInvalid	密码错误，需要处理密码重试的逻辑。

获取渲染页面关键信息

首先父级窗口会接收到以下数据结构JSON字符串消息。

```
{
  action: 'page.getBaseInfoFromServerDone', //当前页面执行状态。
  data: {
    startTime: 1111,//开始时间。
    endTime: 1111, //结束时间。
    useTime: 1111, //花费时间。
    pageStartTime: 1111 //预览页面开始时间。
  }
}
```

说明

- 所有page.*Done格式的事件data都包含{startTime: 开始时间, endTime: 结束时间, useTime: 花费时间, pageStartTime: 页面开始时间}参数，时间的单位均毫秒。
- 下表描述的事件中，page.xxxDone均会有page.xxx事件，page.xxx事件为page.xxxDone开始执行的事件，page.xxx中相对于page.xxxDone缺少endTime和useTime参数。

事件	说明
getBaseInfoFromServerDone	获取页面基本信息。
getSheetInfoFromServerDone	获取表格基础信息。
getPageInfoFromServerDone	获取单页信息{pageIndex: 0, ...}。
firstRendered	首屏渲染完成{startTime, pageStartTime}。
timing	更多信息，请参见Timing格式。

timing格式如下所示。

```
//t为performance.timing。
{
  // 页面加载完成的时间。
  loadPage : t.loadEventEnd - t.navigationStart,
  //dom解析事件。
  domReady : t.domComplete - t.responseEnd,
  //重定向的时间。
  redirect : t.redirectEnd - t.redirectStart,
  //DNS查询时间。
  lookupDomain : t.domainLookupEnd - t.domainLookupStart,
  //TTFB即Time To First Byte。
  ttfb : t.responseStart - t.navigationStart,
  //内容加载完成的时间。
  request : t.responseEnd - t.requestStart,
  //执行onload回调函数的时间。
  loadEvent : t.loadEventEnd - t.loadEventStart,
  //DNS缓存时间。
  appcache : t.domainLookupStart - t.fetchStart,
  //卸载页面的时间。
  unloadEvent : t.unloadEventEnd - t.unloadEventStart,
  //TCP建立连接完成握手的时间。
  connect : t.connectEnd - t.connectStart,
  timing: t
}
```

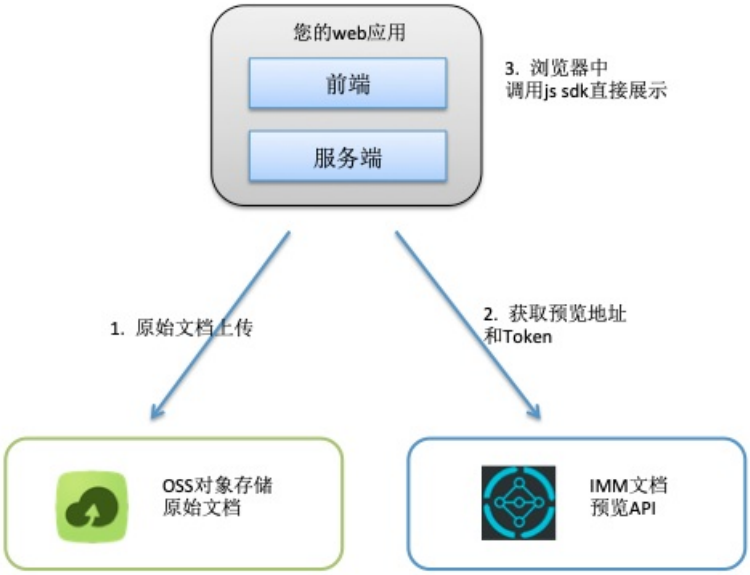
5.4. 文档预览V2

5.4.1. 快速入门

获取文档的预览地址和AccessToken后，无需指定iframe元素，JS文件会自动在自定块元素下生成iframe，并通过JS文件设置AccessToken即可快捷的实现文档预览。文档预览时支持自定义配置，包括组件状态、事件、文字相关、表格相关等配置。

预览原理

获取文档的预览地址和AccessToken后，无需指定iframe元素，JS文件会自动在自定块元素下生成iframe，并通过JS文件设置AccessToken即可快捷的实现文档预览。



准备工作

当访问的OSS Bucket域名与预览引擎的域名不同时，需要在OSS控制台将预览服务域名添加到存储转换后文档的OSS Bucket的跨域访问列表中。具体操作，请参见设置跨域访问。

预览流程

1. 上传原始文档到对象存储OSS中。具体操作，请参见上传文件。
2. 在服务端分别封装GetWebofficeURL和RefreshWebofficeToken接口。
3. 通过JS-SDK将返回的预览地址挂载到HTML块状元素中并设置AccessToken。

服务端封装接口

在服务端需要分别封装GetWebofficeURL和RefreshWebofficeToken接口，用于获取编辑地址和AccessToken，方便前端直接调用。

1. 调用GetWebofficeURL接口，获取协作地址。返回结果示例如下：

```
{
  "RefreshToken": "306ffe61897443c8909aaad325********",
  "RequestId": "D807F412-1EE0-44E0-A60F-47C27B4046CF",
  "AccessToken": "7f9b6ba5baef4c7d80f837d4c9********",
  "RefreshTokenExpiredTime": "2021-04-21T06:48:10.161355914Z",
  "WebofficeURL": "https://office-cn-shanghai.imm.aliyuncs.com/office/p/9eea88df758a75b
6308358500a9e141ccf3b7629?_w_tokentype=1",
  "AccessTokenExpiredTime": "2021-04-20T07:18:10.161355914Z"
}
```

2. 调用RefreshWebofficeToken接口，刷新AccessToken。

AccessToken具有时效性，当过期后前端需要调用服务端RefreshWebofficeToken接口重新刷新AccessToken，所以需要在服务端封装此接口。调用此接口的返回结果格式和调用GetWebofficeURL接口的相同。

前端JS-SDK使用

1. 引入JS-SDK。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0
,user-scalable=no">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Demo</title>
</head>
<body>
  <script src="https://g.alicdn.com/IMM/office-js/${x.y.z}/aliyun-web-office-sdk.min.js
"></script>
  <script>
    console.log('引入后可以开始使用JS-SDK了！');
    console.log(aliyun); //全局变量名。
  </script>
</body>
</html>
```

以上示例中${x.y.z}表示最新版本号，请根据实际填写，最新版本请参见版本。

2. 接入WebOffice。

通过在服务端封装的GetWebofficeURL接口获取tokenInfo对象。假设tokenInfo对象和调用GetWebofficeURL接口返回的结构一致。

如下示例以服务端封装的GetWebofficeURL接口为 `/getTokenInfo` 举例说明。

```
//获取协作地址和AccessToken。
var tokenInfo = await $.get('http://example.com/getTokenInfo')

let instance = aliyun.config({
  url: tokenInfo.WebofficeURL //设置文档协作URL地址。
})
//设置AccessToken。
instance.setToken({
  token: tokenInfo.AccessToken
})
```

3. 自定义Office(iframe)挂载点。

> **说明**
>
> 在DOMContentLoaded事件被触发后，请确保挂载节点存在再执行初始化操作。

iframe(#iframe)默认会挂载到body下，可根据需要自定义iframe(#iframe)的挂载点。

```
<div id="container"></div>
```

```
aliyun.config({
  mount: document.querySelector('#container'),
  url: '文档协作URL地址' //即步骤2示例中的文档协作URL地址（tokenInfo.WebofficeURL）。
})
```

如果需要对iframe对象做特殊处理，可以通过JS-SDK实例化对象快速获取到iframe的DOM（Document Object Model）对象。

```
var instance = aliyun.config({
    mount: document.querySelector('#container')
   //...
})
console.log(instance.iframe)
```

4. 设置令牌（Token）。
   - 在获取协作地址后，需要设置令牌才能在线协作。
   - 每次刷新令牌后，也需要通过此方法设置令牌。

```
//根据业务需求通过异步请求或者模板输出的方式获取token。
var token = 'yourToken';
//设置token。
instance.setToken({
  token: token,
  timeout: 10 * 60 * 1000, //必须设置。token的超时时间，单位为ms。以10分钟示例说明。注意：JS-SDK会提前5分钟调用刷新token方法，所以设置的 timeout 应在10分钟以上（10 * 60 * 1000 以上，单位为毫秒），避免刷新过快
})
```

5. 超时更新令牌（Token）。

   通过在服务端封装的RefreshWebofficeToken接口获取tokenInfo对象。假设tokenInfo对象和调用RefreshWebofficeToken接口返回的结构一致。

   您可以通过传入获取Token的函数，在Token超时时，JS-SDK会自动调用传入的函数重新获取Token，返回一个promise或者object。

   如下示例以服务端封装的GetWebofficeToken接口为 `/refreshTokenInfo` 举例说明。

```
//缓存上次的tokenInfo，用于刷新token。
  let lastTokenInfo = tokenInfo

//获取token函数。
//注意：refreshToken 目前不支持 async，只支持返回 Promise 或者 {token,timeout} 对象
  const refreshToken = function() {
    return new Promise(function(resolve){
      //业务处理逻辑，调用服务端封装的refreshToken接口。
      $.get('http://example.com/refreshTokenInfo',{
        RefreshToken: lastTokenInfo.RefreshToken,
        AccessToken: lastTokenInfo.AccessToken,
        //....
      }).then(function(tokenInfo){
        lastTokenInfo = tokenInfo
        resolve({
          token: tokenInfo.AccessToken, //必须设置。
          timeout: 10 * 60 * 1000, //必须设置。token超时时间，单位为ms。可配合refreshToken配置函数使用，在超时前调用refreshToken重新刷新token。
        })
      })
    })
  }
//配置超时获取token函数。
aliyun.config({
  //...
  refreshToken
})
```

5.4.2. 支持的文档格式

V2版本文档预览提供文档预览和文档在线协作两种功能，本文将介绍两种功能支持的文档类型。

文档预览

文档预览支持的文档类型和格式如下表所示。

文档类型	支持格式
文字	DOC、DOT、WPS、WPT、DOCX、DOTX、DOCM、DOTM、RTF、TXT
表格	XLS、XLT、ET、XLSX、XLTX、CSV、XLSM、XLTM
演示	PPTX、PPT、POTX、PPS、PPSX、DPS、DPT、PPTM、POTM、PPSM
PDF	PDF

文档在线协作

文档在线协作支持的文档类型和格式如下表所示。

文档类型	支持格式
文字	DOC、DOT、WPS、WPT、DOCX、DOTX、DOCM、DOTM、TXT
表格	XLS、XLT、ET、XLSX、XLTX、XLSM、XLTM
演示	PPTX、PPT、POTX、PPS、PPSX、DPS、DPT、PPTM、POTM、PPSM
PDF	PDF

5.4.3. 自定义配置

5.4.3.1. 事件监听

为组件开启事件监听，当组件发生指定事件时，系统执行相应的操作。

实现方法

通过实例的on方法监听事件。

```
demo.on('事件名', function(data) {
    // do something...
})
```

事件列表

参数说明请参见下表。

事件名	说明	最低支持版本
fileOpen	文档打开，用于文档打开成功或者失败时的事件回调。	1.0.0
error	错误事件，用于错误发生时的事件回调。	1.1.2
tabSwitch	PC头部tab切换，用于tab切换时的事件回调。	1.1.2
fileStatus	文件保存状态，用于文件保存时的事件回调。	1.1.5
previewLimit	预览页数限制事件，用于滚动到限制页数底部时的事件回调。 此事件只在预览页数限制模式下有效，且只支持WORD、PDF以及PDF组件。	1.1.5

事件名	说明	最低支持版本
hasDocMap	文档是否存在目录，用于文档存在目录时的事件回调。 此事件只支持文字组件。 由于文字获取目录是动态分片获取，因此需要以监听事件方式确定是否存在目录。	1.1.5
fullscreenchange	进入或退出全屏事件，用于进入或退出全屏时的事件回调。	1.1.5

示例

- 文件打开
  - 示例

```
demo.on('fileOpen', function(data) {
    // do something...
})
```

  - 当事件执行成功时，返回结果如下：

```
{
    "success": true,
    "time": 111, // 打开时长。
    "fileInfo": {
      name: "座位表",
      officeType: "w",
      id: "64784347535"
    }
}
```

  - 当事件执行失败时，返回结果如下：

```
{
    "success": true,
    "time": 111,
    "reason": "InvalidLink", //错误时会有错误码。
}
```

- 错误事件
  - 示例

```
demo.on('error', function(data) {
    // do something...
})
```

  - 返回结果

```
```javascript
{
    "reason": "InvalidLink", //错误时会有错误码。
}
```
```

更多错误码说明请参见错误码。

- PC头部tab切换
  - 示例

```
demo.on('tabSwitch', function(data) {
    // do something...
})
```

  - 返回结果

```
{
  tabKey: 1 //当前tab的序号。
}
```

- 文件保存状态
  - 示例

```
demo.on('fileStatus', function() {
    // do something...
})
```

  - 返回结果

```
{
  status: 0, //文档无更新。
  status: 1, //版本保存成功。当手动保存、定时保存、关闭网页时触发。
  status: 2, //暂不支持保存空文件。当内核保存完后文件为空时触发。
  status: 3, //空间已满。
  status: 4, //保存中请勿频繁操作。当服务端处理保存队列已满，正在排队时触发。
  status: 5, //保存失败。
  status: 6, //文件更新保存中。当修改文档内容触发的保存时触发。
  status: 7, //保存成功。当文档内容修改保存成功时触发。
}
```

- 预览页数限制事件
  - 示例

```
demo.on('previewLimit', function(result) {
    // do something...
})
```

- 返回结果

> **说明** 由于文字文档为流式排版，无法获取准确的真实页数，因此返回结果中只有一个total参数。

```
//文字文档。
{
    total: 4 //限制页数。
}
//PDF和演示文档。
{
    total: 4 //限制页数。
    realTotal: 10 //真实总页数。
}
```

- 文档是否存在目录

  示例如下：

```
demo.on('hasDocMap', function() {
    //文档存在目录时，则会执行回调。
})
```

- 进入或退出全屏事件
  - 示例

```
demo.on('fullscreenchange', , function(result) {
    // do something...
})
```

  - 返回结果

```
{
  status: 0, //退出全屏时触发。
  status: 1, //进入全屏时触发。
}
```

错误码

错误码	说明
Unknown	未知错误。
userNotLogin	用户未登录。
InvalidLink	无效链接。
SessionFull	协作成员已满。
Fail	打开失败。
PermissionDenied	您的操作权限不足。

错误码	说明
GetTokenFailed	获取Token失败。
SessionExpired	会话过期。
fileTooLarge	文件过大，不支持预览服务。
GetFileInfoFailed	获取文件信息失败。

5.4.3.2. 高级用法

通过调用高级接口可用于直接操作文档。本文中的demo对象是指JS-SDK实例化后的对象。

> **说明** JS-SDK提供的接口风格与VBA一致，原则上兼容VBA的接口和参数。

调用流程

1. 等待ready。

```
await demo.ready() //等待demo ready后再调用高级接口。
```

2. 获取文档类型的应用对象。

   根据WordApplication、ExcelApplication、PPTApplication、PDFApplication判断当前的文档类型。

```
//文档类型为文字类型。
const wordApp = demo.WordApplication()
//文档类型为表格类型。
const excelApp = demo.ExcelApplication()
//文档类型为演示类型。
const pptApp = demo.PPTApplication()
//文档类型为PDF类型。
const pdfApp = demo.PDFApplication()
//自动识别文档类型。
const app = demo.Application
```

3. 调用高级接口。

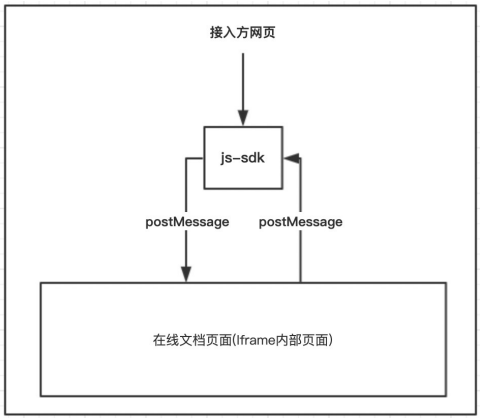
   此处以文字导出PDF为例介绍。

```
//将文字文档导出为PDF文档类型。
  async function exportPdf() {
    await wordApp.ActiveDocument.ExportAsFixedFormat()
  }
```

释放对象

由于iframe的跨域安全限制，无法直接调用iframe内部的函数和变量，JS-SDK提供的高级接口对象只是iframe内部对象的映射，通过postMessage进行通信，示意图如下图所示，所以部分对象需要手动释放。

> **说明** 需要手动释放的对象均会在文档中说明，请以文档说明为准。



系统支持单个或批量释放对象。

- 单个释放对象

  代码示例

```
const app = demo.ExcelApplication()
//例如表格获取WorkSheet对象。
const sheet = demo.Sheets.Item("sheet1")
//此时得到的sheet对象只是一个映射对象，它拥有iframe内部对象的所有属性和方法，该对象的所有属性和方法都会映射到iframe内部。
sheet.Activate() //切换sheet。
//...do something
//不需要该对象时，手动释放，此时会通知iframe内部销毁对应的对象。
sheet.Destroy()
```

- 批量释放对象

  当有多个对象需要销毁时，可以按照区间批量释放对象。

  代码示例

```
const stack1 = app.Stack()
//do something
stack1.End() //释放在stack1区间的对象。
```

5.4.3.3. 文字相关

对文字文档进行操作，如导出为PDF文档，获取页码和跳转页等。

导出为PDF

```
 /*
  * 目前只支持两个参数RangeType和FrameSlides。
  * @param: { Range?: WdExportRange, From?: number, To?: number, Item?: WdExportItem, Inclu
deDocProps?: bool }
  * WdExportRange: {
  *      wdExportAllDocument: 0,
  *      wdExportCurrentPage: 2,
  *      wdExportFromTo: 3,
  *      wdExportSelection: 1,
  *  },
  * WdExportItem: {
  *      wdExportDocumentContent: 0,
  *      wdExportDocumentWithMarkup: 7
  *  }
  */
  await demo.WordApplication().ActiveDocument.ExportAsFixedFormat()
```

获取页码和跳转页

定义后续示例需要用到的变量。

```
const app = demo.WordApplication()
const {Enum} = app
```

- 获取总页数

> **说明** 由于文字文档是流式排版，无法在开始确定最终页数，只有浏览到最后才能知道总页数。

```
 /*
  * @param: WdInformation: {
  *      wdNumberOfPagesInDocument: 4
  *  }
  * @return: {PagesCount: number, End: boolean}
  */
  let totalPages = await app.ActiveDocument.Range.Information(Enum.WdInformation.wdNumber
OfPagesInDocument)
  if (totalPages.End) {
    console.log("加载完了！一共", totalPages.PagesCount, "页")
  }
```

- 获取当前页

```
 /*
  * @param: WdInformation: {
  *      wdActiveEndPageNumber: 3
  *  }
  * @return: number
  */
  let currentPage = await app.ActiveDocument.Selection.Information(Enum.WdInformation.wdA
ctiveEndPageNumber)
```

- 跳转到指定页

> **说明** 由于文字文档是流式排版，文字文档较大时，跳转时间会比较长，建议加一个中间loading过渡效果。

```
 /*
  * @param: { What?: WdGoToItem, Which?: WdGoToDirection.wdGoToAbsolute, Count?: number,
Name?: string}
  * WdGoToItem: {
  *      wdGoToPage: 1,
  *  }
  *  WdGoToDirection: {
  *      wdGoToAbsolute: 1
  *  }
  */
  await app.ActiveDocument.Selection.GoTo(Enum.WdGoToItem.wdGoToPage, Enum.WdGoToDirectio
n.wdGoToAbsolute, 10)
  //或者使用如下代码。
  await app.ActiveDocument.Selection.GoTo({
    What: Enum.WdGoToItem.wdGoToPage,
    Which: Enum.WdGoToDirection.wdGoToAbsolute,
    Count: 10
  })
```

- 当前页改变事件

```
  function eventHandle() {
    // do something
  }
  //监听当前页改变事件。
  app.Sub.CurrentPageChange = eventHandle
  //销毁事件监听。
  app.Sub.CurrentPageChange = null
```

5.4.3.4. PDF相关

对PDF文档进行操作，如获取页码和跳转页，设置页面渲染模式等。

获取页码和跳转页

- 获取总页数

```
 /*
  * @return: number
  */
  let totalPages = await demo.PDFApplication().ActivePDF.PagesCount
```

- 获取当前页

```
 /*
  * @return: number
  */
  let totalPages = await demo.PDFApplication().ActivePDF.CurrentPage
```

- 跳转到相应页

```
/*
 * @param : { PageNum: number }
 */
 let PageNum = 10
 await demo.PDFApplication().ActivePDF.JumpToPage({PageNum})
```

- 当前页改变事件

```
function eventHandle() {
  // do something
}
//监听当前页改变事件。
demo.PDFApplication().Sub.CurrentPageChange = eventHandle
//销毁事件监听。
demo.PDFApplication().Sub.CurrentPageChange = null
```

设置页面渲染模式

从V1.1.2版本开始支持设置页面渲染模式。

```
/*
* 1：单页模式
* 0：多页模式
*/
//设置为单页模式。
demo.PDFApplication().ActivePDF.PageMode = 1
```

5.4.3.5. 演示相关

对演示文档进行操作，如导出为PDF文档，获取页码和跳转页，幻灯片动画控制等。

导出为PDF

```
/*
 * 目前只支持两个参数RangeType以及FrameSlides
 * @param: { RangeType?: MsoTriState, FrameSlides?: MsoTriState }
 * MsoTriState: {
 *      msoFalse: 0,
 *      msoTrue: -1
 *  }
 */
 await demo.PPTApplication().ActivePresentation.ExportAsFixedFormat()
```

获取页码和跳转页

- 获取总页数

```
/*
 * @return: number
 */
 let totalPages = await demo.PPTApplication().ActivePresentation.Slides.Count
```

- 获取当前页

```
 /*
  * @return: number
  */
  let totalPages = await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.Sl
ide.SlideIndex
```

- 跳转到指定页

```
 /*
  * @param: number
  */
  //跳转到第三页。
  await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.GotoSlide(3)
```

- 当前页改变事件

```
  function eventHandle() {
    // do something
  }
  //监听当前页改变事件。
  demo.PPTApplication().Sub.SlideSelectionChanged = eventHandle
  //销毁事件监听。
  demo.PPTApplication().Sub.SlideSelectionChanged = null
```

幻灯片动画控制

从V1.1.2版本开始支持幻灯片播放状态以及幻灯片动画控制。

- 切换到幻灯片播放状态

```
await demo.PPTApplication().ActivePresentation.SlideShowSettings.Run()
```

- 退出幻灯片播放，切换到预览状态

```
await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.Exit()
```

- 幻灯片下一步动画

```
await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.GotoNextClick()
```

- 幻灯片上一步动画

```
await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.GotoPreClick()
```

- 获取当前幻灯片播放状态

```
/*
 * @return: string ('edit' | 'preview' | 'play')
 */
 let currentState = await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.S
tate
```

- 获取当前页幻灯片动画总步数

```
/*
 * @return: number
 */
 let clickCount = await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.Get
ClickCount()
```

- 获取当前页幻灯片动画当前步数

```
/*
 * @return: number
 */
 let clickCount = await demo.PPTApplication().ActivePresentation.SlideShowWindow.View.Get
ClickIndex()
```

- 切换到幻灯片播放状态回调事件

```
function eventHandle() {
   // do something
 }
 //监听事件。
 demo.PPTApplication().Sub.SlideShowBegin = eventHandle
 //销毁事件监听。
 demo.PPTApplication().Sub.SlideShowBegin = null
```

- 退出幻灯片播放，切换到预览状态回调事件

```
function eventHandle() {
   // do something
 }
 //监听事件。
 demo.PPTApplication().Sub.SlideShowEnd = eventHandle
 //销毁事件监听。
 demo.PPTApplication().Sub.SlideShowEnd = null
```

- 幻灯片下一步动画回调事件

```
function eventHandle() {
   // do something
 }
 //监听事件。
 demo.PPTApplication().Sub.SlideShowOnNext = eventHandle
 //销毁事件监听。
 demo.PPTApplication().Sub.SlideShowOnNext = null
```

- 幻灯片上一步动画回调事件

```
function eventHandle() {
   // do something
 }
 //监听事件。
 demo.PPTApplication().Sub.SlideShowOnPrevious = eventHandle
 //销毁事件监听。
 demo.PPTApplication().Sub.SlideShowOnPrevious = null
```

5.4.3.6. 表格相关

对表格文档进行操作，如导出为PDF文档，获取sheet名称和切换sheet等。

导出为PDF

```
//将workbook导出为PDF文档。
await demo.ExcelApplication().ActiveWorkbook.ExportAsFixedFormat()
//将当前sheet的workbook导出为PDF文档。
await demo.ExcelApplication().ActiveWorkbook.ActiveSheet.ExportAsFixedFormat()
```

获取sheet名称和切换sheet

- 获取所有sheet名称

```
await demo.ready()
const app = demo.ExcelApplication()
const Names = []
//For(start, end, step, handle)
await app.For(1, app.Sheets.Count, 1, async (Index) => {
    Names.push(await app.Sheets.Item(Index).Name)
})
console.log(Names)
```

- 获取当前sheet名称

```
await demo.ready()
const app = demo.ExcelApplication()
const name = await app.ActiveSheet.Name
console.log('ActiveSheet:', name)
```

- 切换到指定sheet

```
await demo.ready()
const app = demo.ExcelApplication()
const sheetIndex = 1 //sheets序号，从1开始。
app.Sheets.Item(sheetIndex).Activate() //切换sheet。
```

- 切换sheet回调事件

```
await demo.ready()
const app = demo.Application
app.Sub.Worksheet_Activate = async function() {
    console.log("ActiveSheet:", await app.ActiveSheet.Name)
}
```

6.文档在线协作

6.1. 快速入门

文档在线协作（编辑）和文档预览的接入方式以及前端使用的JS-SDK均相同。后端接口获取AccessToken和编辑URL的方式不同。

服务端封装接口

在服务端需要分别封装GetWebofficeURL和RefreshWebofficeToken接口，用于获取编辑地址和AccessToken，方便前端直接调用。

1. 调用GetWebofficeURL接口，获取协作地址。返回结果示例如下：

```
{
  "RefreshToken": "306ffe61897443c8909aaad325********",
  "RequestId": "D807F412-1EE0-44E0-A60F-47C27B4046CF",
  "AccessToken": "7f9b6ba5baef4c7d80f837d4c9********",
  "RefreshTokenExpiredTime": "2021-04-21T06:48:10.161355914Z",
  "WebofficeURL": "https://office-cn-shanghai.imm.aliyuncs.com/office/p/9eea88df758a75b
6308358500a9e141ccf3b7629?_w_tokentype=1",
  "AccessTokenExpiredTime": "2021-04-20T07:18:10.161355914Z"
}
```

2. 调用RefreshWebofficeToken接口，刷新AccessToken。

   AccessToken具有时效性，当过期后前端需要调用服务端RefreshWebofficeToken接口重新刷新AccessToken，所以需要在服务端封装此接口。调用此接口的返回结果格式和调用GetWebofficeURL接口的相同。

前端JS-SDK使用

1. 引入JS-SDK。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0
,user-scalable=no">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Demo</title>
</head>
<body>
  <script src="https://g.alicdn.com/IMM/office-js/${x.y.z}/aliyun-web-office-sdk.min.js
"></script>
  <script>
    console.log('引入后可以开始使用JS-SDK了！');
    console.log(aliyun); //全局变量名。
  </script>
</body>
</html>
```

以上示例中${x.y.z}表示最新版本号，请根据实际填写，最新版本请参见版本。

2. 接入WebOffice。

   通过在服务端封装的GetWebofficeURL接口获取tokenInfo对象。假设tokenInfo对象和调用GetWebofficeURL接口返回的结构一致。

   如下示例以服务端封装的GetWebofficeURL接口为 `/getTokenInfo` 举例说明。

```
//获取协作地址和AccessToken。
var tokenInfo = await $.get('http://example.com/getTokenInfo')

let instance = aliyun.config({
  url: tokenInfo.WebofficeURL //设置文档协作URL地址。
})
//设置AccessToken。
instance.setToken({
  token: tokenInfo.AccessToken
})
```

3. 自定义Office(iframe)挂载点。

   > **说明**
   >
   > 在DOMContentLoaded事件被触发后，请确保挂载节点存在再执行初始化操作。

   iframe(#iframe)默认会挂载到body下，可根据需要自定义iframe(#iframe)的挂载点。

```
<div id="container"></div>
```

```
aliyun.config({
  mount: document.querySelector('#container'),
  url: '文档协作URL地址' //即步骤2示例中的文档协作URL地址（tokenInfo.WebofficeURL）。
})
```

   如果需要对iframe对象做特殊处理，可以通过JS-SDK实例化对象快速获取到iframe的DOM（Document Object Model）对象。

```
var instance = aliyun.config({
    mount: document.querySelector('#container')
   //...
})
console.log(instance.iframe)
```

4. 设置令牌（Token）。
   - 在获取协作地址后，需要设置令牌才能在线协作。
   - 每次刷新令牌后，也需要通过此方法设置令牌。

```
//根据业务需求通过异步请求或者模板输出的方式获取token。
var token = 'yourToken';
//设置token。
instance.setToken({
  token: token,
  timeout: 10 * 60 * 1000, //必须设置。token的超时时间，单位为ms。以10分钟示例说明。注意：JS-SDK会提前5分钟调用刷新token方法，所以设置的 timeout 应在10分钟以上（10 * 60 * 1000 以上，单位为毫秒），避免刷新过快
})
```

5. 超时更新令牌（Token）。

   通过在服务端封装的RefreshWebofficeToken接口获取tokenInfo对象。假设tokenInfo对象和调用RefreshWebofficeToken接口返回的结构一致。

   您可以通过传入获取Token的函数，在Token超时时，JS-SDK会自动调用传入的函数重新获取Token，返回一个promise或者object。

   如下示例以服务端封装的GetWebofficeToken接口为 `/refreshTokenInfo` 举例说明。

```
//缓存上次的tokenInfo，用于刷新token。
  let lastTokenInfo = tokenInfo

//获取token函数。
//注意：refreshToken 目前不支持 async，只支持返回 Promise 或者 {token,timeout} 对象
  const refreshToken = function() {
    return new Promise(function(resolve){
      //业务处理逻辑，调用服务端封装的refreshToken接口。
      $.get('http://example.com/refreshTokenInfo',{
        RefreshToken: lastTokenInfo.RefreshToken,
        AccessToken: lastTokenInfo.AccessToken,
        //....
      }).then(function(tokenInfo){
        lastTokenInfo = tokenInfo
        resolve({
          token: tokenInfo.AccessToken, //必须设置。
          timeout: 10 * 60 * 1000, //必须设置。token超时时间，单位为ms。可配合refreshToken配置函数使用，在超时前调用refreshToken重新刷新token。
        })
      })
    })
  }
//配置超时获取token函数。
aliyun.config({
  //...
  refreshToken
})
```

6.2. 事件

为组件开启事件监听，当组件发生指定事件时，系统执行相应的操作。

实现方式

在JS-SDK中，通过实例的 `on` 方法监听事件。

```
instance.on('事件名', (data) => {
  //do something...
});
```

事件列表

可用于监听的事件请参见下表。

事件名	描述	最低支持版本
fileOpen	文件打开成功或者失败时的事件回调。	1.0.0
error	错误发生时的事件回调。	1.1.2
tabSwitch	PC头部Tab切换。	1.1.2
fileStatus	文件保存状态，用于文件保存的事件回调。	1.1.5
previewLimit	预览页数限制事件，用于滚动到限制页数底部时的事件回调。 此事件只在预览页数限制模式下有效，且只支持WORD、PDF以及PDF组件。	1.1.5
hasDocMap	文档是否存在目录。	1.1.5
fullscreenChange	进入或退出全屏事件。	1.1.5
fileNameChange	文件名重命名的事件回调。	1.1.9

fileOpen

文件打开成功或者失败时的事件回调。

- 示例

```
//文件打开成功或者失败时的事件回调
instance.on('fileOpen', (data) => {
  console.log('文件打开：', data);
});
```

- 当事件执行成功时，返回结果如下：

```
{
  duration: 812,
  fileInfo: {
    createTime: 1606461829,
    id: "94749723688",
    modifyTime: 1606461829,
    name: "userName",
    officeType: "s",
  },
  stageTime: 1614,
  success: true,
  time: 1614,
  ts: 1607858260164,
}
```

- 当事件执行失败时，返回结果如下：

```
{
  msg: "Fail",
  result: "Fail"
}
```

error

错误发生时的事件回调。

例如将 `doc` 文件改成 `xls` 文件等操作，会引发报错。

- 示例

```
//错误发生时的事件回调
instance.on('error',(err)=>{
  console.log('发生错误：',err);
});
```

- 返回参数

```
{
  reason:"Fail"
}
```

tabSwitch

PC头部tab切换，用于tab切换时的事件回调。

- 示例

```
instance.on('tabSwitch', function(data) {
    //do something...
})
```

- 返回参数

```
{
  tabKey: 1 //当前tab的序号。
}
```

fileStatus

文件保存的事件回调。

- 示例

```
//文件保存的事件回调
jssdk.on('fileStatus', (data) => {
  console.log('文件保存：', data);
});
```

- 返回参数

```
{
  status: 0, //文件无更新
  status: 1, //版本保存成功，触发场景：手动保存、定时保存、关闭网页
  status: 2, //暂不支持保存空文件，触发场景：内核保存完后文件为空
  status: 3, //空间已满
  status: 4, //保存中请勿频繁操作，触发场景：服务端处理保存队列已满，正在排队
  status: 5, //保存失败
  status: 6, //文件更新保存中，触发场景：修改文件内容触发的保存
  status: 7, //保存成功，触发场景：文件内容修改保存成功
}
```

hasDocMap

文档是否存在目录。

示例

```
instance.on('hasDocMap', function() {
    //文档存在目录时，则会执行回调。
})
```

fullscreenchange

进入或退出全屏事件，用于进入或退出全屏时的事件回调。

- 示例

```
instance.on('fullscreenchange', , function(result) {
    //do something...
})
```

- 返回结果

```
{
  status: 0, //退出全屏时触发。
  status: 1, //进入全屏时触发。
}
```

fileNameChange

文件名重命名的事件回调。

- 示例

```
//文件名重命名的事件回调
jssdk.on('fileNameChange', (e) => {
  console.log('新的文件名为：', e.fileName);
});
```

- 返回参数

  当文件重命名（成功）后，会返回新文件名。

```
{ fileName: "NewWebOffice" }
```

错误码

错误码	说明
Unknown	未知错误。
userNotLogin	用户未登录。
InvalidLink	无效链接。
SessionFull	协作成员已满。
Fail	打开失败。
PermissionDenied	您的操作权限不足。
GetTokenFailed	获取token失败。
SessionExpired	会话过期。
fileTooLarge	文件过大，不支持预览服务。
GetFileInfoFailed	获取文件信息失败。

6.3. 兼容性

本文介绍文档在线协作支持的浏览器版本和低版本兼容方法。

支持的浏览器

IMM文档在线协作支持的浏览器和版本如下：

平台	支持浏览器	版本
iOS	Safari、QQ内置浏览器、QQ小程序、微信内置浏览器、微信小程序	iOS 10+
Android	QQ内置浏览器、QQ小程序、微信内置浏览器、微信小程序	Android 7+
Windows	Chrome、IE11	Chrome 80+，IE11（编辑功能不保证完全兼容）
Mac OSX	Chrome、Safari	Chrome 80+

说明

IMM会定期更新适配各平台的主流浏览器最新版本。

低版本兼容

在使用JS-SDK时，如果您在低版本浏览器（例如 `IE11` ）中使用 `Promise` 、 `async...await` 等语法，可能会出现报错。推荐您在项目中使用 `Webpack+Babel` 编译或者直接在 `HTML` 代码中引用 `polyfill` ：

```
<script src="https://cdn.bootcss.com/babel-core/5.8.35/browser.min.js"></script>
<script src="https://cdn.bootcss.com/babel-core/5.8.35/browser-polyfill.min.js"></script>

<!-- 注意添加text/babel，否则无法编译 -->
<script type="text/babel">
  // 具体代码。
</script>
```

示例：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1.0,maximum-scale=1.0,use
r-scalable=no" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>兼容低版本浏览器</title>
</head>

<body>

  <!-- 引用babel -->
  <script src="https://cdn.bootcss.com/babel-core/5.8.35/browser.min.js"></script>
  <script src="https://cdn.bootcss.com/babel-core/5.8.35/browser-polyfill.min.js"></script>

  <!-- 引用js-sdk -->
  <script src="skd 地址"></script>

  <!-- 注意添加text/babel，否则无法编译 -->
  <script type="text/babel">
    window.onload = function() {
      const instance = aliyun.config({
        url: '预览地址',
      });
      const test = async () => {
        await instance.ready();
        console.log('api ready');
      }
      instance.on('fileOpen', function(data) {
        test();
        console.log('打开成功');
      });
    }
  </script>
</body>

</html>
```

6.4. 基础用法

本文介绍文档在线协作如何通过初始化配置，开启不同的显示模式、配置时间监听和组件状态等。

显示模式

JS-SDK通过初始化配置选项，可以开启不同的显示模式。

- 示例

```
const instance= aliyun.config({
  mode: 'normal',
});
```

- 参数

属性	描述
mode	界面显示模式。取值范围如下： ○ normal（默认）：普通模式，展示所有功能界面。 ○ simple：极简模式，不显示头部和工具栏。

协作用户配置

通过 `cooperUserAttribute` 选项，可以控制协作用户头像是否显示以及控制用户光标颜色。

- 示例

```
aliyun.config({
  cooperUserAttribute: {
    isCooperUsersAvatarVisible: true, //是否显示协作用户头像
    cooperUsersColor: [{
      userId: 'xxx', //用户ID
      color: '#F65B90' //用户光标颜色
    }],
  },
});
```

- 参数

属性	类型	描述
isCooperUsersAvatarVisible	Boolean	协作用户头像显示切换。
cooperUsersColor	Array	设置协作用户光标颜色。 以JSON格式的数组表示，包括如下内容： ○ userId：用户ID，类型为string。 ○ color：用户光标颜色，类型为string。

- 动态更新

  以上配置只在初始化时生效，JS-SDK还提供了动态更新组件状态接口，目前只提供设置用户光标颜色接口，示例如下：

```
await instance.cooperUserAttribute([
  {
    isCooperUsersAvatarVisible: true, //是否显示协作用户头像
    cooperUsersColor: [{
      userId: 'xxx', //用户ID
      color: 'red', //用户光标颜色
    }],
  },
]);
```

事件监听配置

监听剪切板

需要在移动端APP从系统剪切板获取数据时，可以使用该接口。

> **注意**
>
> 目前仅支持移动端表格以及文字。

在文档粘贴时，可以通过传入**获取系统剪切板数据函数**调用传入函数，获取系统剪切板数据，返回一个 `Promise` 或者 `Object` ，示例如下：

```
//获取系统剪切板数据函数
const getClipboardData = () => {
  //自身业务处理……

  return Promise.resolve({
    text: 'xxx', //text格式数据
    html: 'xxx', //html格式数据，目前仅表格支持
    updateExternal: true, //是否从外部粘贴数据，为false则从内部剪切板获取
  });
};

//配置获取系统剪切板数据函数
aliyun.config({
  getClipboardData,
  //...
});
```

监听 Toast

初始化JS-SDK时，可以通过配置自定义 `toast` 函数，关闭原有 `toast` 提示并且获取相关自定义 `toast` 样式。

- 示例

```
//拦截toast函数
const onToast = ({
  msg, //提示信息
  action, //提示动作
}) => {
  //自身业务处理
};

//配置toast函数
aliyun.config({
  onToast,
  //...
});
```

- `action` 参数说明

action	描述
success	成功。
error	错误。
warn	警告。
close	关闭toast。

监听外链跳转

通过配置自定义函数，可以拦截原有外链跳转并获取 `link` 信息来做处理，示例如下：

```
//拦截外链跳转函数
const onHyperLinkOpen = ({
  linkUrl, //跳转URL
}) => {
  //自身业务处理
};

//配置外链跳转函数
aliyun.config({
  onHyperLinkOpen,
  //...
});
```

自定义功能

初始化JS-SDK时，通过分别配置不同文档对应的功能选项，可以开启或关闭文档中的特定功能和控制文档打开时的状态。

> **注意**
>
> 所有功能选项只能通过初始化配置控制，不支持热切换。

通用选项

通过配置选项 `commonOptions` ，可以配置页面的通用选项。

- 配置项说明

配置项	描述
isShowTopArea	是否显示顶部区域。
isShowHeader	是否显示头部区域。
isBrowserViewFullscreen	是否在浏览器区域全屏。取值范围如下： ○ true：不允许全屏。 ○ false（默认）：允许全屏。
isIframeViewFullscreen	是否在iframe区域内全屏。取值范围如下： ○ true：不允许全屏。 ○ false（默认）：允许全屏。

- 示例

```
aliyun.config({
  //通用选项，所有类型文档适用
  commonOptions: {
    isShowTopArea: false, //隐藏顶部区域（头部和工具栏）
    isShowHeader: false, //隐藏头部区域
    isBrowserViewFullscreen: false, //是否在浏览器区域全屏
    isIframeViewFullscreen: false, //是否在iframe区域内全屏
  },
});
```

文字选项

- 配置项： `wordOptions`
- 子项：

配置项	描述
isShowDocMap	是否开启目录功能，默认开启。
isBestScale	打开文档时，默认以最佳比例显示。
isShowBottomStatusBar	是否展示底部状态栏。
isOpenIntoEdit	（Mobile）要有编辑权限，移动端打开时是否进入编辑。
isShowHoverToolbars	（Mobile）是否显示文字底部工具栏。
isVoiceCommentEnabled	（Mobile）是否允许插入语音评论。

- 示例

```
aliyun.config({
  wordOptions: {
    isShowDocMap: false, //是否开启目录功能，默认开启
    isBestScale: false, //打开文档时，默认以最佳比例显示
    isShowBottomStatusBar: false, //是否展示底部状态栏
    mobile: {
      isOpenIntoEdit: false, // （Mobile）要有编辑权限，移动端打开时是否进入编辑
      isShowHoverToolbars: false, // （Mobile）是否显示文字底部工具栏
      isVoiceCommentEnabled: false, // （Mobile）是否允许插入语音评论
    },
  },
});
```

演示选项

- 配置项：`pptOptions`
- 子项：

配置项	描述
isShowBottomStatusBar	是否展示底部状态栏。
isShowRemarkView	是否显示备注视图。
isShowInsertMedia	是否显示插入音视频入口。

配置项	描述
isOpenIntoEdit	（Mobile）要有编辑权限，移动端打开时是否进入编辑。
showPrevTipWhilePlay	（Mobile）播放时向上翻页，是否展示上一页的提示。
isShowReviewLogo	（Mobile）是否显示审阅左上logo。

- 示例

```
aliyun.config({
  pptOptions: {
    isShowBottomStatusBar: false, //是否展示底部状态栏
    isShowRemarkView: true, //是否显示备注视图
    isShowInsertMedia: true, //是否显示插入音视频入口
    mobile: {
      isOpenIntoEdit: false, // （Mobile）要有编辑权限，移动端打开时是否进入编辑
      showPrevTipWhilePlay: true, // （Mobile）播放时向上翻页，是否展示"上一页"的提示
      isShowReviewLogo: false, // （Mobile）是否显示审阅左上logo
    },
  },
});
```

PDF 选项

- 配置项：`pdfOptions`
- 子项：

配置项	描述
isShowComment	是否显示注解，默认显示。
isInSafeMode	是否处于安全模式（安全模式下不能划选文字，不能复制以及不能通过链接跳转），默认不是安全模式。
isBestScale	默认以最佳显示比例打开。
isShowBottomStatusBar	是否展示底部状态栏。

- 示例

```
aliyun.config({
  pdfOptions: {
    isShowComment: false, //是否显示注解，默认显示
    isInSafeMode: false, //是否处于安全模式（安全模式下不能划选文字，不能复制以及不能通过链接跳转
），默认不是安全模式
    isBestScale: false, //默认以最佳显示比例打开
    isShowBottomStatusBar: false, //是否展示底部状态栏
  },
});
```

组件状态设置

初始化状态设置

通过配置选项 `commandBars` ，可以隐藏、禁用页面的组件或者执行组件命令。

- 示例（隐藏左上角的按钮）

```
aliyun.config({
  commandBars: [
    {
      cmbId: 'HeaderLeft', //组件ID
      attributes: {
        visible: false, //隐藏组件
        enable: false, //禁用组件，组件显示但不响应点击事件
      },
    },
  ],
});
```

- 支持属性

  目前配置中，支持的 `attributes` 属性如下表所示：

属性	数据类型	描述
visible	Boolean	组件显示切换。取值范围如下： ○ true（默认）：显示组件。 ○ false：隐藏组件。
enable	Boolean	组件状态切换。 ○ true（默认）：开启组件。 ○ false：禁用组件。

- 动态更新

  以上配置只在初始化时生效，JS-SDK还提供了动态更新组件状态的接口，示例如下：

```
const instance= aliyun.config({
  //详细配置
});

instance.setCommandBars([
  //可以配置多个组件
  {
    cmbId: '组件ID',
    attributes: {
      visible: false, //隐藏组件
      enable: false, //禁用组件，组件显示但不响应点击事件
    },
  },
]);
```

组件ID列表

下面列出目前可用的组件列表。

- 公共

组件ID	描述
HeaderLeft	（PC）头部左侧，头部一共分左、中、右三块区域。
HeaderRight	（PC）头部右侧，头部一共分左、中、右三块区域。
FloatQuickHelp	右下角帮助。
CooperationPanelOwnerInfo	（Mobile）协作列表中当前文档所有者信息。
Logo	（Mobile）状态栏Logo。
Cooperation	（Mobile）状态栏协作头像。
More	（Mobile）状态栏更多按钮。
HistoryVersion	（PC）顶部状态栏>历史记录菜单>历史版本。
HistoryRecord	（PC）顶部状态栏>历史记录菜单>协作记录。
HistoryVersionDivider	（PC）表格>右键菜单>历史版本/协作记录分割线。

组件ID	描述
SendButton	（Mobile）顶部工具栏>分享按钮。
CooperHistoryMenuItem	（Mobile）顶部工具栏>协作记录菜单。
TabPrintPreview	（PC）顶部工具栏打印按钮。
MenuPrintPreview	（PC）顶部工具栏打印按钮。

注意

如果需要隐藏整个头部以及工具栏，可以在 `config` 中设置 `{ mode: 'simple' }` 切换到极简模式。

- 文字

组件ID	描述
ReviewTrackChanges	编辑、修订状态按钮。
TrackChanges	编辑、修订状态下拉选项。
ContextMenuConvene	文字右键召唤在线协助者。
WriterHoverToolbars	（Mobile）文字>底部工具栏。
ReadSetting	（Mobile）文字>状态栏>阅读设置。

- 表格

组件ID	描述
CheckCellHistory	（PC）单元格最近的改动。

- 演示

组件ID	描述
WPPMobileMarkButton	（Mobile）浮动备注按钮。
PlayComponentToolbar	全屏播放时Toolbar。
WPPPcCommentButton	（PC）底部工具栏>评论按钮。
WPPMobileCommentButton	（Mobile）底部工具栏>评论按钮。

- PDF

组件ID	描述
PDFMobilePageBar	（Mobile）页码。

组件命令执行

通过调用方法可以执行组件命令。

执行方法如下所示：

```
window.onload = async function() {
  const instance= aliyun.config({
    //详细配置
  });

  //注意：执行命令的时候需要等待组件加载完毕
  await instance.ready();

  //通过调用方法执行组件命令
  instance.executeCommandBar('TabPrintPreview');
};
```

更多可用组件ID如下：

- 公共：

组件ID	描述
TabPrintPreview	顶部工具栏打印按钮。

- 文字

组件ID	描述
BookMark	顶部工具栏书签按钮。

主动保存

可以在适当的时机主动触发保存接口，用来辅助用户保存文件。

- 使用方式

```
await instance.ready();

let result = await instance.save();
```

- 返回参数

```
{
  result: 'nochange', //保存状态
  size: 15302, //文件大小，单位Byte
  version: 16, //版本
}
```

- 状态说明

保存状态	描述
ok	版本保存成功，可在历史版本中查看。
nochange	文档无更新，无需保存版本。
SavedEmptyFile	暂不支持保存空文件。触发场景：内核保存完后文件为空。
SpaceFull	空间已满。
QueneFull	保存中请勿频繁操作。触发场景：服务端处理保存队列已满，正在排队。
fail	保存失败。

6.5. 高级用法

6.5.1. 调用流程

通过调用高级功能接口可用于直接操作文档，本文介绍文档在线协作高级功能接口的调用流程。

说明

- JS-SDK提供的接口风格与VBA一致，原则上兼容VBA的接口和参数。在索引调用的时候，请从 `1` 开始。
- 本文中的instance对象是指JS-SDK实例化后的对象。更多信息，请参见快速入门。

步骤一：等待实例化完成

在调用高级功能接口前，需要先等待ready加载完成。

```
window.onload = async function() {
  const instance = aliyun.config({
    url: '在线文档预览地址',
  });

  //等待instance ready之后再调用高级接口
  await instance.ready();
};
```

步骤二：获取文档类型的应用对象

根据 `WordApplication` 、 `ExcelApplication` 、 `PPTApplication` 、 `PDFApplication` 判断当前文档类型，如果不需要特指文档类型，可使用 `instance.Application` 自动识别文档类型。

```
//文字文档
const wordApp = instance.WordApplication();

//表格文档
const excelApp = instance.ExcelApplication();

//演示文档
const pptApp = instance.PPTApplication();

//PDF文档
const pdfApp = instance.PDFApplication();

//自动识别
const app = instance.Application;
```

步骤三：使用高级接口

以设置文字文档缩放比例为例，高级接口的使用方式如下所示。

```
<script src="https://g.alicdn.com/IMM/office-js/1.1.15/aliyun-web-office-sdk.min.js"></scri
pt>
<script>
  window.onload = async function() {
    const instance = aliyun.config({
      url: 'Word文档路径',
    });

    //等待ready加载完成
    await instance.ready();

    //自动识别文档类型
    const app = instance.Application;

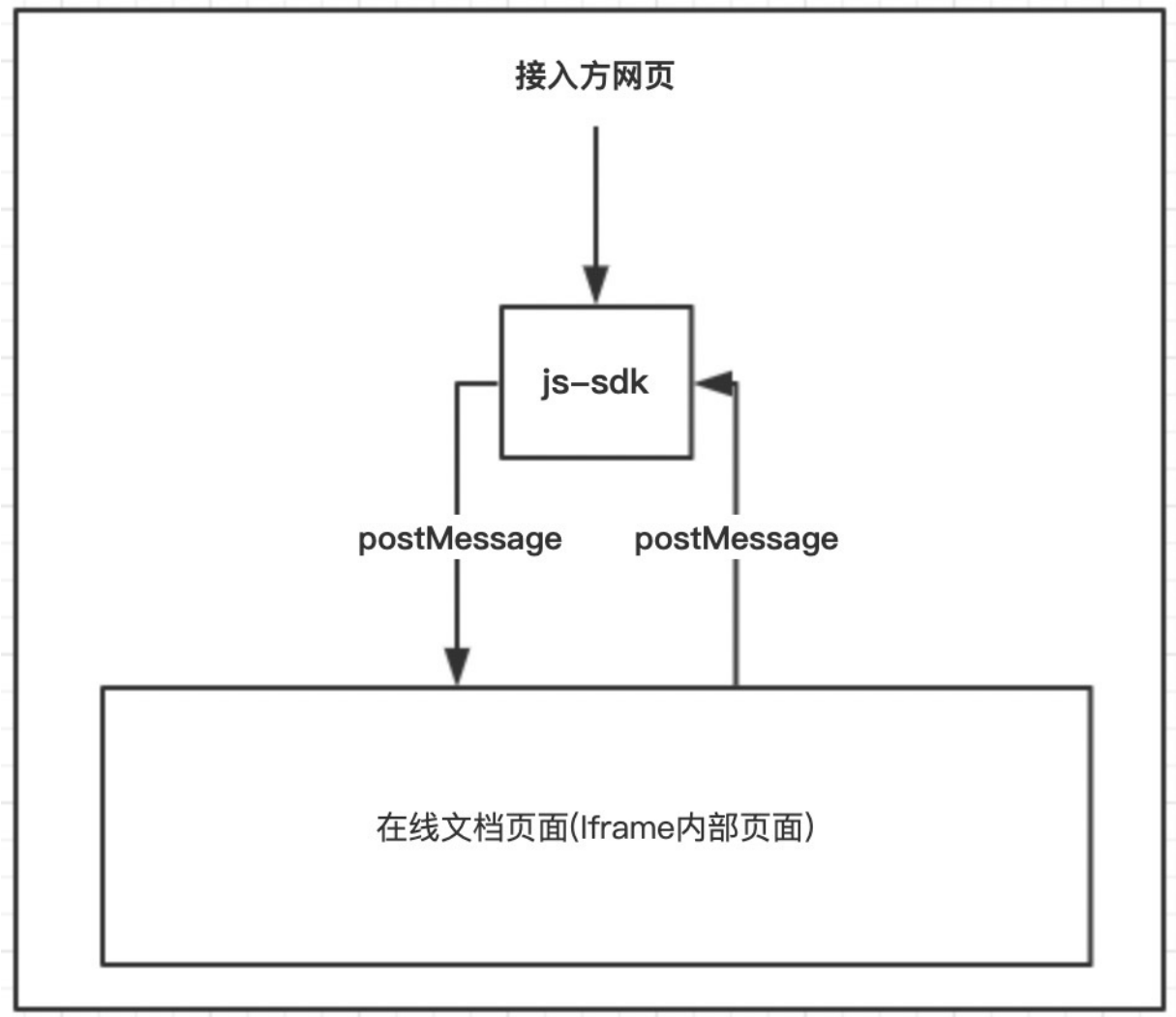
    //设置文档缩放比例为50%
    app.ActiveDocument.ActiveWindow.View.Zoom.Percentage = 50;
  };
</script>
```

6.5.2. 释放对象

由于iframe的跨域安全限制，无法直接调用iframe内部的函数和变量。JS-SDK提供的高级接口对象只是iframe内部对象的映射，通过 `postMessage` 进行通信，所以部分对象需要手动销毁。

> **说明**
>
> 需要手动释放的对象均会在文档中说明，请以文档说明为准。

接入方网页和文档在线编辑的关系如下图所示：



手动释放对象

手动释放单个对象示例：

```
const app = instance.ExcelApplication();

//例如表格获取WorkSheet对象
const sheet = instance.Sheets.Item('sheet1');

//此时得到的sheet对象只是一个映射对象，它拥有iframe内部对象的所有属性和方法，该对象的所有属性和方法都会映射到iframe内部
sheet.Activate(); //切换sheet

//...do something

//不需要改对象时，可以手动销毁
//此时会通知iframe内部销毁对应对象
sheet.Destroy();
```

批量释放对象

当有多个对象需要销毁时，可以按照区间批量释放对象。

代码示例：

```
const stack = app.Stack();

//...do something

stack.End(); //在stack区间的对象都会被销毁
```

6.5.3. 公共

本文介绍如何监听事件、查看可监听的事件名、查看当前SDK版本等。

查看可监听事件

通过 `Events` ，可以查看所有可监听的事件名。

示例如下：

```
async function example() {
  await instance.ready();

  const events = await instance.Events;
  console.log(events);
}
```

监听事件

通过 `on()` ，可以监听事件，例如监听文档保存，然后进一步做其他信息处理。

示例如下：

```
async function example() {
  await instance.ready();

  instance.on('fileSaved', (e) => console.log(e));
}
```

取消事件

通过 `off()` ，可以取消原本的事件。

示例如下：

```
async function example() {
  await instance.ready();

  await instance.off('response', (e) => console.log(e));
}
```

查看当前SDK版本

通过 `version` ，可以查看当前SDK版本。

示例如下：

```
async function example() {
  await instance.ready();

  const version = await instance.version;
  console.log(version);
}
```

查看文档URL

通过 `url` ，可以查看文档的真实路径值（即 `iframe` 嵌套的 `url` ）。

示例如下：

```
async function example() {
  await instance.ready();

  const url = await instance.url;
  console.log(url);
}
```

获取当前iframe对象

通过 `iframe` ，可以获取当前文档实例 `iframe` 对象。

示例如下：

```
async function example() {
  await instance.ready();

  const iframe = await instance.iframe;
  console.log(iframe);
}
```

6.5.4. 文字

6.5.4.1. 页面

本文介绍使用文字文档时如何导出文件为PDF或图片、获取总页数、切换分页和连夜模式、控制目录显示、获取页面宽度等。

导出文件

导出当前文件为对应的PDF或者图片，并获取导出后的URL。

- 语法

```
表达式.ActiveDocument.ExportAsFixedFormat({ OutputFileName, ExportFormat })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
OutputFileName	String	否	导出的文件名称。当前版本不支持。
ExportFormat	Enum	否	导出的文件类型。 `Enum.WdExportFormat` 取值范围如下： ○ 17或wdExportFormatPDF（默认）：将文件导出为PDF格式。 ○ 18或wdExportFormatXPS：将文件导出为XML Paper Specification(XPS)格式，当前版本不支持。 ○ 19或wdExportFormatIMG：将文件导出为图片格式。
OpenAfterExport	Boolean	否	导出内容后打开新文件，当前版本不支持。
OptimizeFor	Enum	否	指定针对屏幕显示或对打印进行优化。当前版本不支持。 `Enum.WdExportOptimizeFor` 取值范围如下： ○ 0或wdExportOptimizeForPrint：针对打印进行导出，质量较高，生成的文件大小较大。 ○ 1或wdExportOptimizeForOnScreen：针对屏幕显示进行导出，质量较差，生成的文件大小较小。

属性	数据类型	是否必填	描述
Range	Enum	否	指定导出区域范围。 `Enum.WdExportRange` 取值范围如下： ○ 0或wdExportAllDocument（默认）：导出整个文件。 ○ 1或wdExportSelection：导出当前页。 ○ 2或wdExportCurrentPage：使用起始和结束位置导出某个范围的内容。 ○ 3或wdExportFromTo：导出当前选择的内容。
From	Number	否	当Range参数设值为wdExportFromTo时，指定起始页码。
To	Number	否	当Range参数设值为wdExportFromTo时，指定结束页码。
Item	Enum	否	指定导出文档是否带标记。 `Enum.WdExportItem` 取值范围如下： ○ 0或wdExportDocumentContent（默认）：导出文档时不带标记。 ○ 7或wdExportDocumentWithMarkup：导出文档时带标记。
IncludeDocProps	Boolean	否	指定在最新导出的文件中是否包括文档属性。
KeepIRM	Boolean	否	如果在源文档具有IRM保护，指定是否将IRM权限复制到XPS文档。当前版本不支持。取值范围如下： ○ true（默认）：是。 ○ false：否。

属性	数据类型	是否必填	描述
CreateBookmarks	Enum	否	指定是否导出书签和要导出的书签的类型。当前版本不支持。 `Enum.WdExportCreateBookmarks` 取值范围如下： ○ 0或wdExportCreateHeadingBookmarks：导出的文档中为每个Word标题创建一个书签，其中仅包含主文档和文本框，不包含页眉、页脚、尾注、脚注或注释的标题。 ○ 1或wdExportCreateNoBookmarks：不在导出的文档中创建书签。 ○ 2或wdExportCreateWordBookmarks：在导出的文档中为每个Word书签创建一个书签，Word书签包括除页眉和页脚中所含书签以外的所有书签。
DocStructureTags	Boolean	否	指定是否包含额外数据来帮助屏幕阅读器，例如有关的流和逻辑组织的内容的信息。取值范围如下： ○ true（默认）：是。 ○ false：否。
BitmapMissingFonts	Boolean	否	指定是否包含文本的位图。当前版本不支持。取值范围如下： ○ true（默认）：不允许在PDF文件中嵌入某一字体。 ○ false：引用该字体，如果编写的一个不可用，会替换合适的字体。
UseISO19005_1	Boolean	否	指定是否将PDF的使用限制为ISO 19005-1标准的PDF子集。当前版本不支持。取值范围如下： ○ false（默认）：否。 ○ true：是。由于受到格式的限制，可能会更大或显示更多的视觉瑕疵。当前版本不支持。

属性	数据类型	是否必填	描述
FixedFormatExtClassPtr	Enum	否	指定一个指针以指向一个允许对代码的备用实现进行调用的加载项。代码的备用实现将对应用程序生成的EMF和EMF+页面描述进行解释，以生成其自身的PDF或XPS。当前版本不支持。
Dpi	Number	否	导出图片的DPI，默认值为92。
Combine2LongPic	Boolean	否	是否输出为长图。取值范围如下： ○ false（默认）：否。 ○ true：是。
ImgFormat	Enum	否	导出图片的格式。`Enum.WdExportImgFormat` 取值范围如下： ○ 0或wdExportFormatPNG（默认）：导出图片格式为PNG。 ○ 1或wdExportFormatJPG：导出图片格式为JPG。 ○ 2或wdExportFormatBMP：导出图片格式为BMP。 ○ 3或wdExportFormatTIF：导出图片格式为TIF。
WaterMark	Boolean	否	是否带水印。取值范围如下： ○ false（默认）：否。 ○ true：是。

- 返回值

  返回打印后 `String` 类型的URL。

- 示例

- 导出PDF

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出PDF，并获取导出后的URL
  const pdfUrl = await app.ActiveDocument.ExportAsFixedFormat(); //默认导出PDF
  console.log(pdfUrl);
}
```

- 导出图片

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出图片，并获取导出后的URL
  const imgUrl = await app.ActiveDocument.ExportAsFixedFormat({ ExportFormat: app.Enum.
WdExportFormat.wdExportFormatIMG }); //或者传递参数19
  console.log(imgUrl);
}
```

获取总页数

通过 `Range.Information` ，可以获取到页面相关的各种信息（目前仅支持获取总页数信息）。

> **注意**
>
> 由于文字文档是流式排版，因此无法在最初就确定最终页数，只有浏览到最底部才知道总页数。

- 语法

  `表达式.ActiveDocument.Range.Information(WdInformation)`

  表达式：文档类型应用对象

- 参数

  通过 `app.Enum.WdInformation` 获取所选内容或区域的信息的类型。

属性	数据类型	是否必填	描述

属性	数据类型	是否必填	描述
WdInformation	Enum	是	获取所选内容或区域的信息。 `Enum.WdInformation` 取值范围如下： ○ 3或wdActiveEndPageNumber：返回包含指定的选定内容或范围，从文档的开头开始计数的活动端的页面数。对页码编号的任何手动调整都将被忽略（与计数不同）。 ○ 4或wdNumberOfPagesInDocument（默认）：返回所选内容或区域相关联的文档的页数。

- 返回值

属性	数据类型	描述
PagesCount	Number	当前已经排版的页码。
End	Boolean	是否已排版结束。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取总页数
  const totalPages = await app.ActiveDocument.Range.Information(app.Enum.WdInformation.wd
NumberOfPagesInDocument);
  console.log(totalPages);
}
```

获取页码

获取当前页页码。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.Information(WdInformation)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
WdInformation	Enum	是	获取所选内容或区域的信息。 `Enum.WdInformation` 取值范围如下： ○ 3或WdInformation.wdActiveEndPageNumber：返回包含指定的选定内容或范围，从文档的开头开始计数的活动端的页面数。对页码编号的任何手动调整都将被忽略（与计数不同）。 ○ 4或WdInformation.wdNumberOfPagesInDocument（默认）：返回与所选内容或区域相关联的文档的页数。

- 返回值

  返回Number代表当前页页码。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取当前页页码
  const currentPage = await app.ActiveDocument.ActiveWindow.Selection.Information(app.Enum.WdInformation.wdActiveEndPageNumber);
  console.log(currentPage );
}
```

切换模式

切换分页或连页模式。

- 语法

  `表达式.ActiveDocument.SwitchTypoMode(Boolean)`

  表达式：文档类型应用对象

  `Boolean` 参数，`true` 表示切换到连页模式，`false` 表示切换到分页模式。

- 返回值

  `Boolean` 返回值，`true` 表示切换成功，`false` 表示切换失败。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换为分页模式
  await app.ActiveDocument.SwitchTypoMode(false);
}
```

获取分页模式总页数

获取分页模式下的总页数。

- 语法

  `表达式.ActiveDocument.ActiveWindow.ActivePane.Pages.Count`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 属性值代表分页模式下总页数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取分页模式下的总页数
  const pages = await app.ActiveDocument.ActiveWindow.ActivePane.Pages.Count;
  console.log(pages);
}
```

显示文件名栏

文件名栏是指连页模式下（非连页模式下不生效），在内容开头会显示文件名相关栏。您可以通过对 `SwitchFileName` 的控制，显示或隐藏文件名栏。

- 语法

  `表达式. ActiveDocument.SwitchFileName(Boolean)`

  表达式：文档类型应用对象

  `Boolean` 参数， `true` 表示显示文件名栏， `false` 表示隐藏文件名栏。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换为连页模式
  await app.ActiveDocument.SwitchTypoMode(true);

  //隐藏文件名栏
  await app.ActiveDocument.SwitchFileName(false);
}
```

控制目录显示

显示或隐藏目录。

- 语法

```
表达式.ActiveDocument.ActiveWindow.DocumentMap = Boolean
```

表达式：文档类型应用对象

`Boolean` 参数， `true` 表示显示目录， `false` 表示隐藏目录。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //显示目录
  app.ActiveDocument.ActiveWindow.DocumentMap = true;
}
```

监听剪切板

获取剪切板回调事件。

- 语法

```
表达式.Sub.ClipboardCopy = Function
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
copyId	String	剪切板ID。
isRestoreFocus	Boolean	是否复原焦点。

属性	数据类型	描述
text	String	复制的内容。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取剪切板回调事件
  app.Sub.ClipboardCopy = async (e) => {
    await console.log(e);
  }
}
```

跳转到指定位置

通过 `GoTo()` 方法，可以跳转到页面的指定位置，例如跳转到指定的页面或者跳转到指定的书签。

> **注意**
>
> 由于文字文档是流式排版，大文档时，跳转时间会比较长，建议加一个中间 `loading` 过渡效果。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.GoTo(What, Which, Count, Name)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
What	Number	可选	指定区域或选定内容要移动到的项的类别。`WdGoToItem` 取值范围如下： ○ -1或WdGoToItem.wdGoToBookmark：跳转类型为书签。 ○ 1或WdGoToItem.wdGoToPage：跳转类型为页面。

属性	数据类型	是否必填	描述
Which	Number	可选	指定区域或选定内容要移动到的项。`WdGoToDirection` 取值为1或WdGoToDirection.wdGoToAbsolute时，表示为绝对定位。
Count	Number	可选	在文档中的项的数目。默认值为1，仅正整数有效。
Name	String	可选	当What取值为wdGoToBookmark时，则指定一个名称。

- 返回值

  返回到跳转后的页面。

- 示例
  - 跳转到指定的页面

```
//跳转到指定页
async function example() {
  await instance.ready();

  const app = instance.Application;

  const page = await app.ActiveDocument.ActiveWindow.Selection.GoTo(instance.Enum.WdGoTo
oItem.wdGoToPage, instance.Enum.WdGoToDirection.wdGoToAbsolute, 3);
}
```

- 跳转到指定的书签

```
//获取所有书签
async function example() {
  await instance.ready();

  const app = instance.Application;

  //添加书签
  await app.ActiveDocument.ActiveWindow.Bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //跳转到指定的书签
  await app.ActiveDocument.ActiveWindow.Selection.GoTo(
    instance.Enum.WdGoToItem.wdGoToBookmark, //类型: Bookmark
    instance.Enum.WdGoToDirection.wdGoToAbsolute, //定位
    undefined, //数量
    'WebOffice', //书签名
  );

  //或者
  await instance.Application.ActiveDocument.ActiveWindow.Selection.GoTo({
    What: Enum.WdGoToItem.wdGoToBookmark,
    Which: Enum.WdGoToDirection.wdGoToAbsolute,
    Name: 'WebOffice', //书签名
  });
}
```

保存文件

保存文件的改动。

> **注意**
>
> JS-SDK V1.1.9及以上版本支持此功能。

- 语法

```
表达式.ActiveDocument.Save()
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
result	String	保存状态。

属性	数据类型	描述
size	Number	文件大小，单位Byte。
version	Number	版本。

- 保存状态说明

保存状态	描述
ok	版本保存成功，可在历史版本中查看。
nochange	文档无更新，无需保存版本。
SavedEmptyFile	暂不支持保存空文件。触发场景：内核保存完后文件为空。
SpaceFull	空间已满。
QueneFull	保存中请勿频繁操作。触发场景：服务端处理保存队列已满，正在排队。
fail	保存失败。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //保存文件的改动
  const save = await app.ActiveDocument.Save();
  console.log(save);
}
```

页面属性

获取页面属性对象

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).PageSetup
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面属性对象
  const pageSetup = await app.ActiveDocument.Sections.Item(1).PageSetup;
}
```

获取页面宽度

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).PageSetup.PageWidth
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面属性对象
  const pageSetup = await app.ActiveDocument.Sections.Item(1).PageSetup;

  //获取页面宽度
  const pageWidth = await pageSetup.PageWidth;
  console.log(pageWidth);
}
```

排版到指定页

由于Word（文档）是流式排版，因此可以通过 `TypesetToPage()` 方法来指定首次加载的页数，滚动页面时进一步加载页面数据。

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).PageSetup.TypesetToPage(PageNum)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
PageNum	Number	是	指定首次加载的页数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面属性对象
  const pageSetup = await app.ActiveDocument.Sections.Item(1).PageSetup;

  //排版到指定页
  await pageSetup.TypesetToPage(2);
}
```

6.5.4.2. 内容控件

本文介绍使用文字文档时如何获取控件数量、设置控件指定属性、获取控件标题、获取控件标签等。

获取内容控件对象

获取文档中的所有内容控件对象。

- 语法

  `表达式.ActiveDocument.ContentControls`

  表达式：文档类型应用对象

- 示例

```
  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;
```

获取内容控件数量

获取文档中的内容控件数量。

- 语法

  `表达式：ActiveDocument.ContentControls.Count`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 代表文档内容控件的数量。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取内容控件数量
  const count = await contentControls.Count;
  console.log(count);
}
```

单个内容控件

获取单个内容控件对象

- 语法

  `表达式.ActiveDocument.ContentControls.Item(Index)`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	String	是	第Index个内容控件。

- 返回值

  返回指定的内容控件对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取第1个内容控件
  const contentControl = await contentControls.Item(1);
}
```

获取和设置控件文本

通过 `Range.Text` 获取和设置内容控件的文本值。

> **注意**
>
> JS-SDK V1.1.15及以上版本支持此功能。

- 语法

  `表达式.ActiveDocument.ContentControls.Item(Index).Range`

  表达式：文档类型应用对象

- 示例
  - 获取内容控件的范围

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取第1个内容控件
  const contentControl = await contentControls.Item(1);

  //获取第1个内容控件的范围
  const range = await contentControl.Range;
  console.log(range);
}
```

  - 获取和设置内容控件的文本值

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取第1个内容控件
  const contentControl = await contentControls.Item(1);

  //获取第1个内容控件的范围
  const range = await contentControl.Range;

  //获取第1个内容控件的文本
  const text = range.Text;
  console.log(text);

  //设置第1个内容控件的文本
  range.Text = 'Aliyun';
}
```

获取控件占位符文本

> **注意**
>
> JS-SDK V1.1.15及以上版本支持此功能。

- 语法

```
表达式.ActiveDocument.ContentControls.Item(Index).PlaceholderText
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取第1个内容控件
  const contentControl = await contentControls.Item(1);

  //获取第1个控件占位符文本
  const placeholderText = await contentControl.PlaceholderText;
  console.log(placeholderText);
}
```

获取控件标题

- 语法

```
表达式.ActiveDocument.ContentControls.Item(Index).Title
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取第1个内容控件
  const contentControl = await contentControls.Item(1);

  //设置第1个内容控件的标题
  contentControl.Title = 'Aliyun';

  //获取第1个内容控件的标题
  const title = await contentControl.Title;
  console.log(title);
}
```

获取和设置控件标签

- 语法

```
表达式.ActiveDocument.ContentControls.Item(Index).Tag
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //获取第1个内容控件
  const contentControl = await contentControls.Item(1);

  //设置第1个内容控件的标签
  contentControl.Tag = 'Aliyun';

  //获取第1个内容控件的标签
  const Tag = await contentControl.Tag;
  console.log(Tag);
}
```

新增内容控件

> **注意**
>
> JS-SDK V1.1.15及以上版本支持此功能。

- 语法

  `表达式.ActiveDocument.ContentControls.Add()`

  表达式：文档类型应用对象

- 返回值

  返回新增的 `ContentControl` 对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //移动光标到指定位置（前面）
  const info = await app.ActiveDocument.Find.Execute('区别');
  const pos = info[0] && info[0].pos || 0;
  await app.ActiveDocument.Range.SetRange(pos, pos);

  //获取内容控件对象
  const contentControls = await app.ActiveDocument.ContentControls;

  //在光标处插入内容控件
  await contentControls.Add();
}
```

6.5.4.3. 书签

本文介绍文字文档如何获取书签对象、添加书签和替换书签内容等。

获取书签对象

- 语法

  `表达式.ActiveDocument.Bookmarks`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //书签对象
  const bookmarks = await app.ActiveDocument.Bookmarks;
}
```

添加书签

- 语法

```
表达式.ActiveDocument.Bookmarks.Add({ Name, Range })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	书签名。命名要求如下： ○ 不能多于一个单词。 ○ 不能含有空格、数字以及半角句号（.）等特殊符号。
Range	Object	是	书签标记的文本区域。可将书签设置到一个折叠的区域（插入点）。

- Range说明

属性	数据类型	是否必填	描述
Start	Number	是	设置书签的起点。
End	Number	是	设置书签的终点。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //书签对象
  const bookmarks = await app.ActiveDocument.Bookmarks;

  //添加书签
  await bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });
}
```

获取书签内容

- 语法

```
表达式.ActiveDocument.Bookmarks.GetBookmarkText(Name)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	书签名。

- 返回值

  返回对应的书签内容。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //书签对象
  const bookmarks = await app.ActiveDocument.Bookmarks;

  //添加书签
  await bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //获取书签内容
  const bookmarkText = await bookmarks.GetBookmarkText('WebOffice');
  console.log(bookmarkText);
}
```

替换书签内容

- 语法

```
表达式.ActiveDocument.Bookmarks.ReplaceBookmark(Data)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Data	Array.<Object>	是	替换内容结构体。

- Data说明

属性	数据类型	是否必填	描述
name	String	是	替换书签的名称。
type	String	是	替换书签的类型。书签类型为TEXT。
value	String	是	替换书签的新内容。

- 返回值

  返回 `Boolean` ，为 `true` 表示替换成功，否则失败。

- 示例

```
//获取所有书签
async function example() {
  await instance.ready();

  const app = instance.Application;

  //书签对象
  const bookmarks = await app.ActiveDocument.Bookmarks;

  //添加书签
  await bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //替换书签内容
  const isReplaceSuccess = await bookmarks.ReplaceBookmark([
    {
      name: 'WebOffice',
      type: 'text',
      value: '替换书签内容',
    },
  ]);
  console.log(isReplaceSuccess); //true
}
```

获取所有书签

- 语法

```
表达式.ActiveDocument.Bookmarks.Json()
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
name	String	书签名。
begin	Number	书签开始位置。
end	Number	书签结束位置。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //添加书签
  await app.ActiveDocument.Bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //获取所有书签
  const bookmarks = await app.ActiveDocument.Bookmarks.Json();
  console.log(bookmarks);
}
```

单个书签

获取单个书签对象

- 语法

```
表达式.ActiveDocument.Bookmarks.Item(Name)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	书签名。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //书签对象
  const bookmarks = await app.ActiveDocument.Bookmarks;

  //添加书签
  await bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //获取单个书签对象
  await app.ActiveDocument.Bookmarks.Item('WebOffice');
}
```

删除书签

- 语法

```
表达式.ActiveDocument.Bookmarks.Item(Name).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //添加书签
  await app.ActiveDocument.Bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //删除书签
  await app.ActiveDocument.Bookmarks.Item('WebOffice').Delete();
}
```

获取书签名称

- 语法

```
表达式.ActiveDocument.Bookmarks.Item(Name).Name
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //添加书签
  await app.ActiveDocument.Bookmarks.Add({
    Name: 'WebOffice',
    Range: {
      Start: 1,
      End: 10,
    },
  });

  //获取书签名称
  await app.ActiveDocument.Bookmarks.Item('WebOffice').Name;
}
```

6.5.4.4. 修订

本文介绍使用文字文档时如何获取修订对象、接受文档所有修订、获取单个修订对象的修订时间等。

获取修订对象

获取指定修订对象。

- 语法

```
表达式.ActiveDocument.Revisions
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;
}
```

切换模式

切换编辑模式与修订模式。

- 语法

```
表达式.ActiveDocument.TrackRevisions = Boolean
```

表达式：文档类型应用对象

`Boolean` 值为 `true` ，表示切换到修订模式， `Boolean` 值为 `false` ，表示切换到编辑模式。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //将当前文档的编辑状态切换成修订模式
  app.ActiveDocument.TrackRevisions = true;
}
```

获取全文修订内容

获取全文的修订内容。

- 语法

```
表达式.ActiveDocument.Revisions.Json()
```

表达式：文档类型应用对象

- 返回值

以 `Array.<Object>` 的形式返回全部修订内容，例如 `[{ user, ... }]` ，具体返回内容如下表所示：

属性	数据类型	描述
user	String	用户名。
leader	String	修订批注。
begin	String	修订开始位置。
end	String	修订结束位置。
content	String	修订后内容。
type	String	修订类型。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取全文修订内容
  const revisionData = await revisions.Json();
  console.log(revisionData);
}
```

获取全文修订数量

获取全文的修订数量。

- 语法

```
表达式.ActiveDocument.Revisions.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取全文修订数量
  const count = await revisions.Count;
  console.log(count);
}
```

接受所有修订

接受对指定文档的所有修订。

- 语法

```
表达式.ActiveDocument.Revisions.AcceptAll()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //接受对指定文档的所有修订
  await revisions.AcceptAll();
}
```

拒绝所有修订

拒绝对指定文档的所有修订。

- 语法

```
表达式.ActiveDocument.Revisions.RejectAll()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //拒绝对指定文档的所有修订
  await revisions.RejectAll();
}
```

单个修订

获取单个修订对象

- 语法

```
表达式.ActiveDocument.Revisions.Item(Index)
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取单个修订对象
  const revision = await revisions.Item(1);
}
```

获取修订时间

- 语法

```
表达式.ActiveDocument.Revisions.Item(Index).Date
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取单个修订对象
  const revision = await revisions.Item(1);

  //获取修订时间
  const date = await revision.Date;
  console.log(date);
}
```

获取修订作者

- 语法

```
表达式.ActiveDocument.Revisions.Item(Index).Author
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取单个修订对象
  const revision = await revisions.Item(1);

  //获取修订作者
  const author = await revision.Author;
  console.log(author);
}
```

获取修订类型

- 语法

```
表达式.ActiveDocument.Revisions.Item(Index).Type
```

表达式：文档类型应用对象

- 返回值

返回 `Enum` 中 `WdRevisionType` 表示对应修订类型。

属性	数据类型	描述
WdRevisionType	Enum	修订类型。`Enum.WdRevisionType` 取值范围如下： ○ 0或wdNoRevision：无修订。 ○ 1或wdRevisionInsert：插入。 ○ 2或wdRevisionDelete：删除。

- 示例

```
async function example() {
  await instance.ready();

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取单个修订对象
  const revision = await revisions.Item(1);

  //获取修订类型
  const type = await revision.Type;
  console.log(type);
}
```

获取修订范围

- 语法

```
表达式.ActiveDocument.Revisions.Item(Index).Range
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  //获取修订对象
  const revisions = await app.ActiveDocument.Revisions;

  //获取单个修订对象
  const revision = await revisions.Item(1);

  //获取修订范围
  const range = await revision.Range;
  console.log(range);
}
```

6.5.4.5. 评论

本文介绍使用文字文档时如何查看文档是否有评论，获取全文评论，控制评论是否显示以及删除所有评论。

查看是否有评论

查看文档是否有评论。

- 语法

  `表达式.ActiveDocument.HasComments()`

  表达式：文档类型应用对象

- 返回值

  类型为 `Boolean` 。当返回值为 `true` 时，表示文档有评论，当返回值为 `false` 时，表示文档无评论。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //查看文档是否有评论
  const hasComments = await app.ActiveDocument.HasComments();
  console.log(hasComments); //返回值为true或者false。
}
```

获取全文评论

获取全文的评论。

- 语法

  `表达式.ActiveDocument.GetComments({ Offset, Limit })`

  表达式：文档类型应用对象

- 参数

  注意

  由于文字文档是流式排版，在使用大文档且 `Limit - Offset` 较大时，获取时间会比较长，建议增加一个中间 `loading` 过渡效果。

属性	数据类型	是否必选	描述
Offset	Number	是	起始位置。

属性	数据类型	是否必选	描述
Limit	Number	是	限制条数。

- 返回值

  `Array.<Object>`

  返回值包含的属性请参见下表。

属性	数据类型	描述
auth	String	评论人。
content	String	内容。
date	Date	评论时间。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取全文评论
  const operatorsInfo = await app.ActiveDocument.GetComments({ Offset: 0, Limit: 20 });
  console.log(operatorsInfo);
}
```

控制评论是否显示

控制文档是否显示评论。

- 语法

  `表达式.ActiveDocument.ActiveWindow.View.ShowComments = Boolean`

  表达式：文档类型应用对象

  当设置 `Boolean` 为 `true` 时，表示显示文档评论，设置 `Boolean` 为 `false` 时，表示隐藏文档评论。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //隐藏文档评论
  app.ActiveDocument.ActiveWindow.View.ShowComments = false;
}
```

删除所有评论

删除文档中的所有评论。

- 语法

```
表达式.ActiveDocument.DeleteAllComments()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //删除文档中的所有评论
  await app.ActiveDocument.DeleteAllComments();
}
```

6.5.4.6. 缩放

本文介绍使用文字文档时如何获取或设置窗口缩放比例以及对文档窗口大小进行调整后，选择是否缩放视图以适应文档窗口的尺寸。

窗口缩放

获取或设置窗口缩放比例。

- 语法

```
表达式.ActiveDocument.ActiveWindow.View.Zoom.Percentage
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口缩放比例
  const result = await app.ActiveDocument.ActiveWindow.View.Zoom.Percentage;
  console.log(result); //窗口默认缩放比例为100%

  //设置窗口缩放比例
  setTimeout(() => {
    app.ActiveDocument.ActiveWindow.View.Zoom.Percentage = 40;
  }, 3000);
}
```

窗口自适应

对文档窗口大小进行调整后，可设置是否缩放视图以适应文档窗口的尺寸。

- 语法

  `表达式.ActiveDocument.ActiveWindow.View.Zoom.PageFit = Number`

  表达式：文档类型应用对象

  `Number` 取值含义如下：

  - 0：对文档窗口大小进行调整后，不为适应文档窗口尺寸而缩放视图。
  - 2：对文档窗口大小进行调整后，使页面以最佳尺寸适应文档窗口。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //对文档窗口大小进行调整后，不缩放视图以适应文档窗口的尺寸
  app.ActiveDocument.ActiveWindow.View.Zoom.PageFit = 0;
}
```

6.5.4.7. 选区

本文介绍使用文字文档时如何获取选区对象、移动选区范围、删除选区内容等。

获取选区对象

选区对象表示窗口或窗格中的当前选定内容。

选定内容表示文档中的选定（或突出显示）区域，或者代表插入点（如果未选择文档中的任何内容）。

- 语法

  `表达式.ActiveDocument.ActiveWindow.Selection`

表达式：文档类型应用对象

每个文档窗格只能有一个 `Selection` 对象，并且在整个应用程序中只能有一个活动的 `Selection` 对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //选区对象
  const selection = await app.ActiveWindow.Selection;
}
```

删除选区内容

删除选区指定内容。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.Delete(WdUnits)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	描述

属性	数据类型	描述
WdUnits	Enum	指定要使用的度量单位。`Enum.WdUnits` 取值范围如下： ○ 1或wdCharacter：字符。 ○ 2或wdWord：字。 ○ 3或wdSentence：句子。 ○ 4或wdParagraph：段落。 ○ 5或wdLine：一个线段。 ○ 6或wdStory：部分。 ○ 7或wdScreen：屏幕尺寸。 ○ 8或wdSection：一节。 ○ 9或wdColumn：列。 ○ 10或wdRow：行。 ○ 11或wdWindow：窗口。 ○ 12或wdCell：单元格。 ○ 13或wdCharacterFormatting：字符格式。 ○ 14或wdParagraphFormatting：段落格式。 ○ 15或wdTable：一个表格。 ○ 16或wdItem：所选项。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //删除选区内容
  app.ActiveDocument.ActiveWindow.Selection.Delete(1);
}
```

向下移动选区

将选区范围向下移动。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.MoveDown()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //将光标向下移动
  await app.ActiveDocument.ActiveWindow.Selection.MoveDown();
}
```

向上移动选区

将选区范围向上移动。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.MoveUp()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //将光标向上移动
  await app.ActiveDocument.ActiveWindow.Selection.MoveUp();
}
```

向左移动选区

将选区范围向左移动。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.MoveLeft()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //将光标向左移动
  await app.ActiveDocument.ActiveWindow.Selection.MoveLeft();
}
```

向右移动选区

将选区范围向右移动。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.MoveRight()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //将光标向右移动
  await app.ActiveDocument.ActiveWindow.Selection.MoveRight();
}
```

获取区域对象

返回一个 `Range` 对象，该对象代表指定对象中包含的文档部分。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.Range
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取选区对象
  const selection = await app.ActiveWindow.Selection;

  //获取区域对象
  const range = await selection.Range;
}
```

修改区域范围

修改指定区域范围。

- 语法

```
表达式.ActiveDocument.Range(Start, End).SetRange({ Start, End })
```
或

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).Range.SetRange({ Start, End })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Start	Number	是	区域开始位置。
End	Number	是	区域结束位置。

- 示例
  - 示例1

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取选中区域
  const range = await app.ActiveDocument.Range(0, 10);

  //设置区域范围
  await range.SetRange(10, 20);
}
```

  - 示例2

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //获取该单元格的区域对象
  const range = await cellOne.Range;

  //单元格字体属性
  const font = await range.Font;

  //修改区域范围
  const newRange = await range.SetRange({
    Start: 1,
    End: 10,
  });
  const newText = await newRange.Text; //获取新区域文本
  console.log(newText);
}
```

6.5.4.8. 水印

本文介绍使用文字文档时如何获取水印对象、插入文字水印、删除水印等。

获取水印对象

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index个代表所选内容、范围或文档中的部分。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;
}
```

插入模板水印

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.AddTemplateWaterMark({ Index, ApplyTo
})
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	使用预设模板水印的索引。
ApplyTo	Number	否	插入的位置。取值范围如下： ○ 0：当前节。 ○ 1：整个文档。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //在整个文档插入第2个预设的模板水印
  waterMarks.AddTemplateWaterMark({
    Index: 2,
    ApplyTo: 1,
  });
}
```

插入文字水印

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.AddTemplateWaterMark({ Text, FontName, FontSize, FontColor, Transparency, Gradient })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Text	String	是	水印文本。
FontName	String	是	水印字体类型。
FontSize	Number	是	水印字体大小。
FontColor	String	是	水印字体颜色，默认值为0xC0C0C0。
Transparency	Number	是	水印透明度。
Gradient	Boolean	是	水印倾斜度。取值范围如下： ○ false（默认）：不倾斜。 ○ true：倾斜。
ApplyTo	Number	是	水印插入位置。默认值为1，即整个文档。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //插入文字水印
  await waterMarks.AddTextWaterMark({
    Text: '水印文本', /Text：水印文本
    FontName: '宋体', //FontName：水印字体类型
    FontSize: 40, //FontSize：水印字体大小
    FontColor: '#171717', //FontColor：水印字体颜色
    Transparency: 0.3, //Transparency：透明度
    Gradient: false, //Gradient：倾斜度
    ApplyTo: 1, //ApplyTo：插入位置
  });
}
```

编辑水印

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.EditTextWaterMark({ Text, FontName,
FontSize, FontColor, Transparency, Gradient })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Text	String	是	水印文本。
FontName	String	是	水印字体类型。
FontSize	Number	是	水印字体大小。
FontColor	String	是	水印字体颜色，默认值为0xC0C0C0。
Transparency	Number	是	水印透明度。
Gradient	Boolean	是	水印倾斜度。

属性	数据类型	是否必填	描述
ApplyTo	Number	是	水印插入位置。默认值为1，即整个文档。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //编辑水印
  await waterMarks.EditTextWaterMark({
    Text: '', //Text水印文本。
    FontName: '宋体', //FontName水印字体类型
    FontSize: 66, //FontSize：水印字体大小
    FontColor: '#f00', //FontColor：水印字体颜色
    Transparency: 1, //Transparency：透明度
    Gradient: true, //Gradient：倾斜度
    ApplyTo: 1, //ApplyTo：插入位置
  });
}
```

删除水印

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.DeleteWaterMark()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //删除水印
  waterMarks.DeleteWaterMark();
}
```

单个水印

获取单个水印对象

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index个水印对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //单个水印对象
  const waterMark = await waterMarks.Item(1);
}
```

设置水印文本

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index).Text
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入文字水印
  await app.ActiveDocument.ActiveWindow.Selection.AddTextWaterMark({
    Text: '水印文本', //Text：水印文本
    FontName: '宋体', //FontName：水印字体类型
    FontSize: 40, //FontSize：水印字体大小
    FontColor: '#171717', //FontColor：水印字体颜色
    Transparency: 0.3, //Transparency：透明度
    Gradient: false, //Gradient：倾斜度
    ApplyTo: 1, //ApplyTo：插入位置
  });

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //获取水印
  const waterMark = await waterMarks.Item(1);

  //设置水印的文本
  waterMark.Text = 'WebOffice';
}
```

设置水印字体类型

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index).FontName
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入文字水印
  await app.ActiveDocument.ActiveWindow.Selection.AddFontNameWaterMark({
    Text: '水印文本', //FontName: 水印文本
    FontName: '宋体', //FontName: 水印字体类型
    FontSize: 40, //FontSize: 水印字体大小
    FontColor: '#171717', //FontColor: 水印字体颜色
    Transparency: 0.3, //Transparency: 透明度
    Gradient: false, //Gradient: 倾斜度
    ApplyTo: 1, //ApplyTo: 插入位置
  });

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //获取水印
  const waterMark = await waterMarks.Item(1);

  //设置水印的字体类型
  waterMark.FontName = '楷体';
}
```

设置水印字体颜色

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index).FontColor
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入文字水印
  await app.ActiveDocument.ActiveWindow.Selection.AddFontNameWaterMark({
    Text: '水印文本', //FontName: 水印文本
    FontName: '宋体', //FontName: 水印字体类型
    FontSize: 40, //FontSize: 水印字体大小
    FontColor: '#171717', //FontColor: 水印字体颜色
    Transparency: 0.3, //Transparency: 透明度
    Gradient: false, //Gradient: 倾斜度
    ApplyTo: 1, //ApplyTo: 插入位置
  });

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //获取水印
  const waterMark = await waterMarks.Item(1);

  //设置水印的字体颜色
  waterMark.FontColor = '#fff000';
}
```

设置水印字体大小

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index).FontSize
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入文字水印
  await app.ActiveDocument.ActiveWindow.Selection.AddFontNameWaterMark({
    Text: '水印文本', //FontName: 水印文本
    FontName: '宋体', //FontName: 水印字体类型
    FontSize: 40, //FontSize: 水印字体大小
    FontColor: '#171717', //FontColor: 水印字体颜色
    Transparency: 0.3, //Transparency: 透明度
    Gradient: false, //Gradient: 倾斜度
    ApplyTo: 1, //ApplyTo: 插入位置
  });

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //获取水印
  const waterMark = await waterMarks.Item(1);

  //设置水印的字体大小
  waterMark.FontSize = 80;
}
```

设置水印倾斜度

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index).Gradient
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入文字水印
  await app.ActiveDocument.ActiveWindow.Selection.AddFontNameWaterMark({
    Text: '水印文本', //FontName: 水印文本
    FontName: '宋体', //FontName: 水印字体类型
    FontSize: 40, //FontSize: 水印字体大小
    FontColor: '#171717', //FontColor: 水印字体颜色
    Transparency: 0.3, //Transparency: 透明度
    Gradient: false, //Gradient: 倾斜度
    ApplyTo: 1, //ApplyTo: 插入位置
  });

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //获取水印
  const waterMark = await waterMarks.Item(1);

  //设置水印的倾斜度
  waterMark.Gradient = true;
}
```

设置水印透明度

- 语法

```
表达式.ActiveDocument.Sections.Item(Index).WaterMarks.Item(Index).Transparency
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入文字水印
  await app.ActiveDocument.ActiveWindow.Selection.AddFontNameWaterMark({
    Text: '水印文本', //FontName: 水印文本
    FontName: '宋体', //FontName: 水印字体类型
    FontSize: 40, //FontSize: 水印字体大小
    FontColor: '#171717', //FontColor: 水印字体颜色
    Transparency: 0.3, //Transparency: 透明度
    Gradient: false, //Gradient: 倾斜度
    ApplyTo: 1, //ApplyTo: 插入位置
  });

  //水印对象
  const waterMarks = await app.ActiveDocument.Sections.Item(1).WaterMarks;

  //获取水印
  const waterMark = await waterMarks.Item(1);

  //设置水印的透明度
  waterMark.Transparency = 0;
}
```

6.5.4.9. 文本

本文介绍使用文字文档时如何修改指定文本、设置选定文字的字体大小、在选定位置插入新段落等。

获取文本

获取指定区域内具体的文本内容信息。

- 语法

  `表达式.ActiveDocument.Range(Start, End).Text` 或

  `表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).Range.Text`

  表达式：文档类型应用对象

- 示例

- 示例1

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取选中区域
  const range = await app.ActiveDocument.Range(0, 100);

  //获取选中文本
  const text = await range.Text;
  console.log(text);
}
```

- 示例2

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //获取该单元格的区域对象
  const range = await cellOne.Range;

  //获取文本
  const text = await range.Text;
  console.log(text);
}
```

修改文本

根据传入数组全文匹配并替换文本。

- 语法

```
表达式.ActiveDocument.ReplaceText(Array.<Object>)
```

表达式：文档类型应用对象

- 参数：
  - 传入一个数组，表示需要替换的文本列表。

属性	数据类型	是否必填	描述
list	Array.<Object>	是	要替换的文本列表。

- list说明

属性	数据类型	是否必填	描述
key	String	是	被替换的文本。
value	String	是	替换的文本。
options	Object	否	替换配置。

- options说明

属性	数据类型	是否必填	描述
isWildcardMatched	Boolean	否	使用通配符。支持星号（*）和问号（?）两种通配符。 ■ 星号（*）：代表任意字符串。 ■ 问号（?）：代表任意单个字符，且此字符必须存在。
isCaseSensitive	Boolean	否	区分大小写。
isWholeWordMatched	Boolean	否	全字匹配。
isWidthIgnored	Boolean	否	忽略全/半角。

- 返回值

  返回 `Boolean` 值，为 `true` 表示替换成功，为 `false` 表示替换失败。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  const isSuccess = await app.ActiveDocument.ReplaceText([
    {
      key: 'WebOffice',
      value: 'js-sdk',
    },
  ]);
  console.log(isSuccess); //true
}
```

插入文本

在选区后面插入文本。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.InsertAfter(Text)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Text	String	是	需要插入的文本。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //在选区后插入文本
  await app.ActiveDocument.ActiveWindow.Selection.InsertAfter('Text');
}
```

插入分隔符

在选区后面插入分隔符。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.InsertBreak()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //在选区位置插入分隔符
  await app.ActiveDocument.ActiveWindow.Selection.InsertBreak();
}
```

删除字符

删除选区位置前的字符。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.TypeBackspace()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //光标向下移动
  await app.ActiveDocument.ActiveWindow.Selection.MoveDown();

  //删除光标前的字符
  await app.ActiveDocument.ActiveWindow.Selection.TypeBackspace();
}
```

获取开始位置

获取区域开始位置。

- 语法

```
表达式.ActiveDocument.Range(Start, End).Start
```

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).Range.Start
```

表达式：文档类型应用对象

- 示例

- 示例1

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取选中区域
  const range = await app.ActiveDocument.Range(0, 100);

  //获取选中文本的开始位置
  const start = await range.Start;
  console.log(start);
}
```

- 示例2

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //获取该单元格的区域对象
  const range = await cellOne.Range;

  //获取选中单元格的开始位置
  const start = await range.Start;
  console.log(start);
}
```

获取结束位置

获取区域结束位置。

- 语法

  `表达式.ActiveDocument.Range(Start, End).End` 或

  `表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).Range.End`

  表达式：文档类型应用对象

- 示例

- 示例1

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取选中区域
  const range = await app.ActiveDocument.Range(0, 100);

  //获取选中文本的结束位置
  const end = await range.End;
  console.log(end);
}
```

- 示例2

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //获取该单元格的区域对象
  const range = await cellOne.Range;

  //获取选择单元格的结束位置
  const end = await range.End;
  console.log(end);
}
```

字体

获取字体对象

通过 `Font` 对象可以控制选中内容的字体，例如设置字体类型、字体大小等内容。

- 语法

  `表达式.ActiveDocument.Range(Start, End).Font`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取字体对象
  const font = await app.ActiveDocument.Range(0, 20).Font;
}
```

设置字体类型

设置选定文字的字体类型。

- 语法

```
表达式.ActiveDocument.Range(Start, End).Font.Name
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取字体对象
  const font = await app.ActiveDocument.Range(0, 20).Font;

  //设置选定文字的字体类型
  font.Name = '宋体';
}
```

设置字体大小

设置选定文字的字体大小。

- 语法

```
表达式.ActiveDocument.Range(Start, End).Font.Size
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取字体对象
  const font = await app.ActiveDocument.Range(0, 20).Font;

  //设置选定文字的字体大小
  font.Size = 30;
}
```

设置字体颜色

设置选定文字的字体颜色。有以下两种方式：

- 方法一：
  - 语法

    ```
    表达式.ActiveDocument.Range(Start, End).Font.Color
    ```

    表达式：文档类型应用对象
  - 示例

    ```
    async function example() {
      await instance.ready();

      const app = instance.Application;

      //获取字体对象
      const font = await app.ActiveDocument.Range(0, 20).Font;

      //设置选定文字的字体颜色
      font.Color = '#228B22';
    }
    ```

- 方法二：
  - 语法

    ```
    表达式.ActiveDocument.Range(Start, End).Font.ColorIndex(WdColorIndex)
    ```

    表达式：文档类型应用对象
  - 参数

属性	数据类型	是否必填	描述

属性	数据类型	是否必填	描述
WdColorIndex	Enum	是	设置字体颜色。`Enum.WdColorIndex` 取值范围如下： ■ -1或wdByAuthor：由文档作者定义的颜色。 ■ 0或wdAutor（默认）：自动配色，通常为黑色。 ■ 1或wdBlackr：黑色。 ■ 2或wdBluer：蓝色。 ■ 3或wdTurquoiser：青绿色。 ■ 4或wdBrightGreen：鲜绿色。 ■ 5或wdPink：粉红色。 ■ 6或wdRed：红色。 ■ 7或wdYellow：黄色。 ■ 8或wdWhite：白色。 ■ 9或wdDarkBlue：深蓝色。 ■ 10或wdTeal：青色。 ■ 11或wdGreen：绿色。 ■ 12或wdViolet：紫色。 ■ 13或wdDarkRed：深红色。 ■ 14或wdDarkYellow：深黄色。 ■ 15或wdGray50：50%灰色底纹。 ■ 16或wdGray25：25%灰色底纹。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取字体对象
  const font = await app.ActiveDocument.Range(0, 20).Font;

  //设置选定文字的字体颜色
  await font.ColorIndex(2);
}
```

设置字体高亮色

- 语法

```
表达式.ActiveDocument.Range(Start, End).Font.HighLight
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取字体对象
  const font = await app.ActiveDocument.Range(0, 20).Font;

  //设置选定文字的高亮色
  font.HighLight = '#228B22';
}
```

段落

获取段落对象

- 语法

```
表达式.ActiveDocument.Range(Start, End).ParagraphFormat
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取段落对象
  const paragraphFormat = await app.ActiveDocument.Range(0, 20).ParagraphFormat;
}
```

插入新段落

在选区位置插入新段落。

- 语法

```
表达式.ActiveDocument.ActiveWindow.Selection.InsertParagraph()
```

表达式：文档类型应用对象

- 示例：

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //在选区位置插入新段落
  await app.ActiveDocument.ActiveWindow.Selection.InsertParagraph();
}
```

缩进首行

- 语法

```
表达式.ActiveDocument.Range(Start, End).ParagraphFormat.CharacterUnitFirstLineIndent
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取段落对象
  const paragraphFormat = await app.ActiveDocument.Range(0, 20).ParagraphFormat;

  //首行缩进2单位
  paragraphFormat.CharacterUnitFirstLineIndent = 2;
}
```

设置行距

- 语法

```
表达式.ActiveDocument.Range(Start, End).ParagraphFormat.LineSpacingRule
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取段落对象
  const paragraphFormat = await app.ActiveDocument.Range(0, 20).ParagraphFormat;

  //设置1.5倍间距
  paragraphFormat.LineSpacingRule = 1.5;
}
```

6.5.4.10. 表格

本文介绍文字文档如何获取表格对象、删除单个表格、设置单元格行高和列宽等。

获取表格对象

获取文档中所有表格对象。

- 语法

```
表达式.ActiveDocument.Tables
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;
}
```

插入表格

- 语法：

```
表达式.ActiveDocument.Tables.Add({ Range, NumRows, NumColumns, DefaultTableBehavior,
AutoFitBehavior })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Range	Object	是	位置信息。 语法： `表达式.ActiveDocument.Selection.Range`
NumRows	Number	是	新建表格的行数。
NumColumns	Number	是	新建表格的列数。

属性	数据类型	是否必填	描述
DefaultTableBehavior	Number	否	单元格是否自动调整大小以适应单元格的内容。`Enum.WdDefaultTableBehavior` 取值范围如下： ○ 0或wdWord8TableBehavior（默认）：禁用自动调整功能。 ○ 1或wdWord9TableBehavior：启用自动调整功能。
AutoFitBehavior	Number	否	设置是否根据Word（文字）自动调整表格大小。`Enum.WdAutoFitBehavior` 取值范围如下： ○ 0或wdAutoFitFixed（默认）：将表格设置为固定大小，与内容无关，不会自动调整表格大小。 ○ 1或wdAutoFitContent：根据表格中包含的内容自动调整表格大小。 ○ 2或wdAutoFitWindow：根据活动窗口的宽度自动调整表格大小。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //插入表格
  await tables.Add(
    app.ActiveDocument.Selection.Range, //位置信息
    3, //新增表格的行数
    3, //新增表格的列数
    1, //启用自动调整功能
    1, //根据表格中包含的内容自动调整表格的大小
  );
}
```

获取表格总个数

获取页面中总表格数量。

- 语法

  `表达式.ActiveDocument.Tables.Count`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 表示页面中总表格数量。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取页面中总表格数量
  const count = await tables.Count;
  console.log(count);
}
```

单个表格

获取单个表格对象

- 语法

  `表达式.ActiveDocument.Tables.Item(Index)`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index个表格。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);
}
```

删除表格

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //删除第1个表格
  await tableOne.Delete();
}
```

获取表格区域

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Range
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的范围
  const range = await tableOne.Range;
}
```

列

获取列对象

获取指定表格的所有列对象。

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有列
  const columns = tableOne.Columns;
}
```

获取总列数

获取指定表格的总列数。

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取表格所有列
  const columns = await tableOne.Columns;

  //获取表格总列数
  const count = await columns.Count;
  console.log(count);
}
```

插入列

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns.Add(BeforeColumn)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
BeforeColumn	String	是	表示将会直接显示在新列右侧的Column对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取表格所有列
  const columns = await tableOne.Columns;

  //插入列
  await columns.Add(1);
}
```

单列

获取单列对象

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index列。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格所有列
  const columns = await tableOne.Columns;

  //获取表格第1列
  const columnOne = await columns.Item(1);
}
```

删除指定列

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格所有列
  const columns = await tableOne.Columns;

  //获取表格第1列
  const columnOne = await columns.Item(1);

  //删除第1列
  columnOne.Delete();
}
```

设置列宽

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns.Item(Index).SetWidth()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
ColumnWidth	Number	是	指定列的宽度，以磅为单位。
RulerStyle	Enum	否	设置Word（文字）调整单元格宽度的方式。`Enum.WdRulerStyle` 取值范围如下： ○ 0或wdAdjustNone（默认）：调整一行或多行的左边缘，通过将所有列左移或右移来保持列宽度。 ○ 1或wdAdjustProportional：调整第一列的左边缘，通过按比例调整指定的一行或多行中所有单元格的宽度，保持表格右边缘的位置。 ○ 2或wdAdjustFirstColumn：只调整第一列的左边缘，保持其他列和表格右边缘的位置。 ○ 3或wdAdjustSameWidth：调整第一列的左边缘，通过将指定的一行或多行中所有单元格的宽度设置为相同值，保持表格右边缘的位置。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格所有列
  const columns = await tableOne.Columns;

  //获取表格第1列
  const columnOne = await columns.Item(1);

  //调整第1列宽度
  columnOne.SetWidth(50);
}
```

获取列区域对象

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Columns.Item(Index).Range
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格所有列
  const columns = await tableOne.Columns;

  //获取表格第1列
  const columnOne = await columns.Item(1);

  //获取第1列的Range对象
  const range = columnOne.Range;
}
```

行

获取行对象

获取指定表格的所有行对象。

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).rows
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const columns = tableOne.Rows;
}
```

获取总行数

获取指定表格的总行数。

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const rows = await tableOne.Rows;

  //获取表格总行数
  const count = await Rows.Count;
  console.log(count);
}
```

插入行

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Add(BeforeRow)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
BeforeRow	String	是	表示将会直接显示在新行右侧的Rows对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const rows = await tableOne.Rows;

  //插入行
  await Rows.Add(1);
}
```

单行

获取单行对象

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index行。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const rows = await tableOne.Rows;

  //获取表格第1行
  const rowOne = await rows.Item(1);
}
```

删除指定行

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const rows = await tableOne.Rows;

  //获取表格第1行
  const rowOne = await rows.Item(1);

  //删除第1行
  rowOne.Delete();
}
```

设置行高

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).SetHeight()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
RowHeight	Number	是	指定行的高度，以磅为单位。
HeightRule	Enum	否	用于确定指定行高度的规规则。`Enum.WdRowHeightRule` 取值范围如下： ○ 0或wdRowHeightAuto（默认）：调整行高以适应该行中的最大高度值。 ○ 1或wdRowHeightAtLeast：行高至少是最小的指定值。 ○ 2或wdRowHeightExactly：行高是固定值。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const rows = await tableOne.Rows;

  //获取表格第1行
  const rowOne = await rows.Item(1);

  //调整第1行高度
  rowOne.SetHeight(50);
}
```

获取行区域对象

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Range
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有表格
  const tables = await app.ActiveDocument.Tables;

  //获取第1个表格
  const tableOne = await tables.Item(1);

  //获取第1个表格的所有行
  const rows = await tableOne.Rows;

  //获取表格第1行
  const rowOne = await rows.Item(1);

  //获取第1行的Range对象
  const range = rowOne.Range;
}
```

单元格

获取单元格对象

表格列、表格行、选定内容或区域中的Cell对象的集合。

- 语法
  - `表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells`
  - `表达式.ActiveDocument.Tables.Item(Index).Columns.Item(Index).Cells`

    表达式：文档类型应用对象
- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格的文本
  const rowText = await tableOne.Rows.Item(1).Cells.Item(1).Range.Text;
  console.log(rowText);

  //获取表格第2列的第2个单元格的文本
  const columnText = await tableOne.columns.Item(2).Cells.Item(2).Range.Text;
  console.log(columnText);
}
```

单个单元格

获取单个单元格对象

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	String	是	第Index个单元格。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const rowText = await tableOne.Rows.Item(1).Cells.Item(1);
}
```

删除单元格

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //删除该单元格所在的行
  await cellOne.Delete();
}
```

设置单元格行宽

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).SetWidth()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
RowWidth	Number	是	指定行的宽度，以磅为单位。
HeightRule	Enum	否	用于确定指定行高度的规则。 `Enum.WdRowHeightRule` 取值范围如下： ○ 0或wdRowHeightAuto（默认）：调整行高以适应该行中的最大高度值。 ○ 1或wdRowHeightAtLeast：行高至少是最小的指定值。 ○ 2或wdRowHeightExactly：行高是固定值。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //调整该单元格所在的宽度
  await cellOne.SetWidth(50);
}
```

设置单元格行高

- 语法

```
表达式.ActiveDocument.Tables.Item(Index).Rows.Item(Index).Cells.Item(Index).SetHeight()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
RowHeight	Number	是	指定行的高度，以磅为单位。
HeightRule	Enum	否	用于确定指定行高度的规则。`Enum.WdRowHeightRule` 取值范围如下： ○ 0或wdRowHeightAuto（默认）：调整行高以适应该行中的最大高度值。 ○ 1或wdRowHeightAtLeast：行高至少是最小的指定值。 ○ 2或wdRowHeightExactly：行高是固定值。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取第1个表格
  const tableOne = await app.ActiveDocument.Tables.Item(1);

  //获取表格第1行的第1个单元格
  const cellOne = await tableOne.Rows.Item(1).Cells.Item(1);

  //调整该单元格所在的高度
  await cellOne.SetHeight(50);
}
```

6.5.4.11. 公文域

本文介绍使用文字文档时如何获取指定公文域名称、批量插入公文域、应用公文域样式等。

获取公文域对象

获取文档中所有的公文域。

- 语法

```
表达式.ActiveDocument.DocumentFields
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取公文域对象
  const documentFields = await app.ActiveDocument.DocumentFields;
}
```

插入公文域

在指定位置插入公文域。

- 语法

```
表达式.ActiveDocument.DocumentFields.Add()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	公文域名称。
Range	Object	是	公文域范围。
Hidden	Boolean	否	是否隐藏。取值范围如下： ○ false（默认）：显示。 ○ true：隐藏。
PrintOut	Boolean	否	是否可打印。取值范围如下： ○ true（默认）：可打印。 ○ false：不可打印。
ReadOnly	Boolean	否	是否只读。取值范围如下： ○ false（默认）：否。 ○ true：是。

- Range说明

属性	数据类型	是否必填	描述
Start	Number	是	开始位置。
End	Number	是	结束位置。

- 返回值

属性	数据类型	描述
params1	String	返回值1说明。
params2	Number	返回值2说明。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });
}
```

批量插入公文域

在指定位置批量插入公文域。

- 语法

```
表达式.ActiveDocument.DocumentFields.AddDocumentFields()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	公文域名称。
Range	Object	是	公文域范围。
Hidden	Boolean	否	是否隐藏。取值范围如下： ○ false（默认）：显示。 ○ true：隐藏。
PrintOut	Boolean	否	是否可打印。取值范围如下： ○ true（默认）：可打印。 ○ false：不可打印。
ReadOnly	Boolean	否	是否只读。取值范围如下： ○ false（默认）：否。 ○ true：是。

属性	数据类型	是否必填	描述
Value	String	否	域对应的值。

- Range说明

属性	数据类型	是否必填	描述
Start	Number	是	开始位置。
End	Number	是	结束位置。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //批量插入公文域
  await app.ActiveDocument.DocumentFields.AddDocumentFields([
    {
      Name: '1', //设置公文域名称
      Range: { Start: 0, End: 10 }, //设置公文域位置
      Value: 'WebOffice1', //设置公文域值
    },
    {
      Name: '2', //设置公文域名称
      Range: { Start: 12, End: 18 }, //设置公文域位置
      Value: 'WebOffice2', //设置公文域值
    },
  ]);
}
```

获取公文域总数

获取公文域总个数。

- 语法

```
表达式.ActiveDocument.DocumentFields.Count
```

表达式：文档类型应用对象

- 返回值

返回 `Number` 表示公文域的个数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取公文域的个数
  const count = app.ActiveDocument.DocumentFields.Count;
  console.log(count);
}
```

获取公文域名称

获取所有公文域名称。

- 语法

```
表达式.ActiveDocument.DocumentFields.GetAllNames()
```

表达式：文档类型应用对象

- 返回值

返回 `Array` 表示所有公文域的名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //批量插入公文域
  await app.ActiveDocument.DocumentFields.AddDocumentFields([
    {
      Name: '1', //设置公文域名称
      Range: { Start: 0, End: 10 }, //设置公文域位置
      Value: 'WebOffice1', //设置公文域值
    },
    {
      Name: '2', //设置公文域名称
      Range: { Start: 12, End: 18 }, //设置公文域位置
      Value: 'WebOffice2', //设置公文域值
    },
  ]);

  //获取所有公文域的名称
  const names = await app.ActiveDocument.DocumentFields.GetAllNames();
  console.log(names); // ['1', '2']
}
```

判断公文域是否存在

判断是否存在某个公文域。

- 语法

```
表达式.ActiveDocument.DocumentFields.Exists(Name)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	公文域名称。

- 返回值

返回 `Boolean` 表示是否有某公文域。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //批量插入公文域
  await app.ActiveDocument.DocumentFields.AddDocumentFields([
    {
      Name: '1', //设置公文域名称
      Range: { Start: 0, End: 10 }, //设置公文域位置
      Value: 'WebOffice1', //设置公文域值
    },
    {
      Name: '2', //设置公文域名称
      Range: { Start: 12, End: 18 }, //设置公文域位置
      Value: 'WebOffice2', //设置公文域值
    },
  ]);

  //判断是否有某公文域
  const hasOne = await app.ActiveDocument.DocumentFields.Exists('1');
  console.log(hasOne); //true
}
```

单个公文域

获取单个公文域

获取公文域集合中的单个公文域。

- 语法

```
表达式.ActiveDocument.DocumentFields.Item({ Index, Name })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 返回值

  返回 `DocumentField` 表示单个公文域。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');
  console.log(documentField);
}
```

删除指定公文域

- 语法

```
表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Delete()
```

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //删除公文域
  await documentField.Delete();
}
```

跳转到公文域开始位置

- 语法

```
表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).GotoBegin()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //跳转到公文域的开始位置
  await documentField.GotoBegin();
}
```

跳转到公文域结束位置

- 语法

```
表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).GotoEnd()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //跳转到公文域的结束位置
  await documentField.GotoEnd();
}
```

查看公文域是否隐藏

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Hidden`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 返回值

  返回 `Boolean` 表示该公文域是否隐藏。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: true, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //查看公文域是否隐藏
  const isHidden = await documentField.Hidden;
  console.log(isHidden);
}
```

查看公文域是否可打印

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).PrintOut`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 返回值

  返回 `Boolean` 表示该公文域是否可打印。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //查看公文域是否可打印
  const isPrintOut = await documentField.PrintOut;
  console.log(isPrintOut);
}
```

查看公文域是否只读

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).ReadOnly`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 返回值

  返回 `Boolean` 表示该公文域是否只读。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: true, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //查看公文域是否只读
  const isReadOnly = await documentField.ReadOnly;
  console.log(isReadOnly);
}
```

查看公文域范围

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Range`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域范围。

- 返回值

  返回 `Range` 对象表示该公文域的范围。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: false, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //查看公文域范围
  const range = await documentField.Range;
  console.log(range);
}
```

查看公文域名称

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Name`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域名称。

- 返回值

  返回 `String` 表示该公文域的名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: true, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');
  console.log(documentField);

  //查看公文域名称
  const name = await documentField.Name;
  console.log(name);
}
```

查看公文域值

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Value`

  表达式：文档类型应用对象

- 返回值

  返回 `String` 表示对应公文域的值。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: true, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //查看公文域的值
  const value = await documentField.Value;
  console.log(value);
}
```

样式

查看公文域样式

- 语法

  `表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Style`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	否	公文域索引。
Name	String	否	公文域样式。

- 返回值

  返回 `Style` 对象表示该公文域的样式。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //插入公文域
  await app.ActiveDocument.DocumentFields.Add({
    Name: '1',
    Range: { Start: 12, End: 20 },
    Hidden: true, //设置公文域显示
    PrintOut: true, //设置公文域可打印
    ReadOnly: true, //设置公文域可读写
  });

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //查看公文域样式
  const style = await documentField.Style;
  console.log(style);
}
```

应用公文域样式

将某个公文域样式应用到另一个公文域。

- 语法

```
表达式.ActiveDocument.DocumentFields.Item({ Index, Name }).Style.ApplyTo(Name)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	公文域名称。

- 返回值

返回 `Array` 表示所有公文域的名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //批量插入公文域
  await app.ActiveDocument.DocumentFields.AddDocumentFields([
    {
      Name: '1', //设置公文域名称
      Range: { Start: 0, End: 10 }, //设置公文域位置
      Value: 'WebOffice1', //设置公文域值
    },
    {
      Name: '2', //设置公文域名称
      Range: { Start: 12, End: 18 }, //设置公文域位置
      Value: 'WebOffice2', //设置公文域值
    },
  ]);

  //获取公文域集合的单个公文域
  const documentField = await app.ActiveDocument.DocumentFields.Item('1');

  //应用公文域样式
  const style = await documentField.Style.ApplyTo('2');
  console.log(style);
}
```

6.5.4.12. 超链接

本文介绍使用文字文档时如何获取文档中的超链接和在指定位置插入超链接。

获取超链接

获取文档中所有超链接的集合。

- 语法

  `表达式.ActiveDocument.Hyperlinks`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //超链接对象
  const hyperlinks = await app.ActiveDocument.Hyperlinks;
}
```

插入超链接

在选区位置插入超链接。

- 语法

```
表达式.ActiveDocument.Hyperlinks.Add({ Address, TextToDisplay })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Address	String	是	插入的链接地址。
TextToDisplay	String	否	超链接展示的文本。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //在选区位置插入超链接
  await app.ActiveDocument.Hyperlinks.Add({
    Address: 'https://www.aliyun.com/', //插入的链接地址
    TextToDisplay: '阿里云官网', //超链接展示的文本
  });
}
```

6.5.4.13. 图形

本文介绍使用文字文档时如何获取图形对象、图形个数、插入图片等。

获取图形对象

获取文档中所有图形对象的集合。

- 语法

```
表达式.ActiveDocument.Shapes
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;
}
```

插入图片

插入非嵌入式图片。

- 语法：

```
表达式.ActiveDocument.Shapes.AddPicture({ FileName, LinkToFile, SaveWithDocument, Left, Top, Width, Height })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
FileName	String	是	图片的路径和文件名。
LinkToFile	Boolean	否	图片是否链接到文件。取值范围如下： ○ true：将图片链接到创建它的文件。 ○ false（默认）：使图片文件成为独立副本。
SaveWithDocument	Boolean	否	图片是否随文档保存。取值范围如下： ○ true：链接的图片随文档一起保存。 ○ false（默认）：链接的图片不随文档一起保存。
Left	Number	否	新图片的左边缘相对于绘图画布的位置，以磅为单位。
Top	Number	否	新图片的上边缘相对于绘图画布的位置，以磅为单位。
Width	Number	否	图片的宽度，以磅为单位。
Height	Number	否	图片的高度，以磅为单位。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });
}
```

获取图形个数

获取图形对象的个数。

- 语法

```
表达式.ActiveDocument.Shapes.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取所有图形数量
  const count = await shapes.Count;
  console.log(count);
}
```

单个图形

获取单个图形对象

获取图形对象集合的单个对象。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	String	是	第Index个图形。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);
}
```

图形宽度

设置单个图形对象的宽度。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index).Width
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //设置第1个图形的宽度
  shape.Width = 200;
}
```

图形高度

设置单个图形对象的高度。

- 语法

  `表达式.ActiveDocument.Shapes.Item(Index).Height`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //设置第1个图形的高度
  shape.Height = 240;
}
```

删除图形

删除单个图形对象。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //删除单个图形
  await shape.Delete();
}
```

转换成嵌入式图形

非嵌入式图形对象转换成嵌入式图形对象。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index).ConvertToInlineShape()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //将第1个图形对象转换成嵌入式对象
  await shape.ConvertToInlineShape();
}
```

设置图形距左值

将指定图形对象水平移动指定的磅数。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index).IncrementLeft(Increment)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Increment	Number	是	指定图形对象水平移动的距离，以磅为单位。取值范围如下： ○ 正数：将图形对象右移。 ○ 负数：将图形对象左移。 ○ 0：图形对象不移动。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //将第1个图形往右移动30磅
  await shape.IncrementLeft(30);
}
```

设置图形距上值

将指定图形对象垂直移动指定的磅数。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index).IncrementTop(Increment)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Increment	Number	是	指定图形对象垂直移动的距离，以磅为单位。取值范围如下： ○ 正数：将图形对象下移。 ○ 负数：将图形对象上移。 ○ 0：图形对象不移动。

示例：

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入非嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片地址
    LinkToFile: true,
    SaveWithDocument: true,
    Left: 10, //图片距离左边位置
    Top: 10, //图片距离顶部位置
    Width: 60, //图片宽度
    Height: 120, //图片高度
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //将第1个图形往下移动30磅
  await shape.IncrementTop(30);
}
```

嵌入式图形

获取嵌入式图形对象

获取所选内容中的所有嵌入式图形对象。

- 语法

  `表达式.ActiveDocument.InlineShapes`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所选内容中的所有嵌入式对象
  const info = await app.ActiveDocument.InlineShapes;
}
```

插入嵌入式图片

- 语法

```
表达式.ActiveDocument.InlineShapes.AddPicture({ FileName, LinkToFile, SaveWithDocument,
Range })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
FileName	String	是	图片的路径和文件名。
LinkToFile	String	否	图片是否链接到文件。取值范围如下： ○ true：将图片链接到创建它的文件。 ○ false（默认）：使图片文件成为独立副本。
SaveWithDocument	String	否	图片是否随文档保存。取值范围如下： ○ true：链接的图片随文档一起保存。 ○ false（默认）：链接的图片不随文档一起保存。
Range	String	否	图片置于文本中的位置。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
    //Range：图形对象
  });
}
```

获取嵌入式图形对象个数

- 语法

```
表达式.ActiveDocument.InlineShapes.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取嵌入式对象的个数
  const count = await shapes.Count;
  console.log(count);
}
```

单个嵌入式图形

获取单个嵌入式图形对象

- 语法

```
表达式.ActiveDocument.InlineShapes.Item(Index)
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取第1个嵌入式图形对象
  const shape = await shapes.Item(1);
}
```

设置嵌入式图形宽度

设置单个嵌入式图形对象的宽度。

- 语法

```
表达式.ActiveDocument.InlineShapes.Item(Index).Width
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取第1个嵌入式图形对象
  const shape = await shapes.Item(1);

  //设置嵌入式图形对象的宽度
  shape.Width = 200;
}
```

设置嵌入式图形高度

设置单个嵌入式图形对象的高度。

- 语法

```
表达式.ActiveDocument.InlineShapes.Item(Index).Height
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取第1个嵌入式图形对象
  const shape = await shapes.Item(1);

  //设置嵌入式图形对象的高度
  shape.Height = 200;
}
```

删除嵌入式图形对象

- 语法

```
表达式.ActiveDocument.InlineShapes.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取第1个嵌入式图形对象
  const shape = await shapes.Item(1);

  //删除嵌入式图形对象
  await shape.Delete();
}
```

转换为非嵌入式图形对象

将嵌入式图形对象转换为非嵌入式图形对象。

- 语法

```
表达式.ActiveDocument.InlineShapes.Item(Index).ConvertToShape()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.InlineShapes;

  //插入嵌入式图片
  await shapes.AddPicture({
    FileName: 'https://example.com/a.jpg', //图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取第1个嵌入式图形对象
  const shape = await shapes.Item(1);

  //嵌入式图形对象转换为非嵌入式图形对象
  await shape.ConvertToShape();
}
```

选中嵌入式图形

选中单个嵌入式图形对象。

- 语法

```
表达式.ActiveDocument.Shapes.Item(Index).Select()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取图形对象
  const shapes = await app.ActiveDocument.Shapes;

  //插入单个图片
  await shapes.AddPicture({
    FileName: 'https://example.aliyundoc.com/583546003_25a7bc59d95249c2a3c257fa369ca289?imageMogr2/thumbnail/180x180!', // 图片路径
    LinkToFile: false,
    SaveWithDocument: false,
  });

  //获取第1个图形对象
  const shape = await shapes.Item(1);

  //选中单个图形对象
  await shape.Select();
}
```

6.5.4.14. 会议

本文介绍使用文字文档时如何获取指定的区域或图形对象、滚动通知事件和选区变化通知事件。

获取当前视图内某个位置的GCP

获取由正文左上角到指定点所在的区域或图形对象。

- 语法

```
表达式.ActiveDocument.ActiveWindow.RangeFromPoint(x, y)
```

表达式：文档类型应用对象。

- 参数

属性	数据类型	是否必填	说明
x	Number	是	该点距离正文左上角的水平距离（以像素为单位），页面的滚动不会影响该点水平距离的改变。
y	Number	是	该点距离正文左上角的垂直距离（以像素为单位），页面的滚动不会影响该点垂直距离的改变。

- 返回值

返回获取到的区域或图形对象range。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //返回由正文左上角到指定点所在的区域或图形对象
  const range = await app.ActiveDocument.ActiveWindow.RangeFromPoint(10, 10);
  console.log(range);

}
```

显示指定GCP

滚动文档窗口，以便在文档窗口显示指定的区域或图形对象。

- 语法

```
表达式.ActiveDocument.ActiveWindow.ScrollIntoView(range)
```

表达式：文档类型应用对象。

- 参数

属性	数据类型	是否必填	说明
range	Range	是	某个位置的区域或图形对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取range
  const range = await app.ActiveDocument.ActiveWindow.RangeFromPoint(10, 10);
  console.log(range);

  //滚动文档窗口,显示指定的区域
  await app.ActiveDocument.ActiveWindow.ScrollIntoView(range)

}
```

滚动通知事件

- 语法

```
表达式.Sub.WindowScrollChange = Function
```

表达式：文档类型应用对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //滚动通知事件
  app.Sub.WindowScrollChange = ({ Data }) => {
    //左上角坐标x,y
    console.log(Data.scrollLeft, Data.scrollTop);
  }
}
```

选区变化通知事件

- 语法

```
表达式.Sub.WindowSelectionChange = Function
```

表达式：文档类型应用对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //选区变化通知事件
  app.Sub.WindowSelectionChange = (e) => {
    const { begin, end } = e;
    //选区开始位置,结束位置
    console.log(`选区开始位置:${begin}, 结束位置:${end}`);
  }

  //设置选区范围
  setTimeout( () => {
    await app.ActiveDocument.Range(100,100).SetRange(10, 10)
  }, 2000);
}
```

6.5.4.15. 目录

本文介绍在文字文档中如何对文档添加目录、获取目录数量、设置目录样式和删除目录等。

目录对象

获取文档中的所有目录对象。

- 语法

```
表达式.ActiveDocument.TablesOfContents
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;
  //获取所有目录对象
  const tablesOfContents = await app.ActiveDocument.TablesOfContents;
}
```

添加目录

- 语法

```
表达式.ActiveDocument.TablesOfContents.Add({ Range })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Range	Object	是	目录插入的区域。

- Range说明

名称	类型	是否必填	描述
Start	Number	是	设置目录的起点。
End	Number	是	设置目录的终点。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);
}
```

获取目录数量

获取文档中目录的数量。

- 语法

```
表达式.ActiveDocument.TablesOfContents.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);

  //获取文档中目录的数量
  await app.ActiveDocument.TablesOfContents.Count;
}
```

单个目录

单个目录对象

获取文档中的单个目录对象。

- 语法

```
表达式.ActiveDocument.TablesOfContents.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	目录索引。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);

  //获取单个目录对象
  await app.ActiveDocument.TablesOfContents.Item(1);
}
```

更新整个目录

- 语法

```
表达式.ActiveDocument.TablesOfContents.Item(Index).Update()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);

  //更新整个目录
  await app.ActiveDocument.TablesOfContents.Item(1).Update();
}
```

更新目录页码

- 语法

```
表达式.ActiveDocument.TablesOfContents.Item(Index).UpdatePageNumbers()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);

  //仅更新目录页码
  await app.ActiveDocument.TablesOfContents.Item(1).UpdatePageNumbers();
}
```

设置目录样式

- 语法

```
表达式.ActiveDocument.TablesOfContents.Item(Index).SetStyles(LowerHeadingLevel,
IncludePageNumbers, TabLeader)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
LowerHeadingLevel	Number	可选	显示级别，可选范围为1~9。
IncludePageNumbers	Boolean	可选	是否显示页码。
TabLeader	Number	可选	制表符前导符类型。`WdTabLeader` 取值： ○ 0或wdTabLeaderSpaces：空格。 ○ 1或wdTabLeaderDots：点。 ○ 2或wdTabLeaderDashes：划线。 ○ 3或wdTabLeaderLines：直线。 ○ 4或wdTabLeaderHeavy：点线。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);

  //设置目录样式
  const tabLeader = app.Enum.WdTabLeader.wdTabLeaderDashes
  await app.ActiveDocument.TablesOfContents.Item(1).SetStyles(9, true, tabLeader)
}
```

删除目录

- 语法

```
表达式.ActiveDocument.TablesOfContents.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.ActiveDocument.Range(0, 0);

  //插入目录
  await app.ActiveDocument.TablesOfContents.Add(range);

  //删除目录
  await app.ActiveDocument.TablesOfContents.Item(1).Delete();
}
```

6.5.5. 表格

6.5.5.1. 页面

在使用表格文档在线协作时，您可以导出表格、获取工作表名称、切换工作表、获取登录操作者信息、监听剪切板等。

导出表格

导出整个表格

将整个表格导出为对应的PDF或者IMG图片，并获取导出后的URL。

- 语法

```
表达式.ActiveWorkbook.ExportAsFixedFormat({ Type }
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Type	Enum	可选	导出的文件类型。`Enum.XlFixedFormatType` 取值： ○ 0或xlTypePDF：将文件导出为PDF格式。 ○ 1或xlTypeXPS：将文件导出为XML Paper Specification(XPS)格式，当前版本不支持。 ○ 2或xlTypeIMG：将文件导出为IMG图片。

- 返回值

属性	数据类型	描述
url	string	导出后的URL。

- 示例
  - 导出PDF

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出整个表格
  const workbookPdfUrl = await app.ActiveWorkbook.ExportAsFixedFormat();
  console.log(workbookPdfUrl);
}
```

- 导出图片

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出整个表格
  const workbookPdfUrl = await app.ActiveWorkbook.ExportAsFixedFormat({
    Type: app.Enum.XlFixedFormatType.xlTypeIMG,
  });
  console.log(workbookPdfUrl);
}
```

导出当前工作表

将当前工作表导出为对应的PDF或者IMG图片，并获取导出后的URL。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.ExportAsFixedFormat({ Type }
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Type	Enum	可选	导出的文件类型。`Enum.XlFixedFormatType` 取值： ○ 0或xlTypePDF：将文件导出为PDF格式。 ○ 1或xlTypeXPS：将文件导出为XML Paper Specification(XPS)格式，当前版本不支持。 ○ 2或xlTypeIMG：将文件导出为IMG图片。

- 返回值

属性	数据类型	描述
url	string	导出后的URL。

- 示例

- 导出PDF

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出当前工作表
  const workbookPdfUrl = await app.ActiveWorkbook.ActiveSheet.ExportAsFixedFormat();
  console.log(workbookPdfUrl);
}
```

- 导出图片

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出当前工作表
  const workbookPdfUrl = await app.ActiveWorkbook.ActiveSheet.ExportAsFixedFormat({
    Type: app.Enum.XlFixedFormatType.xlTypeIMG,
  });
  console.log(workbookPdfUrl);
}
```

获取工作表名称

获取所有工作表名称

- 语法

```
表达式.For(Start, End, Step, Handle)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Start	Number	是	表格开头Sheet位置。
End	Number	否	表格结束Sheet位置。
Step	Number	否	每次遍历步骤。
Handle	Number	否	回调方法。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有工作表名称
  const Names = [];
  await app.For(
    1,
    app.Sheets.Count,
    1,
    async (Index) => {
      Names.push(await app.Sheets.Item(Index).Name);
    },
  );
  console.log(Names);
}
```

获取当前工作表名称

- 语法

  `表达式.ActiveWorkbook.ActiveSheet.Name`

  表达式：文档类型应用对象

- 返回值

  返回 `String` 表示对应的Sheet名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取当前工作表名称
  const name = await app.ActiveWorkbook.ActiveSheet.Name;
  console.log(name);
}
```

切换工作表

切换到指定工作表

- 语法
  - `表达式.ActiveWorkbook.Sheets.Item(Index).Activate()`
  - `表达式.Sheets(Index).Activate()`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index个Sheet。

- 示例
  - 示例一

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到指定工作表
  const sheetIndex = 2; //Sheets序号，从1开始
  app.ActiveWorkbook.Sheets.Item(sheetIndex).Activate();
}
```

  - 示例二

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到指定工作表
  const sheetIndex = 2; //Sheets序号，从1开始
  app.Sheets(sheetIndex).Activate();
}
```

切换监听工作表

- 语法

```
表达式.Sub.Worksheet_Activate = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换监听工作表
  app.Sub.Worksheet_Activate = async () => {
    console.log('ActiveSheet: ', await app.ActiveSheet.Name);
  };
}
```

获取登录操作者信息

获取当前登录操作者信息。

- 语法

```
表达式.ActiveWorkbook.GetOperatorsInfo()
```

表达式：文档类型应用对象

- 返回值
  - Object object

属性	数据类型	描述
type	String	请求类型。
response	Object	用户信息列表。

  - response结构

属性	数据类型	描述
id	String	用户ID。
avatar_url	Object	用户头像。
logined	Object	用户登录状态。
name	Object	用户姓名。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取当前登录操作者信息
  const operatorsInfo = await app.ActiveWorkbook.GetOperatorsInfo();
  console.log(operatorsInfo);
}
```

监听剪切板

监听复制操作。

- 语法

```
表达式.Sub.Clipboard_Copy = Function
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
copyId	String	剪切板ID。
isRestoreFocus	Boolean	undefined。
text	String	复制的内容。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听复制操作
  app.Sub.Clipboard_Copy = async (e) => {
    await console.log(e);
  }
}
```

缩放窗口

缩放窗口比例。

- 语法

```
表达式.ActiveWindow.Zoom
```

表达式：文档类型应用对象

- 返回值

返回 `Number` 型的缩放比例。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口缩放比例
  const zoom = await app.ActiveWindow.Zoom;
  console.log(zoom);

  //操作窗口缩放比例
  app.ActiveWindow.Zoom = 10;
}
```

显示或隐藏网格线

可显示或隐藏网格线。

- 语法

```
表达式.ActiveWindow.DisplayGridlines = Boolean
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //隐藏网格线
  app.ActiveWindow.DisplayGridlines = false;
}
```

显示或隐藏标题

可显示或隐藏行标题和列标题。

- 语法

```
表达式.ActiveWindow.DisplayHeadings = Boolean
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //隐藏行标题和列标题
  app.ActiveWindow.DisplayHeadings = false;
}
```

保存文件

> **注意**
>
> 支持JS-SDK V1.1.9及以上版本。

保存文件的改动。

- 语法

```
表达式.ActiveWorkbook.Save()
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
result	String	保存状态。
size	Number	文件大小，单位Byte。
version	Number	版本。

- 保存状态说明

保存状态	描述
ok	版本保存成功，可在历史版本中查看。
nochange	文档无更新，无需保存版本。
SavedEmptyFile	暂不支持保存空文件。 触发场景：内核保存完后文件为空。
SpaceFull	空间已满。
QueneFull	保存中请勿频繁操作。 触发场景：服务端处理保存队列已满，正在排队。
fail	保存失败。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //保存文件的改动
  const saveResult = await app.ActiveWorkbook.Save();
  console.log(saveResult);
}
```

切换卡片视图面板

> **注意**
>
> 支持JS-SDK V1.1.15及以上版本。

- 语法

```
表达式.ActiveWorkbook.SwitchCardView()
```

表达式：文档类型应用对象

- 参数

传入一个布尔值，表明打开或者关闭卡片视图面板。

属性	数据类型	是否必填	描述
Status	Boolean	是	目前仅支持打开面板。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //打开卡片视图面板
  await app.ActiveWorkbook.SwitchCardView(true);
}
```

6.5.5.2. 活动工作簿

本文介绍如何在表格文件中获取工作簿对象、工作簿名称和新增工作表等功能。

活动工作簿对象

获取一个表示活动工作簿的工作簿对象。

- 语法

```
表达式.ActiveWorkbook
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //活动工作簿
  const activeWorkbook = await app.ActiveWorkbook;
}
```

名称管理器

获取工作簿名称的集合。

- 语法

```
表达式.ActiveWorkbook.Names
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //活动工作簿
  const activeWorkbook = await app.ActiveWorkbook;

  //工作簿名称集合
  const names = await activeWorkbook.Names;
}
```

工作表

获取工作表对象

- 语法

```
表达式.ActiveWorkbook.Sheets
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const sheets = await app.ActiveWorkbook.Sheets;
}
```

新增工作表

- 语法

```
表达式.ActiveWorkbook.Sheets.Add({ Before, After, Count, Type, Name })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Before	String/Number	是	指定工作表的对象，新建的工作表将置于此工作表之前。
After	String/Number	否	指定工作表的对象，新建的工作表将置于此工作表之后。
Count	Number	否	要添加的工作表数。默认值为选定工作表的数量。
Type	Enum	否	指定工作表类型。`Enum.XlSheetType` 取值： ○ -4167或xlWorksheet：工作表。 ○ -4116或xlDialogSheet：对话框工作表。 ○ -4109或xlChart：图标。 ○ 3或xlExcel4MacroSheet：Excel版本4宏工作表。 ○ 4或xlExcel4IntlMacroSheet：Excel版本4国际宏工作表。
Name	Name	否	指定工作表名称。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const sheets = await app.ActiveWorkbook.Sheets;

  //添加工作表
  await sheets.Add(null, null, 1, app.Enum.XlSheetType.xlWorksheet, '新工作表');
}
```

获取工作表数量

- 语法

```
表达式.ActiveWorkbook.Sheets.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const sheets = await app.ActiveWorkbook.Sheets;

  //工作表数量
  const count = await sheets.Count;
  console.log(count);
}
```

获取工作表类型

- 语法

```
表达式.ActiveWorkbook.Sheets.Item(Index).Type
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const sheetIndex = 2; //Sheets序号，从1开始
  const sheet = await app.ActiveWorkbook.Sheets.Item(sheetIndex)

  //工作表类型
  const Type = await sheet.Type;
  console.log(Type);
}
```

获取单个工作表对象

获取活动工作簿中的活动工作表。

> **注意**
>
> 目前该对象（WorkSheet）可以和活动工作表（ActiveSheet）一样使用。

- 语法

```
表达式.ActiveWorkbook.Sheets.Item(Index)
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.Sheets.Item(1);
}
```

开启或关闭多人同步筛选

- 语法

```
表达式.ActiveWorkbook.SetFilterShared({ Checked })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Checked	Boolean	是	可开启或关闭筛选。 ○ true：开启。 ○ false：关闭。

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //活动工作簿
  const activeWorkbook = await app.ActiveWorkbook;

  //开启多人同步筛选
  const result = await activeWorkbook.SetFilterShared(true);
}
```

6.5.5.3. 活动工作表

本文介绍使用表格文档时如何获取可编辑区域对象、保护工作表、设置工作表是否可见等。

获取活动工作表对象

获取活动工作簿中的活动工作表。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;
}
```

可编辑区域

获取可编辑区域对象

获取活动工作簿中活动工作表的可编辑区域。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.AllowEditRanges
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取可编辑区域对象
  const allowEditRanges = await activeSheet.AllowEditRanges;
}
```

添加可编辑区域

在受保护的工作表（Sheet）中添加一个可编辑区域。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.AllowEditRanges.Add({ Title, Range, Password })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Range	Object	是	区域对象。
Title	String	否	标题。
Password	String	否	密码。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取可编辑区域
  const allowEditRanges = await activeSheet.AllowEditRanges;

  //设置范围：左上角至A1，右下角至E20
  const range = await activeSheet.Range('$A1:$E20');

  //添加可编辑区域
  await allowEditRanges.Add(undefined, range, 'WebOffice');
}
```

获取图表对象

获取工作表上的所有图表对象。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.ChartObjects
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表上的所有图表
  const chartObjects = await activeSheet.ChartObjects;
}
```

获取行

获取工作表上的所有行，返回的是一个Range对象。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Rows
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表上的所有行
  const chartObjects = await activeSheet.Rows;
}
```

获取列

获取工作表上的所有列，返回的是一个Range对象。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Columns
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表上的所有列
  const chartObjects = await activeSheet.Columns;
}
```

获取单元格

获取工作表上的所有单元格，返回的是一个Range对象。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Cells
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表上的所有单元格
  const cells = await activeSheet.Cells;
}
```

删除工作表

删除活动工作簿中的活动工作表。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //删除工作表
  await activeSheet.Delete();
}
```

获取索引号

获取工作表对应的索引号。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Index
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表对应索引号
  const index = await activeSheet.Index;
  console.log(index);
}
```

获取工作表名称

返回一个Names集合，该集合代表所有工作表特定的名称。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //返回一个Names集合，该集合代表所有工作表特定的名称
  const names = await activeSheet.Names;
}
```

保护工作表

保护工作表使其不被修改。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Protect({ Password, DrawingObjects, Scenarios,
AllowFormattingCells, AllowFormattingColumns, AllowFormattingRows, AllowInsertingColumns,
AllowInsertingRows, AllowInsertingHyperlinks, AllowDeletingColumns, AllowDeletingRows,
AllowSorting, AllowFiltering, AllowUsingPivotTables })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Password	String	否	指定工作表或工作簿的密码。该密码字符串区分大小写。 ○ 省略此参数，可以不使用密码解除对工作表或工作簿的保护。 ○ 使用此参数，必须使用指定密码才能解除对工作表或工作簿的保护。 注意 请务必牢记设置的密码，如果忘记密码，则无法解除对工作表或工作簿的保护。
DrawingObjects	Boolean	否	是否保护工作表形状。取值范围如下： ○ false（默认）：否。 ○ true：是。
Scenarios	Boolean	否	是否保护工作表方案，此参数仅对工作表有效。取值范围如下： ○ true（默认）：是。 ○ false：否。
AllowFormattingCells	Boolean	否	是否允许用户格式化受保护工作表上的任何单元格。取值范围如下： ○ false（默认）：否。 ○ true：是。
AllowFormattingColumns	Boolean	否	是否允许用户格式化受保护工作表上的任何列。取值范围如下： ○ false（默认）：否。 ○ true：是。
AllowFormattingRows	Boolean	否	是否允许用户格式化受保护的工作表上的任何行。取值范围如下： ○ false（默认）：否。 ○ true：是。

属性	数据类型	是否必填	描述
AllowInsertingColumns	Boolean	否	是否允许用户在受保护工作表上插入列。取值范围如下： ○ false（默认）：否。 ○ true：是。
AllowInsertingRows	Boolean	否	是否允许用户在受保护工作表上插入行。取值范围如下： ○ false（默认）：否。 ○ true：是。
AllowInsertingHyperlinks	Boolean	否	是否允许用户在受保护的工作表上插入超链接。取值范围如下： ○ false（默认）：否。 ○ true：是。
AllowDeletingColumns	Boolean	否	是否允许用户在受保护的工作表上删除列。取值范围如下： ○ false（默认）：否。 ○ true：是。要删除的列中的每个单元格都处于解锁状态。
AllowDeletingRows	Boolean	否	是否允许用户在受保护的工作表上删除行。取值范围如下： ○ false（默认）：否。 ○ true：是。要删除的行中的每个单元格都处于解锁状态。
AllowSorting	Boolean	否	是否允许用户对受保护工作表进行排序。取值范围如下： ○ false（默认）：否。 ○ true：是。排序范围内的每个单元格都必须处于解锁或不受保护的状态。

属性	数据类型	是否必填	描述
AllowFiltering	Boolean	否	是否允许用户在受保护的工作表上设置筛选。取值范围如下： ○ false（默认）：否。 ○ true：是。用户可以在现有自动筛选上设置筛选或更改筛选条件，但无法启用或禁用自动筛选。
AllowUsingPivotTables	Boolean	否	是否允许用户在受保护的工作表上使用数据透视表。取值范围如下： ○ false（默认）：否。 ○ true：是。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //输入密码
  activeSheet.Protect('123456');
}
```

解除保护

解除工作表或工作簿中的保护。如果工作表或工作簿不受保护，则此方法无效。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Unprotect({ Password })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Password	String	否	设置的密码。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //解除保护
  activeSheet.Unprotect('123456');
}
```

获取图形对象

获取当前工作表上的所有Shape对象的集合。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;
}
```

设置工作表是否可见

设置活动工作簿中的活动工作表是否可见。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Visible
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //设置工作表不可见
  activeSheet.Visible = false;
}
```

获取所用区域对象

获取工作表上的所用区域，返回Range对象。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.UsedRange
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //工作表上的所用区域
  const UsedRange = await activeSheet.UsedRange;
}
```

获取区域对象

获取区域对象，返回一个Range对象，表示一个单元格或单元格区域。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Range
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取区域对象，返回一个Range对象，它表示一个单元格或单元格区域
  const range = await activeSheet.Range;
}
```

6.5.5.4. 名称管理器

本文介绍使用表格文档时如何获取名称合集、新建名称、删除单个名称等。

获取名称对象

获取名称对象，返回一个Name集合，该集合代表所有工作表特定的名称。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取名称对象，返回一个Name集合，该集合代表所有工作表特定的名称
  const names = await activeSheet.Names;
}
```

新建名称

为单元格区域定义新名称。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names.Add({ Name, RefersTo })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Name	String	是	名称。
RefersTo	Number	否	名称引用的内容。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表名称的集合
  const names = await activeSheet.Names;

  //新增一个名称
  names.Add('新表', 'A1');
}
```

查看名称数量

查看工作表名称数量。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取工作表名称的集合
  const names = await activeSheet.Names;

  //查看名称数量
  const count = await names.Count;
  console.log(count);
}
```

单个名称

获取单个名称对象

获取第Index个名称。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names.Item(Index)
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取名称对象
  const names = await activeSheet.Names;

  //获取单个名称对象
  const name = await names.Item(1);
}
```

查看具体名称

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names.Item(Index).Name
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取名称对象
  const names = await activeSheet.Names;

  //获取单个名称对象
  const name = await names.Item(1);

  //查看具体名称
  const nameName = await name.Name;
  console.log(nameName);
}
```

获取名称的值

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names.Item(Index).Value
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取名称对象
  const names = await activeSheet.Names;

  //获取单个名称对象
  const name = await names.Item(1);

  //获取名称的值
  const value = await name.Value;
  console.log(value);
}
```

删除名称

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Names.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取名称对象
  const names = await activeSheet.Names;

  //获取单个名称对象
  const name = await names.Item(1);

  //删除名称
  await name.Delete();
}
```

6.5.5.5. 图形

本文介绍使用表格文档时如何获取图形数量、添加图表、设置图表标题等。

获取图形对象

获取当前工作表上所有Shape对象的集合。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;
}
```

查看图形数量

查看当前工作表上图形的数量。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Count
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //查看当前图形的数量
  const count = await shapes.Count;
  console.log(count);
}
```

添加图表

在当前工作表添加指定图表。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.AddChart2({ Style, XlChartType, Left, Top, Width, Height })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Style	String	否	指定新建图表的样式。
XlChartType	Enum	否	指定新建图表的类型。
Left	Number	否	指定新建图表的左边距，单位px。
Top	Number	否	指定新建图表的上边距，单位px。
Width	Number	否	指定新建图表的宽度，单位px。
Height	Number	否	指定新建图表的高度，单位px。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //添加300*300的矩形
  await shapes.AddChart2(340, 51, 0, 0, 300, 300);
}
```

单个图形

获取单个图形对象

代表绘图层中的对象。例如自选图形、任意多边形、OLE对象或图片。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index)
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //添加300×300的矩形
  await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图形对象
  const shape = await shapes.Item(1);
}
```

设置单个图形标题

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Title
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //添加300×300的矩形
  await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图形对象
  const shape = await shapes.Item(1);

  //设置图形标题
  shape.Title = 'WebOffice';
}
```

设置单个图形名称

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Name
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //添加300×300的矩形
  await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图形对象
  const shape = await shapes.Item(1);

  //设置图形名称
  const name = await shape.Name;
  console.log(name);
}
```

设置单个图形ID

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).ID
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //添加300×300的矩形
  await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图形对象
  const shape = await shapes.Item(1);

  //设置图形ID
  const id = await shape.ID;
  console.log(id);
}
```

选择单个图形

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Select()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //获取单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取图形数据源
  const source = await activeSheet.Range('A1:B4');

  //设置图形数据源
  await shape.Chart.SetSourceData(source, 1);

  //选择单个图形
  setTimeout( async () => {
    await shape.Select();
  }, 3000);
}
```

删除单个图形

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //获取单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取图形数据源
  const source = await activeSheet.Range('A1:B4');

  //设置图形数据源
  await shape.Chart.SetSourceData(source, 1);

  //删除单个图形
  setTimeout( async () => {
    await shape.Delete();
  }, 3000);
}
```

单个图表

获取单个图表对象

- 语法

  `表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Chart`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //获取单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图表对象
  const chart = await shape.Chart;
}
```

设置图表数据源

为指定图表设置源数据区域。

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Chart.SetSourceData({ Source, PlotBy
})
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Source	Range	是	包含源数据的区域，可用Range对象。
PlotBy	Enum	否	指定图表类型。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图表对象
  const chart = await shape.Chart;

  //获取图表数据源
  const source = await activeSheet.Range('A1:D4');

  //设置图表数据源
  await chart.SetSourceData(source, 1);
}
```

设置标题是否可见

如果坐标轴或图表有可见标题，则该属性值为 `true` 。

- 语法

  `表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Chart.HasTitle`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合取图形对象
  const shapes = await activeSheet.Shapes;

  //获取单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图表对象
  const chart = await shape.Chart;

  //获取图表数据源
  const source = await activeSheet.Range('A1:D4');

  //设置图表数据源
  await chart.SetSourceData(source, 1);

  //设置标题不可见
  chart.HasTitle = false;
}
```

设置图例是否可见

如果图表有图例，则该属性值为 `true` 。

- 语法

  `表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Chart.HasLegend`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //获取单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图表对象
  const chart = await shape.Chart;

  //获取图表数据源
  const source = await activeSheet.Range('A1:D4');

  //设置图表数据源
  await chart.SetSourceData(source, 1);

  //设置图例不可见
  chart.HasLegend = false;
}
```

设置标题

- 语法

```
表达式.ActiveWorkbook.ActiveSheet.Shapes.Item(Index).Chart.ChartTitle.Text
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取活动工作簿中的活动工作表
  const activeSheet = await app.ActiveWorkbook.ActiveSheet;

  //获取当前工作表上的所有Shape对象的集合
  const shapes = await activeSheet.Shapes;

  //获取单个图形对象
  const shape = await shapes.AddChart2(340, 51, 0, 0, 300, 300);

  //获取单个图表对象
  const chart = await shape.Chart;

  //获取图表数据源
  const source = await activeSheet.Range('A1:D4');

  //设置图表数据源
  await chart.SetSourceData(source, 1);

  //设置新标题
  chart.ChartTitle.Text = '这是新标题';
}
```

6.5.5.6. 区域

本文介绍如何在表格文件中获取指定区域、设置区域属性和为指定区域添加数据有效性规则等功能。

区域对象

表示一个单元格、一行、一列、一个包含单个或若干连续单元格区域的选定单元格范围。

- 语法

  `表达式.Range`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');
}
```

区域子项

返回一个 `Range` 对象，表示区域中指定的位置。

- 语法

```
表达式.Range.Item({ RowIndex, ColumnIndex })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
RowIndex	Number	是	行号或索引。 ○ 如果提供了第二个参数，则为要返回的单元格的相对行号。 ○ 如果未提供第二个参数，则为要返回的子范围的索引。
ColumnIndex	Number	否	要返回的单元格的相对列号。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');

  //区域子项：B2
  const item1 = await range.Item(2, 2);
  await item1.Select();
}
```

属性

获取区域宽度

- 语法：

```
表达式.Range.ColumnWidth
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域宽度
  range.ColumnWidth = 30;
}
```

获取区域高度

- 语法

  `表达式.Range.RowHeight`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域高度
  range.RowHeight = 100;
}
```

获取区域数量

- 语法

  `表达式.Range.Count`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域的数量
  const count = await range.Count;
  console.log(count);
}
```

获取左边距

左边距为从列A的左边缘到该区域的左边缘的距离。

- 语法

  `表达式.Range.Left`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('B2');

  //左边距
  const left = await range.Left;
  console.log(left);
}
```

获取上边距

上边距为从行1的上边缘到区域的上边缘之间的距离。

- 语法：

  `表达式.Range.Top`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('B2');

  //上边距
  const top = await range.Top;
  console.log(top);
}
```

隐藏行或列

- 语法

  `表达式.Range.Hidden`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //获取整列
  const entireColumn = await range.Rows.EntireColumn;
  console.log(entireColumn);

  //隐藏该列
  entireColumn.Hidden = true;
}
```

设置区域水平对齐方式

- 语法：

  `表达式.Range.HorizontalAlignment`

  表达式：文档类型应用对象

- 可设置的值

字段	值	释义
xlHAlignRight	-4152	靠右对齐。

字段	值	释义
xlHAlignLeft	-4131	靠左对齐。
xlHAlignJustify	-4130	两端对齐。
xlHAlignDistributed	-4117	分散对齐。
xlHAlignCenter	-4108	居中对齐。
xlHAlignGeneral	1	按数据类型对齐。
xlHAlignFill	5	填充对齐。
xlHAlignCenterAcrossSelection	7	跨列居中对齐。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //设置对齐方式：居中
  range.HorizontalAlignment = await app.Enum.XlHAlign.xlHAlignCenter;

  setTimeout( async () => {
    //获取对齐方式
    const horizontalAlignment = await range.HorizontalAlignment;
    console.log(horizontalAlignment);
  }, 3000);
}
```

设置区域垂直对齐方式

- 语法：

```
表达式.Range.VerticalAlignment
```

表达式：文档类型应用对象

- 可设置的值

字段	值	释义
xlVAlignTop	-4160	向上。
xlVAlignJustify	-4130	调整使全行排满。
xlVAlignDistributed	-4117	一起。
xlVAlignCenter	-4108	居中。
xlVAlignBottom	-4107	向下。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //设置对齐方式：底部对齐
  range.VerticalAlignment = await app.Enum.XlVAlign.xlVAlignBottom;

  setTimeout( async () => {
    //获取对齐方式
    const horizontalAlignment = await range.VerticalAlignment;
    console.log(horizontalAlignment);
  }, 3000);
}
```

内部属性

获取内部属性对象

- 语法

```
表达式.Range.Interior
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //内部属性对象
  const interior = await range.Interior;
}
```

获取区域内部颜色

- 语法

```
表达式.Range.Interior.Color
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //内部属性对象
  const interior = await range.Interior;

  //设置内部颜色
  interior.Color = '#e1ff02';

  setTimeout( async () => {
    //获取内部颜色
    const color = await interior.Color;
    console.log(color);
  }, 3000);
}
```

方法

激活区域

- 语法

```
表达式.Range.Activate()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //激活区域A1
  range.Activate();
}
```

选择区域

- 语法

```
表达式.Range.Select()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //选择区域A1
  range.Select();

  //5000 ms后选择区域A1:D2
  setTimeout(async () => {
    const newRange = await app.Range('B1:D2');
    newRange.Select();
  }, 5000);
}
```

删除区域内容

- 语法

```
表达式.Range.ClearContents()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //删除区域内容
  await range.ClearContents();
}
```

判断区域是否重叠

- 语法

```
表达式.Range.Contain({ Range })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Range	Range	是	Range对象，另一块区域。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');

  //第二块区域对象
  const newRange = await app.Range('A1:B4');

  //判断是否重叠
  const contain = await range.Contain(newRange);
  console.log(contain);
}
```

区域引用

获取使用宏语言的区域引用的 `String` 值。

- 语法

```
表达式.Range.Address({ RowAbsolute, ColumnAbsolute, ReferenceStyle, External, RelativeTo
})
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
RowAbsolute	Boolean	否	默认值为true。若为true，以绝对引用的形式返回引用的行部分。
ColumnAbsolute	Boolean	否	默认值为true。若为true，以绝对引用的形式返回引用的列部分。
ReferenceStyle	Enum	否	引用样式。`Enum.XlReferenceStyle` 取值： ○ -4150或xlR1C1：返回R1C1样式的引用。 ○ 1或xlA1（默认）：返回A1样式的引用。
External	Boolean	否	默认值为false。 ○ true：返回外部引用。 ○ false：返回本地引用。
RelativeTo	Range	否	定义起点的Range对象。如果 `RowAbsolute` 和 `ColumnAbsolute` 为false，且 `ReferenceStyle` 为xlR1C1，则必须为相对引用包含一个起点。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //获取使用宏语言的区域引用的String值
  const address1 = await range.Address();
  console.log('address1: ', address1);

  const address2 = await range.Address(false, false);
  console.log('address2: ', address2);

  const address3 = await range.Address(true, true, -4150);
  console.log('address3: ', address3);
}
```

自动填充

对指定区域中的单元格执行自动填充。

- 语法

```
表达式.Range.AutoFill({ Destination, Type })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Destination	Range	否	目标区域。目标区域必须包含源区域。

属性	数据类型	是否必填	描述
Type	Enum	否	填充类型。`Enum.XlAutoFillType` 取值： ○ 0或xlFillDefault（默认）：确定用于填充目标区域的值和格式。 ○ 1或xlFillCopy：将源区域的值和格式复制到目标区域，如有必要可重复执行。 ○ 2或xlFillSeries：将源区域的值扩展到目标区域，形式为系列（如“1,2”扩展为“3,4,5”）；将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 3或xlFillFormats：只将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 4或xlFillValues：只将源区域的值复制到目标区域，如有必要可重复执行。 ○ 5或xlFillDays：将星期中每天的名称从源区域扩展到目标区域；将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 6或xlFillWeekdays：将工作周每天的名称从源区域扩展到目标区域；将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 7或xlFillMonths：将月名称从源区域扩展到目标区域；将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 8或xlFillYears：将年从源区域扩展到目标区域；将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 9或xlLinearTrend：将数值从源区域扩展到目标区域，假定数字之间是加法关系（如“1,2,”扩展为“3,4,5”，假定每个数字都是前一个数字加上某个值的结果）；将源区域的格式复制到目标区域，如有必要可重复执行。 ○ 10或xlGrowthTrend：将数值从源区域扩展到目标区域，假定源区域的数字之间是乘法关系（如“1,2,”扩展为“4,8,16”，假定每个数字都是前一个数字乘以某个值的结果）。将源区域的格式复制到目标区域，如有必要可重复执行。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:A2');

  //要填充的单元格
  const fillRange = await app.Range('A1:A20');

  //对指定区域中的单元格执行自动填充
  await range.AutoFill(fillRange);
}
```

迁移区域

对指定区域进行迁移操作。

- 语法

```
表达式.Range.Offset({ RowOffset, ColumnOffset })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
RowOffset	Number	否	区域偏移的行数：可以是正值、负值或零。默认值为0。 ○ 正值表示向下偏移。 ○ 负值表示向上偏移。
ColumnOffset	Number	否	区域偏移的列数：可以是正值、负值或零。默认值为0。 ○ 正值表示向右偏移。 ○ 负值表示向左偏移。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');
  await range.Select();

  //对指定区域进行迁移操作
  const newRange = await range.Offset(2, 2);
  await newRange.Select();
}
```

导出图片

将区域导出成图片。

- 语法

  `表达式.Range.ToImageDataURL()`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');

  //将区域导出成图片
  const img = await range.ToImageDataURL();
  console.log(img);
}
```

边框

获取边框对象

- 语法

  `表达式.Range.Borders`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //边框对象
  const borders = await range.Borders;
}
```

单个边框

获取单个边框对象

单个边框对象代表单元格区域或样式的边框之一。

- 语法

  `表达式.Range.Borders.Item(Index)`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Enum	是	指定要检索的边框。`Enum.XlBordersIndex` 取值： ○ 5或xlDiagonalDown：从区域中每个单元格的左上角到右下角的边框。 ○ 6或xlDiagonalUp：从区域中每个单元格的左下角到右上角的边框。 ○ 7或xlEdgeLeft：区域左边缘的边框。 ○ 8或xlEdgeTop：区域顶部的边框。 ○ 9或xlEdgeBottom：区域底部的边框。 ○ 10或xlEdgeRight：区域右边缘的边框。 ○ 11或xlInsideVertical：区域中所有单元格的垂直边框（区域以外的边框除外）。 ○ 12或xlInsideHorizontal：区域中所有单元格的水平边框（区域以外的边框除外）。 ○ 13或xlOutside：区域中的上下左右。 ○ 14或xlInside：中间区域。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //边框对象
  const borders = await range.Borders;

  //单个边框对象
  const border = await borders.Item(5);
}
```

获取边框颜色

- 语法

```
表达式.Range.Borders.Item(Index).Color
```

表达式：文档类型应用对象

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //边框对象
  const borders = await range.Borders;

  /单个边框对象
  const border = await borders.Item(app.Enum.XlBordersIndex.xlDiagonalDown);

  //获取边框颜色
  const color = await border.Color;
  console.log(color);
}
```

设置边框的线条样式

- 语法

```
表达式.Range.Borders.Item(Index).LineStyle
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A3');

  //边框对象
  const borders = await range.Borders;

  //单个边框对象
  const border = await borders.Item(app.Enum.XlBordersIndex.xlOutside);

  //设置边框的线条样式
  border.LineStyle = app.Enum.XlLineStyle.xlDash;
}
```

设置边框的粗细

- 语法

```
表达式.Range.Borders.Item(Index).Weight
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A3');

  //边框对象
  const borders = await range.Borders;

  //单个边框对象
  const border = await borders.Item(app.Enum.XlBordersIndex.xlOutside);

  //设置边框的粗细
  border.Weight = app.Enum.XlBorderWeight.xlThick;
}
```

行

获取指定行

获取区域中的某行，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.Range.Rows
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //行对象
  const rows = await range.Rows;

  //指定行
  const row = await rows.Item(2);

  //选中第2行
  await row.Select();
}
```

获取首行行号

获取区域中第一行的行号。

- 语法

```
表达式.Range.Row
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域中第一行的行号
  const row = await range.Row;
  console.log(row);
}
```

获取尾行行号

获取区域中最后一行的行号。

- 语法

```
表达式.Range.RowEnd
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域中最后一行的行号
  const rowEnd = await range.RowEnd;
  console.log(rowEnd);
}
```

获取整行

获取包含指定区域的整行。

- 语法

```
表达式.Range.EntireRow
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //获取该区域的行
  const entireRow = range.EntireRow;

  //选择这一行
  await entireRow.Select();
}
```

列

获取指定列

获取区域中的某列，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.Range.Columns
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //列对象
  const columns = await range.Columns;

  //指定列
  const column = await columns.Item(2);

  //选中第2列
  await column.Select();
}
```

获取首列列号

获取区域中第一列的列号。

- 语法

  `表达式.Range.Column`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域中第一列的列号
  const column = await range.Column;
  console.log(column);
}
```

获取尾列列号

获取区域中最后一列的列号。

- 语法

  `表达式.Range.ColumnEnd`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //区域中最后一列的列号
  const columnEnd = await range.ColumnEnd;
  console.log(columnEnd);
}
```

获取整列

获取包含指定区域的整列。

- 语法

```
表达式.Range.EntireColumn
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //获取该区域的行
  const entireColumn = range.EntireColumn;

  //选择这列
  await entireColumn.Select();
}
```

单元格

获取指定单元格

获取区域中的单元格，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.Range.Cells
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //单元格对象
  const cells = await range.Cells;

  //指定单元格
  const cell = await cells.Item(1);
  window.cells = cells;

  //选中第1个单元格
  await cell.Select();
}
```

获取单元格文本

- 语法

  `表达式.Range.Text`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //文本
  const text = await range.Text;
  console.log(text);
}
```

合并单元格

- 语法

  `表达式.Range.Merge({ Across })`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Across	Boolean	是	默认值为false。如果设置为true，则将指定区域中每一行的单元格合并为一个单独的合并单元格。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');

  //合并单元格
  await range.Merge();
}
```

获取单元格合并区域

- 语法

```
表达式.Range.MergeArea
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');

  //合并单元格
  await range.Merge();

  //页面加载合并后，5000 ms内点击空白区域，5000 ms后自动选择该区域
  setTimeout( async () => {
    //单元格的合并区域
    const mergeArea = await range.MergeArea;
    mergeArea.Select();
  }, 5000);
}
```

查询合并单元格

查询区域内是否存在合并的单元格。

- 语法

  `表达式.Range.MergeCells`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:D2');

  //合并单元格
  await range.Merge();

  //查询区域内是否存在合并的单元格
  const mergeCells = await range.MergeCells;
  console.log(mergeCells);
}
```

单元格赋值

在单元格中写入值。

- 语法

  `表达式.Range.Value`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //写入值到单元格中
  range.Value = 'WebOffice';
}
```

字体

获取字体对象

- 语法

  `表达式.Range.Font`

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //字体对象
  const font = range.Font;
}
```

设置和获取字体大小

- 语法

```
表达式.Range.Font.Size
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //字体对象
  const font = range.Font.Size;

  //设置字体大小
  font.Size = 30;

  //获取字体大小
  const size = await font.Size;
  console.log(size);
}
```

设置和获取字体加粗

- 语法

```
表达式.Range.Font.Bold
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //字体对象
  const font = range.Font;

  //获取字体加粗
  const isBold = await font.Bold;
  console.log('字体是否加粗：', isBold);

  //设置字体加粗
  font.Bold = true;
}
```

评论

添加评论

给指定区域添加评论。

- 语法

```
表达式.Range.AddComment({ Text })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Text	String	否	评论文本。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //给A1区域添加评论
  range.AddComment('WebOffice');
```

清除评论

清除指定区域的评论。

- 语法

```
表达式.Range.ClearComments()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1:B2');

  //清除区域的评论
  await range.ClearComments();
}
```

公式

设置公式

以A1样式表示法表示对象的隐式交叉公式。

- 语法

```
表达式.Range.Formula
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A2');

  //设置公式：A2=A1+B1
  range.Formula = '=A1+B1';
}
```

设置和获取数组公式

设置和获取指定区域的数组公式。

- 语法

```
表达式.Range.FormulaArray
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A2:B2');

  //设置公式：A2到B2的值为A1+B1的和
  range.FormulaArray = '=Sum(A1:B1)';

  //获取该公式
  const formulaArray = await range.FormulaArray;
  console.log(formulaArray);
}
```

条件格式

获取条件格式对象

获取一个区域内所有条件格式的集合。

- 语法

```
表达式.Range.FormatConditions
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //条件格式对象
  const formatConditions = await range.FormatConditions;
}
```

添加条件格式

代表一个区域内所有条件格式的集合。

- 语法

```
表达式.Range.FormatConditions.Add({ Type, Operator, Formula1, Formula2 })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Type	Enum	是	指定条件格式是基于单元格值还是基于表达式。`Enum.XlFormatConditionType` 取值： ○ 1或xlCellValue：单元格值。 ○ 2或xlExpression：表达式。 ○ 3或xlColorScale：色阶。 ○ 5或xlTop10：前10个值。 ○ 8或xlUniqueValues：唯一值。 ○ 9或xlTextString：文本字符串。 ○ 10或xlBlanksCondition：空值条件。 ○ 11或xlTimePeriod：时间段。 ○ 12或xlAboveAverageCondition：高于平均值条件。 ○ 13或xlNoBlanksCondition：无空值条件。 ○ 16或xlErrorsCondition：错误条件。 ○ 17或xlNoErrorsCondition：无错误条件。
Operator	Number	否	条件格式运算符，可以是以下常量之一，`Enum.XlFormatConditionOperator` 取值： ○ 1或xlBetween：行间。只在提供了两个公式的情况下才能使用。 ○ 2或xlNotBetween：不介于。只在提供了两个公式的情况下才能使用。 ○ 3或xlEqual：等于。 ○ 4或xlNotEqual：不等于。 ○ 5或xlGreater：大于。 ○ 6或xlLess：小于。 ○ 7或xlGreaterEqual：大于或等于。 ○ 8或xlLessEqual：小于或等于。
Formula1	Number	否	与条件格式关联的值或表达式，可为常量值、字符串值、单元格引用或公式。

属性	数据类型	是否必填	描述
Formula2	Number	否	当条件格式运算符为xlBetween或xlNotBetween时，为条件格式第二部分相关联的值或表达式（取值为其他值时，将忽略此参数）。可为常量值、字符串值、单元格引用或公式。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A4:D5');

  //选择区域
  await range.Select();

  //设置公式
  range.Formula = 'WebOffice';

  //条件格式对象
  const formatConditions = await range.FormatConditions;

  //添加条件格式
  formatConditions.Add(
    app.Enum.XlFormatConditionType.xlExpression,
    undefined,
    '=D1=1',
  );
}
```

数据有效性

获取数据有效性对象

- 语法

```
表达式.Range.Validation
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //数据有效性对象
  const validation = await range.Validation;
}
```

添加数据验证

新增数据有效性规则。

- 语法

```
表达式.Range.Validation.Add({ Type, Operator, Formula1, Formula2 })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Type	Enum	是	验证警报样式。`Enum.XlDVType` 取值： ○ 1或xlValidateWholeNumber：全部数值。 ○ 2或xlValidateDecimal：数值。 ○ 3或xlValidateList：值必须存在于指定列表中。 ○ 4或xlValidateDate：日期值。 ○ 5或xlValidateTime：时间值。 ○ 6或xlValidateTextLength：文本长度。 ○ 7或xlValidateCustom：使用任意公式验证数据有效性。

属性	数据类型	是否必填	描述
Operator	Enum	否	数据验证运算符。`Enum.XlFormatConditionOperator` 取值： ○ 1或xlBetween：行间。只在提供了两个公式的情况下才能使用。 ○ 2或xlNotBetween：不介于。只在提供了两个公式的情况下才能使用。 ○ 3或xlEqual：等于。 ○ 4或xlNotEqual：不等于。 ○ 5或xlGreater：大于。 ○ 6或xlLess：小于。 ○ 7或xlGreaterEqual：大于或等于。 ○ 8或xlLessEqual：小于或等于。
Formula1	String	否	数据验证公式中的第一部分，取值不得超过255个字符。
Formula2	String	否	当数据验证运算符为xlBetween或xlNotBetween时，为数据验证等式的第二部分内容（取值为其他值时将忽略此参数）。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //数据有效性对象
  const validation = await range.Validation;

  //添加数据验证
  await validation.Add({
    Type: app.Enum.XlDVType.xlValidateWholeNumber,
    Operator: app.Enum.XlFormatConditionOperator.xlBetween,
    Formula1: '1',
    Formula2: '5',
  });
}
```

修改数据验证

修改数据有效性规则。

- 语法

```
表达式.Range.Validation.Modify({ Type, Operator, Formula1, Formula2 })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Type	Enum	是	验证警报样式。`Enum.XlDVType` 取值： ○ 1或xlValidateWholeNumber：全部数值。 ○ 2或xlValidateDecimal：数值。 ○ 3或xlValidateList：值必须存在于指定列表中。 ○ 4或xlValidateDate：日期值。 ○ 5或xlValidateTime：时间值。 ○ 6或xlValidateTextLength：文本长度。 ○ 7或xlValidateCustom：使用任意公式验证数据有效性。
Operator	Enum	否	数据验证运算符。`Enum.XlFormatConditionOperator` 取值： ○ 1或xlBetween：行间。只在提供了两个公式的情况下才能使用。 ○ 2或xlNotBetween：不介于。只在提供了两个公式的情况下才能使用。 ○ 3或xlEqual：等于。 ○ 4或xlNotEqual：不等于。 ○ 5或xlGreater：大于。 ○ 6或xlLess：小于。 ○ 7或xlGreaterEqual：大于或等于。 ○ 8或xlLessEqual：小于或等于。
Formula1	String	否	数据验证公式中的第一部分，取值不得超过255个字符。
Formula2	String	否	当数据验证运算符为xlBetween或xlNotBetween时，为数据验证等式的第二部分内容（取值为其他值时将忽略此参数）。

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //数据有效性对象
  const validation = await range.Validation;

  //添加数据验证
  await validation.Add({
    Type: app.Enum.XlDVType.xlValidateWholeNumber,
    Operator: app.Enum.XlFormatConditionOperator.xlBetween,
    Formula1: '1',
    Formula2: '5',
  });

  //10000 ms后修改规则
  setTimeout( async () => {
    await validation.Modify({
      Type: app.Enum.XlDVType.xlValidateWholeNumber,
      Operator: app.Enum.XlFormatConditionOperator.xlNotBetween,
      Formula1: '1',
      Formula2: '5',
    });
  }, 10000);
}
```

删除数据验证

删除数据有效性规则。

- 语法

  `表达式.Range.Validation.Delete()`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //区域对象
  const range = await app.Range('A1');

  //数据有效性对象
  const validation = await range.Validation;

  //添加数据验证
  await validation.Add({
    Type: app.Enum.XlDVType.xlValidateWholeNumber,
    Operator: app.Enum.XlFormatConditionOperator.xlBetween,
    Formula1: '1',
    Formula2: '5',
  });

  //10000 ms后删除规则
  setTimeout( async () => {
    await validation.Delete();
  }, 10000);
}
```

6.5.5.7. 会议

本文介绍如何在表格文件中设置可视区域的指定行号、指定列号和退出编辑等功能。

> **注意**
>
> 支持JS-SDK V1.1.15及以上版本。

可视区域

设置和获取列号

设置和获取可视区域最左侧的列号。

- 语法

  `表达式.ActiveWindow.ScrollColumn`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 型的可视区的列数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取左侧的列号
  const ScrollColumn = await app.ActiveWindow.ScrollColumn;
  console.log(ScrollColumn);

  //设置左侧的列号
  app.ActiveWindow.ScrollColumn = 10;
}
```

设置和获取行号

设置和获取可视区域最上面的行号。

- 语法

  `表达式.ActiveWindow.ScrollRow`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 型的可视区的行数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取最上面的行号
  const ScrollRow = await app.ActiveWindow.ScrollRow;
  console.log(ScrollRow);

  //设置最上面的行号
  app.ActiveWindow.ScrollRow = 10;
}
```

选区

一个单元格或单元格区域

获取一个单元格或单元格区域，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

  `表达式.ActiveSheet.Range`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const Sheet = await app.ActiveSheet;

  //一个单元格或单元格区域
  const Range = await Sheet.Range;

  //指定单元格
  const range = await Range.Item("D2");
  // 或者 range = await Range("D2")

  //选中D2单元格
  await range.Select();
}
```

获取工作表中的单元格

获取工作表中的单元格，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.ActiveSheet.Cells
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const Sheet = await app.ActiveSheet;

  //单元格对象
  const cells = await Sheet.Cells;

  //指定单元格
  const cell = await cells.Item(1);
  // 或者 cell = await cells(1)

  //选中第1个单元格
  await cell.Select();
}
```

获取工作表中的列

获取工作表中的列，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.ActiveSheet.Columns
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const Sheet = await app.ActiveSheet;

  //列对象
  const Columns = await Sheet.Columns;

  //指定列
  const column = await Columns.Item(1);
  //或者column = await Columns(1)

  //选中第1个列
  await column.Select();
}
```

获取工作表中的行

获取工作表中的行，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.ActiveSheet.Rows
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //工作表对象
  const Sheet = await app.ActiveSheet;

  //行对象
  const Rows = await Sheet.Rows;

  //指定行
  const row = await Rows.Item(1);
  //或者row = await Rows(1)

  //选中第1行
  await row.Select();
}
```

获取工作表中的所用区域

获取工作表中的所用区域，返回一个Range对象（可使用Range相关的属性、方法）。

- 语法

```
表达式.ActiveSheet.UsedRange
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Applictaion;

  //工作表对象
  const Sheet = await app.ActiveSheet;

  //工作表上的所用区域
  const UsedRange = await Sheet.UsedRange;

  //选中所用区域
  UsedRange.Select();

}
```

监听选区改变事件

- 语法

```
表达式.Sub.Worksheet_SelectionChange = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听选区改变事件
  app.Sub.Worksheet_SelectionChange = (e) => {
    console.log('SelectionChange', e);
  };
}
```

激活工作表的index

- 语法

```
表达式.ActiveSheet.Index
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Applictaion;

  //激活工作表的index
  const Index = await app.ActiveSheet.Index;
  console.log(Index);
}
```

退出编辑

- 语法

```
表达式.ActiveWorkbook.QuitCellEdit()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //退出编辑
  await app.ActiveWorkbook.QuitCellEdit();
}
```

关闭所有打开的下拉面板

- 语法

```
表达式.ActiveWorkbook.CloseDropdownPanels()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //通知所有打开的下拉面板关闭
  await app.ActiveWorkbook.CloseDropdownPanels();
}
```

关闭超链接

- 语法

```
表达式.ActiveWorkbook.CloseHyperLink()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //通知超链接关闭
  await app.ActiveWorkbook.CloseHyperLink();
}
```

关闭筛选hover提示

- 语法

```
表达式.ActiveWorkbook.CloseFilterTips()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //关闭筛选hover提示
  await app.ActiveWorkbook.CloseFilterTips();
}
```

监听滚动画布行为

- 语法

```
表达式.Sub.Worksheet_ScrollChange = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听用户自己的滚动画布行为
  app.Sub.Worksheet_ScrollChange = () => {
    console.log('ScrollChange');
  };
}
```

监听强制横屏通知

- 语法

```
表达式.Sub.Worksheet_ForceLandscape = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听强制横屏时通知
  app.Sub.Worksheet_ForceLandscape = (e) => {
    console.log('ForceLandscape', e);
  };
}
```

6.5.6. 演示

6.5.6.1. 页码

本文介绍使用演示文档时如何获取总页数、监听改变事件、跳转到指定页等。

获取总页数

获取演示文档中的幻灯片总页数。

- 语法

```
表达式.ActivePresentation.Slides.Count
```

表达式：文档类型应用对象

- 返回值

返回 `Number` 表示对应的总页数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取演示文档对象
  const presentation = await app.ActivePresentation;

  //获取幻灯片对象
  const slides = await presentation.Slides;

  //获取总页数
  const count = await slides.Count;
  console.log(count);
}
```

获取当前页码

获取当前幻灯片页码。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.Slide.SlideIndex
```

表达式：文档类型应用对象

- 返回值

返回 `Number` 表示对应的当前页码。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取当前页码
  const curryPage = await app.ActivePresentation.SlideShowWindow.View.Slide.SlideIndex;
  console.log(curryPage);
}
```

监听改变事件

监听当前页改变事件。

- 语法

```
表达式.Sub.SlideSelectionChanged = eventHandle
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听当前页改变事件
  app.Sub.SlideSelectionChanged = async (curryPage) => {
    console.log('切换到：', curryPage);
  };
}
```

跳转到指定页

跳转到指定幻灯片页。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.GotoSlide({ Index })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	跳转到第Index个页面。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //跳转到指定页
  await app.ActivePresentation.SlideShowWindow.View.GotoSlide(3);
}
```

6.5.6.2. 页面

本文介绍演示文件页面的高级接口，包含导出、获取登录操作者信息、缩放、自适应、保存、注脚信息和触发器动画。

导出

导出文件为PDF或者图片。

- 语法

```
表达式.ActivePresentation.ExportAsFixedFormat({ RangeType, FrameSlides, FixedFormatType,
ImgFormat, Dpi, Combine2LongPic, From, To, WaterMark })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述

属性	数据类型	是否必填	描述
RangeType	Enum	可选	幻灯片范围的类型，`Enum.PpPrintRangeType` 取值： ○ 1或ppPrintAll：打印演示文稿中的所有幻灯片。 ○ 2或ppPrintSelection：打印所选幻灯片。 ○ 3或ppPrintCurrent：打印演示文稿中的当前幻灯片。 ○ 4或ppPrintSlideRange：打印若干张幻灯片。 ○ 5或ppPrintNamedSlideShow：打印命名的幻灯片 。
FrameSlides	Enum	可选	导出的幻灯片是否带边框。`Enum.MsoTriState` 取值： ○ -1或msoTrue：给导出的幻灯片加上边框。 ○ 0或msoFalse：不给导出的幻灯片加上边框。
FixedFormatType	Enum	可选	导出的文件类型，`Enum.PpFixedFormatType` 取值： ○ 1或ppFixedFormatTypeXPS：XPS格式，当前版本不支持。 ○ 2或ppFixedFormatTypePDF（默认值）：PDF格式。 ○ 3或ppFixedFormatTypeIMG：IMG格式。

属性	数据类型	是否必填	描述
ImgFormat	Enum	可选	导出图片的格式，当导出文件类型为图片时，可设置此参数。 `Enum.PpExportImgFormatType` 取值： ○ 0或ppImgTypePNG：导出PNG格式的图片。 ○ 1或ppImgTypeJPG：导出JPG格式的图片。 ○ 2或ppImgTypeBMP：导出BMP格式的图片。 ○ 3或ppImgTypeTIF：导出TIF格式的图片。
Dpi	Number	可选	导出的图片质量，当导出文件类型为图片时，可设置此参数。默认值为96。
Combine2LongPic	Boolean	可选	是否输出为长图，当导出文件类型为图片时，可设置此参数。默认值为false。
From	Number	可选	从哪一页导出到哪一页，起始。当导出文件类型为图片时，可设置此参数。默认值为0。
To	Number	可选	从哪一页导出到哪一页，结束。当导出文件类型为图片时，可设置此参数。默认值为0。
WaterMark	Boolean	可选	是否带水印。默认值为false。

- 返回值

  返回打印的 `url` 。

属性	数据类型	描述
url	string	打印的url。

- 示例

- 导出文件为PDF

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出文件为PDF
  const pdfUrl = await app.ActivePresentation.ExportAsFixedFormat();
  console.log(pdfUrl);
}
```

- 导出文件为图片

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出文件为图片
  const pdfUrl = await app.ActivePresentation.ExportAsFixedFormat({
    FixedFormatType: app.Enum.PpFixedFormatType.ppFixedFormatTypeIMG,
    From: 0,
    To: 1,
  });
  console.log(pdfUrl);
}
```

登录操作者信息

获取登录操作者信息。

- 语法

```
表达式.ActivePresentation.GetOperatorsInfo()
```

表达式：文档类型应用对象

- 返回值

Object object

属性	数据类型	描述
type	String	请求类型。
response	Object	用户信息列表。

response结构

属性	数据类型	描述
id	String	用户id。
avatar_url	Object	用户头像。
logined	Object	用户登录状态。
name	Object	用户姓名。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取登录操作者信息
  const operatorsInfo = await app.ActivePresentation.GetOperatorsInfo();
  console.log(operatorsInfo);
}
```

缩放

窗口缩放。

- 语法

```
表达式.ActiveWindow.View.Zoom
```

表达式：文档类型应用对象

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //文档窗口对象
  const activeWindow = await app.ActiveWindow;

  //视图对象
  const view = await activeWindow.View;

  //获取窗口缩放比例
  const result = await view.Zoom;
  console.log(result);

  //设置窗口缩放比例
  view.Zoom = 66;
}
```

自适应

对文档窗口大小进行调整后，是否缩放视图以适应文档窗口的尺寸。

- 语法

  `表达式.ActiveWindow.View.ZoomToFit = Number`

  - 表达式：文档类型应用对象
  - `Number` 依据不同值进行调整：
    - `-1` ：文档窗口大小进行调整后使页面以最佳尺寸适应活动窗口。
    - `0` ：对文档窗口大小进行调整后不为适应文档窗口尺寸而缩放视图。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //文档窗口对象
  const activeWindow = await app.ActiveWindow;

  //视图对象
  const view = await activeWindow.View;

  //窗口自适应
  view.ZoomToFit = 0;
}
```

保存

> **注意**
>
> 支持JS-SDK v1.1.9及以上版本。

保存文件的改动。

- 语法

```
表达式.ActivePresentation.Save()
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
result	String	保存状态。
size	Number	文件大小，单位byte。
version	Number	版本。

保存状态说明

保存状态	描述
ok	版本保存成功，可在历史版本中查看。
nochange	文档无更新，无需保存版本。
SavedEmptyFile	暂不支持保存空文件。触发场景：内核保存完后文件为空。
SpaceFull	空间已满。
QueneFull	保存中请勿频繁操作。触发场景：服务端处理保存队列已满，正在排队。
fail	保存失败。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //保存文件的改动
  const saveResult = await app.ActivePresentation.Save();
  console.log(saveResult);
}
```

注脚信息

> **注意**
>
> 支持JS-SDK V1.1.15及以上版本。

获取当前页或指定页的注脚信息

- 语法

  `表达式.ActivePresentation.GetSlideRemark()`

  表达式：文档类型应用对象

- 参数

  传入一个数字，表明指定页的序列号。

属性	数据类型	是否必填	描述
SlideIndex	`Number`	是	指定页的序列号。默认值为当前激活页的序列号。

- 返回值

属性	数据类型	描述
remarkList	Array	注脚信息列表。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取指定页或当前页的注脚信息
  const operatorsInfo = await app.ActivePresentation.GetSlideRemark();
  console.log(operatorsInfo);
}
```

获取所有页的注脚信息

- 语法

```
表达式.ActivePresentation.GetRemarkList()
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
remarkList	Array	注脚信息列表。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取所有页的注脚信息
  const operatorsInfo = await app.ActivePresentation.GetRemarkList();
  console.log(operatorsInfo);
}
```

执行触发器动画

> **注意**
>
> 支持JS-SDK V1.1.15及以上版本。

- 语法

```
表达式.ActivePresentation.PlayTriggerAnim()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
TargetShapeId	number	是	触发器ID。
SlideCategory	string	是	触发器类型。
Steps	number	否	指定步数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //执行触发器动画
  await app.ActivePresentation.PlayTriggerAnim(TargetShapeId, SlideCategory);
}
```

6.5.6.3. 页面设置

本文介绍演示文件页面设置相关高级接口，包含页面设置对象、页面宽度、页面比例和页面高度。

页面设置对象

- 语法

```
表达式.ActivePresentation.PageSetup
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //演示文稿对象
  const presentation = await app.ActivePresentation;

  //页面设置对象
  const pageSetup = await presentation.PageSetup;
}
```

页面宽度

- 语法

```
表达式.ActivePresentation.PageSetup.SlideWidth
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //演示文稿对象
  const presentation = await app.ActivePresentation;

  //页面设置对象
  const pageSetup = await presentation.PageSetup;

  //获取宽度
  const slideHeight = await pageSetup.SlideHeight;
  console.log(slideHeight);
}
```

页面高度

- 语法

```
表达式.ActivePresentation.PageSetup.SlideHeight
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //演示文稿对象
  const presentation = await app.ActivePresentation;

  //页面设置对象
  const pageSetup = await presentation.PageSetup;

  //获取高度
  const slideHeight = await pageSetup.SlideHeight;
  console.log(slideHeight);
}
```

页面比例

- 语法

```
表达式.ActivePresentation.PageSetup.SlideRatio
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
Ratio	number	页面比例。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //演示文稿对象
  const presentation = await app.ActivePresentation;

  //页面设置对象
  const pageSetup = await presentation.PageSetup;

  //获取比例
  const slideHeight = await pageSetup.SlideRatio;
  console.log(slideHeight);
}
```

6.5.6.4. 幻灯片

本文介绍演示幻灯片高级接口，包含幻灯片对象、全屏、动画、播放、幻灯片播放状态、打开关闭播放缩略图的方法以及插入新的幻灯片。

幻灯片对象

- 语法

```
表达式.ActivePresentation.Slides
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //演示文稿对象
  const presentation = await app.ActivePresentation;

  //幻灯片对象
  const view = await presentation.Slides;
}
```

全屏

进入全屏

切换到PPT全屏播放状态。

- 语法

```
表达式.ActivePresentation.SlideShowSettings.Run()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();
}
```

监听进入全屏

监听进入幻灯片全屏播放。

- 语法

```
表达式.Sub.SlideShowBegin = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听进入幻灯片全屏播放
  app.Sub.SlideShowBegin = async () => {
    console.log('进入');
  };

  setTimeout( async () => {
    //切换到PPT全屏播放状态
    await app.ActivePresentation.SlideShowSettings.Run();
  }, 2000);
}
```

退出全屏

退出PPT全屏播放状态。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.Exit()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到幻灯片全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //退出PPT全屏播放状态
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.Exit();
  }, 5000);
}
```

监听退出全屏

监听退出幻灯片全屏播放。

- 语法

```
表达式.Sub.SlideShowEnd = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //监听退出幻灯片全屏播放
  app.Sub.SlideShowEnd = async () => {
    console.log('退出',);
  };

  //3000 ms后退出全屏播放
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.Exit();
  }, 3000);
}
```

动画

动画总数量

获取当前页幻灯片动画的个数。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.GetClickCount()
```

表达式：文档类型应用对象

- 返回值

返回 `Number` 类型的值表示当前幻灯片对应的动画数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //获取当前页幻灯片动画的个数
  const clickCount = await app.ActivePresentation.SlideShowWindow.View.GetClickCount();
  console.log(clickCount);
}
```

当前页数量

获取鼠标单击的正在幻灯片上播放或刚刚播放完成的动画的索引号。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.GetClickIndex()
```

表达式：文档类型应用对象

- 返回值

返回 `Number` 类型的值，表示当前鼠标单击动画的索引号。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //获取当前鼠标单击动画的索引号
  const clickCount = await app.ActivePresentation.SlideShowWindow.View.GetClickIndex();
  console.log(clickCount);
}
```

下一步动画

切换到下一步动画。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.GotoNextClick()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //切换到下一步动画
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.GotoNextClick();
  }, 2000);
}
```

监听下一步动画

监听下一步动画操作。

- 语法

```
表达式.Sub.SlideShowOnNext = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //监听下一步动画操作
  app.Sub.SlideShowOnNext = async () => {
    console.log('下一步');
  };

  //3000 ms后执行下一步操作
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.GotoNextClick();
  }, 3000);
}
```

上一步动画

切换到上一步动画。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.GotoPreClick()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //切换到上一步动画
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.GotoPreClick();
  }, 2000);
}
```

监听上一步动画

监听上一步动画操作。

- 语法

```
表达式.Sub.SlideShowOnPrevious = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();

  //监听上一步动画操作
  app.Sub.SlideShowOnPrevious = async () => {
    console.log('上一步');
  };

  //3000 ms后执行下一步
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.GotoNextClick();
  }, 3000);

  //5000 ms后执行上一步
  setTimeout( async () => {
    await app.ActivePresentation.SlideShowWindow.View.GotoPreClick();
  }, 5000);
}
```

播放

> **注意**
>
> 支持JS-SDK V1.1.15及以上版本。

显示或隐藏页码（PC端）

- 语法

  `表达式.ActivePresentation.SlideShowWindow.View.ShowPage`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //视图对象
  const view = await SlideShowWindow.View;

  //播放模式下显示页码
  view.ShowPage = true;
}
```

显示或隐藏工具栏

- 语法

```
表达式.ActivePresentation.SlideShowSettings.ShowPlayToolbar
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //显示隐藏工具栏
  app.ActivePresentation.SlideShowSettings.ShowPlayToolbar = false;
}
```

修改工具栏样式

> **注意**
>
> 该高级接口自由度很高，为了避免样式的修改影响功能使用，请尽量使用 `top` 、 `left` 、 `right` 、 `bottom` 、 `translate` 修改样式。

- 语法

```
表达式.ActivePresentation.SlideShowSettings.SetPlayToolbarPosition({ Style })
```

表达式：文档类型应用对象

- 参数

传入一个布尔值，打开或者关闭缩略图。

属性	数据类型	是否必填	描述
Style	Object	是	样式。

Style说明

属性	数据类型	描述
Show	Object	显示时的样式，传入参数类似为 `{ left: '100px' }` 。
Hidden	Object	隐藏时的样式。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //修改工具栏样式
  await SlideShowSettings.SetPlayToolbarPosition({
    Style: {
      Show: { top: '10px' }, //style样式，工具栏显示时的位置
      Hidden: { top: '-100px' }, //style样式，工具栏隐藏时的位置
    },
  });
}
```

修改画笔工具样式

- 语法

```
表达式.ActivePresentation.SlideShowSettings.SetPlayInkPosition({ Style })
```

表达式：文档类型应用对象

- 参数

传入一个布尔值，表明打开或者关闭缩略图。

属性	数据类型	是否必填	描述
Style	Object	是	样式，传入参数类似为 `{ left: '100px' }` 。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //修改画笔工具样式
  await SlideShowSettings.SetPlayInkPosition({
    Style: { background: 'deepskyblue' }, //画笔工具样式
  });
}
```

打开或保存画笔并退出

- 语法

```
表达式.ActivePresentation.SetOpenMarkerInkEdit({ Open })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Open	Boolean	否	打开或保存画笔并退出。 ○ true（默认）：开启画笔。 ○ false：保存画笔并关闭。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //开启画笔
  await app.ActivePresentation.SetOpenMarkerInkEdit(true);

  setTimeout(async () => {
    //保存画笔并关闭
    await app.ActivePresentation.SetOpenMarkerInkEdit(false);
  }, 5000);
}
```

打开备注视图

- 语法

```
表达式.ActivePresentation.SlideShowSettings.OpenRemarkSpeaker()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //打开备注视图
  await SlideShowSettings.OpenRemarkSpeaker();
}
```

打开倒计时

- 语法

```
表达式.ActivePresentation.SlideShowSettings.SetCountDown()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //打开倒计时
  await SlideShowSettings.SetCountDown();
}
```

开启自动播放

- 语法

```
表达式.ActivePresentation.SlideShowSettings.StartAutoPlay()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //开启自动播放
  await SlideShowSettings.StartAutoPlay();
}
```

幻灯片播放状态

获取当前幻灯片播放状态。

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.State
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
state	String	幻灯片状态。 ○ edit：编辑。 ○ play：播放。 ○ masterView：母版视图。

- 示例：

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取状态
  const getState = () => {
    return app.ActivePresentation.SlideShowWindow.View.State;
  };

  //获取当前幻灯片播放状态
  let currentState = await getState();
  console.log(currentState); // 'edit'

  //切换到PPT全屏播放状态
  await app.ActivePresentation.SlideShowSettings.Run();
  currentState = await getState();
  console.log(currentState); // 'play'
}
```

打开或关闭播放缩略图

- 语法

```
表达式.ActivePresentation.SlideShowSettings.SetMiniThumbnailVisible()
```

表达式：文档类型应用对象

- 参数

传入一个布尔值，表示打开或者关闭缩略图。

属性	数据类型	是否必填	描述
Visible	Boolean	是	是否打开缩略图。 ○ false（默认）：关闭。 ○ true：打开。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //打开缩略图
  await app.ActivePresentation.SlideShowSettings.SetMiniThumbnailVisible(true);
}
```

插入新的幻灯片

- 语法

```
表达式.ActivePresentation.Slides.AddSlide()
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	number	否	新幻灯片插入的位置。不传或参数小于0时插入到当前幻灯片的后面。
CustomLayout	object	否	本地版式的类型。不传时默认插入空白幻灯片。
LayoutUrl	string	否	在线PPT文件的链接。如果传递了该参数，layoutType会被忽略，优先使用在线PPT文件内的版式，并返回一个promise通知结果。
LayoutIndex	number	否	新建幻灯片时需要使用在线文件的第几个版式，layoutUrl不传时，该参数无效。

6.5.6.5. 会议

本文介绍使用演示文档时如何获取页码、保存文件和开启自动播放等。

> **注意**
>
> JS-SDK V1.1.15及以上版本支持该功能。

标注

打开标注工具栏

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.MarkerEditVisible
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //打开标注工具栏
  view.MarkerEditVisible = true;

  setTimeout(() => {
    //3000 ms后关闭标注工具栏
    view.MarkerEditVisible = false;
  }, 3000)

}
```

显示标注内容

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.PointerVisible
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //监听标注内容是否显示事件
  app.Sub.SlideInkVisible = async (e) => {
    const { Data } = e;
    console.log(Data.showmark);

    setTimeout(() => {
      //3000 ms后设置标注内容的显示隐藏
      view.PointerVisible = !Data.showmark;
    }, 3000)


  }

}
```

监听显示标注事件

- 语法

  `表达式.Sub.SlideInkVisible = eventHandle`

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View

  //监听是否显示标注内容
  app.Sub.SlideInkVisible = ({Data}) => {
    console.log('监听是否显示标注内容：', Data);

    if(Data.showmark){
      setTimeout(() => {
        view.PointerVisible = false;
      }, 2000)

    }
  };
}
```

播放

播放幻灯片

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.Play(slideIndex, animateIndex)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	描述
slideIndex	Number	当前幻灯片的下标。
animateIndex	Number	当前幻灯片动画的下标。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;
  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //从第一张幻灯片的第一个动画开始播放
  await view.Play(1, 1);
}
```

跳转幻灯片动画

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.GotoClick(number)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	描述
number	Number	当前幻灯片动画的下标。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;
  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //执行跳转幻灯片第二个动画
  await view.GotoClick(2);
}
```

禁止翻页

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.IsBanSlideChange
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //禁止翻页
  view.IsBanSlideChange = true;

  setTimeout(() => {
    //3000 ms后允许翻页
    view.IsBanSlideChange = false;
  }, 3000)


}
```

停止自动翻页

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.StopAutoPlay()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;

  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //关闭自动播放
  await SlideShowSettings.StopAutoPlay();
}
```

预览模式

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.CoreReadonly
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  view.CoreReadonly = true;

}
```

设置全屏

- 语法

```
表达式.ActivePresentation.SlideShowWindow.IsFullScreen
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //设置全屏
  SlideShowWindow.IsFullScreen = true;

  setTimeout(() => {
    //3000 ms后退出全屏
    SlideShowWindow.IsFullScreen = false;
  }, 3000)


}
```

监听视频播放状态

- 语法

```
表达式.Sub.SlideMediaChanged = eventHandle
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听视频播放状态改变
  app.Sub.SlideMediaChanged = ({Data}) => {
    console.log('监听视频播放状态改变：', Data);
  };
}
```

观众同步视频播放信息

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.SetMediaObj(Data)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	描述
Data	Object	视频播放信息。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //设置视频播放改变监听事件
  app.Sub.SlideMediaChanged = async (e) => {
    const { Data } = e;
    console.log(Data);

    //同步视频播放信息
    await view.SetMediaObj(Data);
  }

}
```

设置销毁视频播放工具（移动端）

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.DestroyMediaPlayer()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //设置视频播放改变监听事件
  app.Sub.SlideMediaChanged = async (e) => {
    const { Data } = e;
    console.log(Data);
    setTimeout(async () => {
      //3000 ms后销毁视频播放工具
      await view.DestroyMediaPlayer(Data);
    }, 3000);
  }
}
```

激光笔

同步激光笔信息

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.SetLaserPenData(Data)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	描述
Data	Object	激光笔信息。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  //设置发送激光笔的墨迹监听事件
  app.Sub.SlideLaserPenInkPointsChanged = async (e) => {
    const { Data } = e;
    console.log(Data);

    //同步激光笔信息
    await view.SetLaserPenData(Data);
  }

}
```

监听发送激光笔的墨迹

- 语法

```
表达式.Sub.SlideLaserPenInkPointsChanged = eventHandle
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听发送激光笔的墨迹
  app.Sub.SlideLaserPenInkPointsChanged = ({Data}) => {
    console.log('监听发送激光笔的墨迹：', Data);
  };
}
```

监听是否使用激光笔和标注

- 语法

```
表达式.Sub.SlideInkToolbarVisible = eventHandle
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View

  //监听是否使用激光笔和标注
  app.Sub.SlideInkToolbarVisible = ({Data}) => {
    console.log('监听是否使用激光笔和标注：', Data);

  };

  //设置标记工具栏出现
  view.MarkerEditVisible = true;
}
```

设置指定工具栏是否隐藏

- 语法

```
表达式.ActivePresentation.SlideShowWindow.View.SetToolVisible(toolName, flag)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	描述
toolName	String	需要操作的tool的名称。
flag	Boolean	是否隐藏指定的tool。取值范围如下： ○ false：隐藏。 ○ true：显示。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取Slide设置对象
  const SlideShowSettings = await app.ActivePresentation.SlideShowSettings;
  //进入幻灯片播放模式
  await SlideShowSettings.Run();

  //获取窗口对象
  const SlideShowWindow = await app.ActivePresentation.SlideShowWindow;

  //获取视图对象
  const view = await SlideShowWindow.View;

  const linkTip = app.Enum.PpToolType.pcPlayHoverLink //hover超链接
  const imageTip = app.Enum.PpToolType.pcImageHoverTip //hover图片
  const menu = app.Enum.PpToolType.pcPlayingMenu //右键菜单

  //2000 ms后设置隐藏
  setTimeout(() => {
    //设置进入播放模式后隐藏指定的tool
    view.SetToolVisible(linkTip, false);
    view.SetToolVisible(imageTip, false);
    view.SetToolVisible(menu, false);
  }, 2000)
}
```

6.5.7. PDF

6.5.7.1. 页码

本文介绍使用PDF文档时如何获取总页数、监听改变事件、跳转到指定页等。

获取总页数

获取PDF文档总页数。

- 语法

  `表达式.ActivePDF.PagesCount`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 表示总页数。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取总页数
  const totalPages = await app.ActivePDF.PagesCount;
  console.log(totalPages);
}
```

获取当前页码

获取PDF文档当前页码。

- 语法

  `表达式.ActivePDF.CurrentPage`

  表达式：文档类型应用对象

- 返回值

  返回 `Number` 表示当前页码。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取当前页码
  const curryPage = await app.ActivePDF.CurrentPage;
  console.log(curryPage);
}
```

监听改变事件

监听当前页改变事件。

- 语法

  `表达式.Sub.CurrentPageChange = eventHandle`

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //监听当前页改变事件
  app.Sub.CurrentPageChange  = async (curryPage) => {
    console.log('切换到：', curryPage);
  };
}
```

跳转到指定页

跳转到指定PDF页。

- 语法

```
表达式.ActivePDF.JumpToPage(PageNum)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
PageNum	Number	是	跳转到的页码。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //跳转到指定页
  await app.ActivePDF.JumpToPage(3);
}
```

6.5.7.2. 页面

本文档介绍如何获取登录操作者信息、分页和连页模式切换、目录隐藏、保存文件、导出文件的类型以及获取页面高度、获取页面宽度和页面坐标。

获取登录操作者信息

- 语法：

```
表达式.ActivePDF.GetOperatorsInfo()
```

表达式：文档类型应用对象

- 返回值：

  Object object

属性	数据类型	描述
type	String	请求类型。
response	Object	用户信息列表。

response结构：

属性	数据类型	描述
id	String	用户ID。
avatar_url	Object	用户头像。
logined	Object	用户登录状态。
name	Object	用户姓名。

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取登录操作者信息
  const operatorsInfo = await app.ActivePDF.GetOperatorsInfo();
  console.log(operatorsInfo);
}
```

分页和连页模式切换

- 语法

```
表达式.ActivePDF.PageMode = Number
```

表达式：文档类型应用对象

赋值Number。1为单页模式，0为多页模式。

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //单页/多页模式切换
  app.ActivePDF.PageMode = 1;
}
```

目录显示隐藏

控制目录的显示隐藏。

- 语法

```
表达式.ActivePDF.DocumentMap = Boolean
```

表达式：文档类型应用对象

赋值Boolean。true为显示目录，false为隐藏目录。

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //控制目录的显示隐藏
  app.ActivePDF.DocumentMap = true;
}
```

保存

保存文件的改动。

- 语法

```
表达式.ActivePDF.Save()
```

表达式：文档类型应用对象

- 返回值

属性	数据类型	描述
result	String	保存状态。
size	Number	文件大小，单位为字节。
version	Number	版本。

保存状态说明如下表。

保存状态	描述
ok	版本保存成功，可在历史版本中查看。
nochange	文档无更新，无需保存版本。
SavedEmptyFile	暂不支持保存空文件。 触发场景：内核保存完后文件为空。
SpaceFull	空间已满。
QueneFull	保存中请勿频繁操作。 触发场景：服务端处理保存队列已满，正在排队。
fail	保存失败。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //保存文件的改动
  const saveResult = await app.ActivePDF.Save();
  console.log(saveResult);
}
```

导出图片文件

- 语法

```
表达式.ActivePDF.ExportAsFixedFormat({ FixedFormatType })
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
FixedFormatType	Enum	可选	导出的文件类型，当前版本仅支持导出图片。默认值为0。

属性	数据类型	是否必填	描述
Print	Boolean	可选	是否直接打开。取值范围如下： ○ false（默认）：否。 ○ true：是。
ImgFormat	Enum	可选	导出图片的格式。Enum.ExportImgFormatType取值： ○ 0或ImgTypePNG（默认）：PNG格式。 ○ 1或ImgTypeJPG：JPG格式。 ○ 2或ImgTypeBMP：BMP格式。 ○ 3或ImgTypeTIF：TIF格式。
Dpi	Number	可选	图片质量。默认值96。
Combine2LongPic	Boolean	可选	是否导出长图。取值范围如下： ○ false（默认）：否。 ○ true：是。
RangeType	Enum	可选	导出范围。Enum.RangeType取值： ○ 0或ImgTypeAll（默认）：所有页面。 ○ 1或ImgTypePage：指定范围。 ○ 2或ImgTypeCurrent：当前幻灯片。
WaterMark	Boolean	可选	是否含水印。取值范围如下： ○ false（默认）：否。 ○ true：是。
From	Boolean	可选	起始页，默认值为当前页。
To	Boolean	可选	结束页，默认值为当前页。

- 返回值

  返回打印的URL。

属性	数据类型	描述
url	string	打印的URL。

- **示例**

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //导出文件为图片
  const imgUrl = await app.ActivePDF.ExportAsFixedFormat({
    FixedFormatType: app.Enum.FixedFormatType.TypeIMG,
  });
  console.log(imgUrl);
}
```

获取页面宽度

获取指定页码页面的宽度。

> **注意**
>
> JS-SDK V1.1.15及以上版本支持此功能。

- 语法

```
表达式.ActivePDF.GetPageWidth(PageIndex)
```

表达式：文档类型应用对象

- 参数

参数	数据类型	是否必填	描述
PageIndex	Number	是	页码（索引从1开始）。

- 返回值

返回 `Number` 表示指定页码页面的宽度。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取指定页码页面的宽度，索引从1开始
  const pageWidth = await app.ActivePDF.GetPageWidth(1);
  console.log(pageWidth);
}
```

获取页面高度

获取指定页码页面的高度。

> **注意**
>
> JS-SDK V1.1.15及以上版本支持此功能。

- 语法

  `表达式.ActivePDF.GetPageHeigh(PageIndex)`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
PageIndex	Number	是	页码（索引从1开始）。

- 返回值

  返回 `Number` 表示指定页码页面的高度。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取指定页码页面的高度，索引从1开始
  const pageHeight = await app.ActivePDF.GetPageHeight(1);
  console.log(pageHeight);
}
```

获取页面坐标

通过相对窗口页面的坐标获取原始PDF页面的坐标和页码。

- 语法

```
表达式.ActivePDF.GetPointPageInfoByWinPoint(x, y)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
x	Number	是	相对窗口页面的x坐标。
y	Number	是	相对窗口页面的y坐标。

- 返回值

Object object

属性	数据类型	描述
inContentRange	Boolean	传入坐标是否在PDF正文区域（值为false代表不在正文区域，此时不会有其他的属性）。
pageIndex	Number	传入坐标对应原始PDF的所在页的页码。
x	Number	传入坐标对应原始PDF页面的x坐标。
y	Number	传入坐标对应原始PDF页面的y坐标。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //通过相对窗口的坐标，获取到对应的点在原始页面的坐标和页码
  const pointPageInfo = await app.ActivePDF.GetPointPageInfoByWinPoint(500,500);
  console.log(pointPageInfo);
}
```

6.5.7.3. 缩放

PDF高级用法支持获取和设置窗口缩放比例和自适应文档窗口。

获取和设置窗口缩放比例

- 语法

`表达式.ActivePDF.Zoom`

表达式：文档类型应用对象

- 返回值

  返回 `Number` 型的缩放比例

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取窗口缩放比例
  const result = await app.ActivePDF.Zoom;
  console.log(result);

  //设置窗口缩放比例
  app.ActivePDF.Zoom = 66;
}
```

自适应

对文档窗口大小进行调整后，是否缩放视图以适应文档窗口的尺寸。

- 语法

  `表达式.ActivePDF.ZoomToFit = Number`

  表达式：文档类型应用对象

  `Number` 依据不同值进行调整：

  - -1：文档窗口大小进行调整后使页面以最佳尺寸适应活动窗口。
  - 0：文档窗口大小进行调整后不为适应文档窗口尺寸而缩放视图。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //窗口自适应
  app.ActivePDF.ZoomToFit = 0;
}
```

6.5.7.4. 滚动

本文介绍PDF文档如何获取滚动条位置、设置滚动条位置和监听滚动事件。

获取滚动条位置

- 语法

  `表达式.ActivePDF.Scroll`

  表达式：文档类型应用对象。

- 返回值

  返回滚动条位置{x: xx, y: xx}。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  // 获取滚动位置
  const result = await app.ActivePDF.Scroll;
  console.log(result);

}
```

设置滚动条位置

- 语法

  `表达式.ActivePDF.ScrollTo(x, y)`

  表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
x	Number	否	滚动位置的x坐标。
y	Number	否	滚动位置的y坐标。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  await app.ActivePDF.ScrollTo(100, 100);

}
```

监听滚动事件

- 语法

```
表达式.Sub.Scroll = eventHandle
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  // ActivePDF
  const pdf = await app.ActivePDF;

  // 监听滚动事件
  app.Sub.Scroll = (e) => {
    console.log('Scroll', e)
  };

  // 设置滚动条位置
  setTimeout(async () => {
    await pdf.ScrollTo(100, 200)
  }, 2000);
}
```

6.6. 页面定制

6.6.1. 页面定制概述

页面定制是指通过调用API控制页面中的每个元素。本文通过Word文档案例，帮助您了解如何使用页面定制。

案例一：控制已有组件

- 定制需求

  隐藏页面左上角表示更多菜单的按钮 ，如下图所示。

- 语法

```
表达式.CommandBars(CommandBarId).Visible = false
```

表达式：文档类型应用对象

其中 `CommandBars` 表示调用的对象， `CommandBarId` 表示需要控制的元素。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面定制对象：更多菜单
  const moreMenus = await app.CommandBars('MoreMenus');

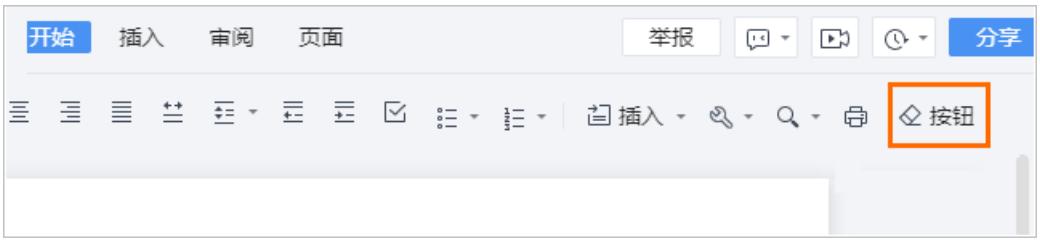
  //控制更多菜单隐藏
  moreMenus.Visible = false;
}
```

关于页面中更多可以控制的元素，请参见可定制列表。

案例二：自定义组件

- 定制需求

  通过 `CommandBars` 自定义组件，设置组件的元素图标、标题等属性，如下图所示。



- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面定制对象：开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加定制元素：按钮
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮'; //设置标题
  //设置Icon
  controlButton.Picture = '
IHZpZXdCb3g9IjAgMCAxNiAxNiIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIj48ZyBmaWxsPSIjM
0Q0NzU3IiBmaWxsLXJ1bGU9ImV2ZW5vZGQiPjxwYXRoIGQ9Ik04LjIxMyAxM0g2LjhsNi42MzYtNi42MzYtNC4yND
MtNC4yNDMtNy4wNyA3LjA3MUw1LjkyOCAxM0g0LjUxNUwxLjA2IDkuNTQ2YS41LjUgMCAwIDEgMC0uNzA3TDguODM
5IDEuMDZhLjUuNSAwIDAgMSAuNzA3IDBsNC45NSA0Ljk1YS41LjUgMCAwIDEgMCAuNzA3TDguMjEzIDEzeiIgZmls
bC1ydWxlPSJub256ZXJvIi8+PHBhdGggZD0iTTQuNTM2IDYuMzY0bDQuOTUgNC45NS0uNzA3LjcwNy00Ljk1LTQuO
TV6TTQuNTIxIDEzaDEwLjAzdjFINS40OTZ6Ii8+PC9nPjwvc3ZnPg==';
  controlButton.SetFocus(true); //聚焦该元素
}
```

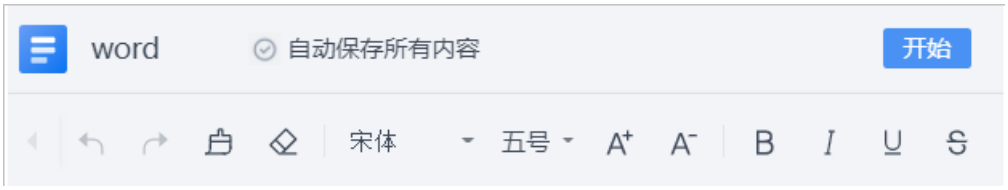
其他说明

页面定制功能适用于各个场景（Word、Excel、PPT、PDF），为了避免混淆，后续页面定制相关功能都将以Word为例进行演示。

6.6.2. 隐藏或显示组件

通过设置 `Visible` 属性，您可以控制组件的显示或隐藏。本文以更多菜单按钮为例，介绍如何隐藏组件。

- 效果演示
  - 隐藏按钮



  - 显示按钮

- 语法

```
表达式.CommandBars(CommandBarId).Visible
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面定制对象：更多菜单
  const moreMenus = await app.CommandBars('MoreMenus');

  //控制更多菜单隐藏
  moreMenus.Visible = false;

  //获取更多菜单状态
  const visible = await moreMenus.Visible;
  console.log(visible);
}
```

6.6.3. 禁用或启用组件

通过设置 `Enabled` 属性，您可以控制组件的禁用或启用。本文以更多菜单按钮为例，介绍如何禁用组件。

- 效果演示

- 禁用按钮（置灰）

- 启用按钮

- 语法

```
表达式.CommandBars(CommandBarId).Enabled
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面定制对象：更多菜单
  const moreMenus = await app.CommandBars('MoreMenus');

  //控制更多菜单禁用
  moreMenus.Enabled = false;

  //获取更多菜单禁用状态
  const enabled = await moreMenus.Enabled;
  console.log(enabled);
}
```

6.6.4. 控制按钮单击

通过设置 `Execute()` 属性，您可以控制组件的单击。本文以更多菜单按钮为例，介绍如何在需要时控制按钮自行单击打开。

- 效果演示
  - 未单击

- 单击后

- 语法

```
表达式.CommandBars(CommandBarId).Execute()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面定制对象更多菜单
  const moreMenus = await app.CommandBars('MoreMenus');

  //3000 ms后触发具体事件
  setTimeout( async () => {
    //单击更多菜单
    await moreMenus.Execute();
  }, 3000);
}
```

6.6.5. 搜索指定内容

本文介绍如何根据搜索的文本，调用FloatQuickHelp（右下角浮动帮助按钮）对指定的内容进行搜索。

- 语法

```
表达式.CommandBars(CommandBarId).Search(Text)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Text	String	是	搜索的关键字。

- 返回值

  返回的类型为 `Array.<Object>` ，返回的内容有以下2种情况：

  - 普通的功能按钮

属性	数据类型	描述
iconClassName	String	图标的类名。
text	String	按钮文本。
shortcut	String	快捷键。

  - 对应的文档链接

属性	数据类型	描述
content	String	搜索结果。
url	String	跳转链接。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取页面定制对象：FloatQuickHelp
  const floatQuickHelp = await app.CommandBars('FloatQuickHelp');

  //通过FloatQuickHelp搜索【打印】相关信息
  const data = await floatQuickHelp.Search('打印');
  console.log(data);
}
```

6.6.6. 可定制列表

在页面定制过程中，您需要指定对应的 `CommandBarId` 来操作相关组件，本文介绍文档在线协作提供的 `CommandBarIds` 列表。

分类

`CommandBarIds` 包括公共、Word（文字）、Excel（表格）、PPT（演示）和PDF五个部分。每个部分根据使用平台不同又分为PC端和Mobile端两部分。

公共

- **PC端**

CommandBarId	位置
TabPrintPreview	工具栏>打印
FloatQuickHelp	右下角浮动帮助按钮
HeaderLeft	头部左侧，头部一共分左、中、右三块区域
HeaderMiddle	头部中部，头部一共分左、中、右三块区域
HeaderRight	头部右侧，头部一共分左、中、右三块区域
HistoryRecord	顶部状态栏>历史记录菜单>协作记录
HistoryVersion	顶部状态栏>历史记录菜单>历史版本
MenuPrintPreview	更多菜单>打印按钮
HeaderHistoryMenuBtn	右边菜单>历史版本入口按钮
PreviewDownload	顶部状态栏>历史记录菜单>历史版本>预览版本>下载
MoreMenus	头部>更多菜单

- **Mobile端**

CommandBarId	位置
Cooperation	状态栏协作头像
More	状态栏更多按钮
Logo	状态栏Logo
SendButton	顶部工具栏>分享按钮
Invitation	顶部工具栏>邀请按钮

文字

- **PC端**

CommandBarId	位置
StartTab	工具栏>开始Tab
InsertTab	工具栏>插入Tab
ReviewTab	工具栏>审阅Tab
PageTab	工具栏>页面Tab
TabStartTab	工具栏>开始按钮
TabInsertTab	工具栏>插入按钮
TabPageTab	工具栏>页面按钮
TabReviewWord	工具栏>审阅按钮
TaskPane	任务窗格（右侧边栏）
ContextMenuConvene	右键>召唤在线协助者

CommandBarId	位置
ReviewTrackChanges	审阅>开启修订按钮
BookMark	插入>书签
PanelBookMark	开始Tab>插入>书签
RightMenuHistory	文字>右边菜单>历史记录>查看最新改动

- **Mobile端**

CommandBarId	位置
WriterHoverToolbars	文字>底部工具栏
Meeting	底部>会议按钮
Print	底部>打印按钮
FontTab	字体属性面板
Arrange	段落属性面板
Insert	插入对象属性面板
Review	修订评论属性面板
Keyboard	键盘
InsertImage	插入图片

表格

- **PC端**

CommandBarId	位置
HistoryVersionDivider	右键菜单>历史版本/协作记录分割线
StartTab	开始Tab
InsertTab	插入Tab
DataTab	数据Tab
FormulaTab	公式Tab
CooperationTab	协作Tab
ViewTab	视图Tab
TabStartTab	工具栏>开始按钮
TabInsertTab	工具栏>插入按钮
TabDataTab	工具栏>数据按钮
TabFormulaTab	工具栏>公式按钮
TabCooperationTab	工具栏>协作按钮
TabViewTab	工具栏>视图按钮
TaskPane	任务窗格
ContextMenuHistory	单元格右键>单元格最近的改动
RightMenuHistory	表格>右边菜单>历史记录>单元格最近改动
RightMenuHistoryToady	表格>右边菜单>历史记录>今天的改动

CommandBarId	位置
RightMenuHistoryThreeDay	表格>右边菜单>历史记录>最近3天的改动
RightMenuHistorySevenDay	表格>右边菜单>历史记录>最近7天的改动

- **Mobile端**

CommandBarId	位置
FormatTab	格式Tab
InsertTab	插入Tab
DataTab	数据Tab
ViewTab	视图Tab
Keyboard	键盘

演示

- **PC端**

CommandBarId	位置
InsertTab	工具栏>插入Tab
ReviewTab	工具栏>审阅Tab
PlayTab	工具栏>播放Tab
ViewTab	工具栏>视图Tab
TabInsertTab	工具栏>插入按钮

CommandBarId	位置
TabReviewTab	工具栏>审阅按钮
TabPlayTab	工具栏>播放按钮
TabViewTab	工具栏>视图Tab
TabThumbNumber	母版视图>slide hover缩略图
TaskPane	任务窗格
PlayComponentToolbar	全屏播放时toolbar
WPPPcCommentButton	底部工具栏>评论按钮
Comment	审阅Tab>评论按钮
MeetingEnter	播放Tab>会议入口
ToolBarFullScreenBtn	全屏播放toolbar全屏按钮
ContextDownloadImage	右键>下载原图
TaskbarDownloadImage	右侧工具栏>下载原图
PlayContextCheckImage	播放右键>查看原图
FloatMenuCheckImage	选中图片浮动菜单栏>查看原图

- **Mobile端**

CommandBarId	位置
WPPMobileCommentButton	底部工具栏>评论按钮

CommandBarId	位置
WPPMobileMarkButton	浮动备注按钮
property	格式Tab
insert	插入Tab
comment	评论Tab
keyboard	键盘
InsertImage	插入图片
WPPMobileComponentToolbar	播放时的toolbar
WPPMobileThumbnailContainer	播放时的缩略图
WPPMobileTimeStamp	播放时时间显示
FloatMenuDownloadImage	浮动菜单栏>下载图片

PDF

- **PC端**

CommandBarId	位置
StartTab	工具栏>开始Tab
AnnoTab	工具栏>批注Tab
TabStartTab	工具栏>开始按钮
TabAnnoTab	工具栏>批注按钮
TaskPane	任务窗格

CommandBarId	位置
PlayModeMenu	播放模式菜单

- **Mobile端**

CommandBarId	位置
Meeting	底部>会议按钮
PDFMobilePageBar	页码

6.6.7. 定制元素

定制元素对象 `CommandBars.Controls` 是自定义元素的总开关，其中 `Controls` 为复数形式的控制列表。本文介绍如何新增定制元素、隐藏定制元素、新增下拉框定制元素等。

了解元素定制

如下以 `开始Tab` 为例介绍如何添加自定义按钮和自定义下拉组件。

- 效果演示
  - 添加前

  - 添加后

- `CommandBarId` 列表

CommandBarId	位置
StartTab	工具栏>开始Tab
InsertTab	工具栏>插入Tab
ReviewTab	工具栏>审阅Tab

CommandBarId	位置
PageTab	工具栏>页面Tab

更多信息，请参见可定制列表。

- 语法

```
表达式.CommandBars(CommandBarId).Controls
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //通过CommandBar的Controls对象，可获取到对应类型的内容集合，从而进行新增操作或者获取到具体内容进行操作。
  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //新增1个按钮定制元素
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';

  //新增1个下拉框定制元素
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
}
```

新增定制元素

通过 `Controls.Add()` 属性，您可以在对应的Tab中添加自定义按钮和自定义下拉组件。

- 效果演示
  - 添加前

  - 添加后

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必选	描述
Type	Enum	是	新增的定制元素类型。`Enum.MsoControlType` 取值范围如下： ○ 1或msoControlButton：可控按钮。 ○ 10或msoControlPopup：可控下拉框。

- 示例

```
async funcion example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //新增1个按钮定制元素
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';

  //新增1个下拉框定制元素
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
}
```

单个定制元素

单个定制元素，即具体到对应的自定义元素，您可以获取该元素，从而控制该元素的标题、单击该元素等等。

了解单个元素定制

如下以 `开始Tab` 为例介绍按钮新增和删除的操作。

- 效果演示
  - 新增

  - 删除

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Item(Index)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Index	Number	是	第Index个定制元素。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //新增1个按钮定制元素
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';

  //6000 ms后删除该定制元素
  setTimeout(() => {
    controlButton.Delete();
  }, 6000);
}
```

设置标题

通过 `Caption` 属性，您可以设置定制元素的标题。

- 效果显示

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).Caption
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置标题为“按钮”
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';

  //添加1个下拉框定制元素并设置标题为“下拉框”
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
}
```

删除定制元素

通过 `Delete()` 属性，您可以删除定制元素。

- 效果演示
  - 删除前

  - 删除后

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).Delete()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置标题为“按钮”
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';

  //添加1个下拉框定制元素并设置标题为“下拉框”
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';

  //6000 ms后删除这2个定制元素
  setTimeout(() => {
    controlButton.Delete();
    controlPopup.Delete();
  }, 6000);
}
```

禁用定制元素

通过 `Enabled` 属性，您可以禁用定制元素。

- 效果演示（置灰）

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).Enabled
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并禁用
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';
  controlButton.Enabled = false;

  //添加1个下拉框定制元素并禁用
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
  controlPopup.Enabled = false;
}
```

单击定制元素

通过 `Execute()` 属性，可以单击定制元素。配合监听定制元素单击事件，可以得知按钮是否被单击。

- 语法

  表达式.CommandBars(CommandBarId).Controls.Add(Type).Execute()

  表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置单击监听
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';
  controlButton.OnAction = () => console.log('按钮点击了');

  //添加1个下拉框定制元素并设置单击监听
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
  controlPopup.OnAction = () => console.log('下拉框点击了');

  //6000 ms后执行单击操作
  setTimeout( async () => {
    await controlButton.Execute();
    await controlPopup.Execute();
  }, 6000);
}
```

监听定制元素单击事件

设置监听定制元素单击事件后，若按钮被单击，可以通过控制台查看到对应的信息，从而制定相关的业务。

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).OnAction() = Function
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置单击监听
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';
  controlButton.OnAction = () => console.log('按钮点击了');

  //添加1个下拉框定制元素并设置单击监听
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
  controlPopup.OnAction = () => console.log('下拉框点击了');
}
```

定制元素图标

通过 `Picture` 属性，可以定制元素图标。如下以 `Base64` 为例，介绍如何设置对应的元素图标。

- 效果演示

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).Picture
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置图标
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';
  controlButton.Picture = '
IHZpZXdCb3g9IjAgMCAxNiAxNiIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIj48ZyBmaWxsPSIjM
0Q0NzU3IiBmaWxsLXJ1bGU9ImV2ZW5vZGQiPjxwYXRoIGQ9Ik04LjIxMyAxM0g2LjhsNi42MzYtNi42MzYtNC4yND
MtNC4yNDMtNy4wNyA3LjA3MUw1LjkyOCAxM0g0LjUxNUwxLjA2IDkuNTQ2YS41LjUgMCAwIDEgMC0uNzA3TDguODM
5IDEuMDZhLjUuNSAwIDAgMSAuNzA3IDBsNC45NSA0Ljk1YS41LjUgMCAwIDEgMCAuNzA3TDguMjEzIDEzeiIgZmls
bC1ydWxlPSJub256ZXJvIi8+PHBhdGggZD0iTTQuNTM2IDYuMzY0bDQuOTUgNC45NS0uNzA3LjcwNy00Ljk1LTQuO
TV6TTQuNTIxIDEzaDEwLjAzdjFINS40OTZ6Ii8+PC9nPjwvc3ZnPg==';

  //添加1个下拉框定制元素并设置图标
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
  controlPopup.Picture = '
HZpZXdCb3g9IjAgMCAxNiAxNiIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIj48ZyBmaWxsPSJub2
5lIiBmaWxsLXJ1bGU9ImV2ZW5vZGQiPjxwYXRoIGQ9Ik03LjUgMnYyLjVINGEuNS41IDAgMCAwLS41LjV2MmEuNS4
1IDAgMCAwIC41LjVoOWEuNS41IDAgMCAwIC41LS41VjVhLjUuNSAwIDAgMC0uNS0uNUg5LjVWMmEuNS41IDAgMCAw
LS41LS41SDhhLjUuNSAwIDAgMC0uNS41eiIgc3Ryb2tlPSIjM0Q0NzU3Ii8+PHBhdGggZmlsbD0iIzNENDc1NyIgZ
D0iTTEzIDdoMXY0aC0xeiIvPjxwYXRoIGQ9Ik0xMSAxM2EyIDIgMCAwIDAgMi0yVjguNzY0QTMgMyAwIDEgMSA4Lj
c2NCAxM0gxMXoiIGZpbGw9IiMzRDQ3NTciLz48cGF0aCBmaWxsPSIjM0Q0NzU3IiBkPSJNMSAxM2gxMHYxSDF6Ii8
+PHBhdGggZD0iTTEgMTNhMiAyIDAgMCAwIDItMlY4Ljc2NEEzIDMgMCAwIDEgMSAxNHYtMXoiIGZpbGw9IiMzRDQ3
NTciLz48cGF0aCBmaWxsPSIjM0Q0NzU3IiBkPSJNMyA3aDF2NEgzeiIvPjwvZz48L3N2Zz4=';
}
```

聚焦定制元素

通过 `SetFocus` 属性，可以将光标聚集到对应位置，即聚焦定制元素。

- 效果演示

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).SetFocus()
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并聚焦
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';
  await controlButton.SetFocus();

  //添加1个下拉框定制元素并聚焦
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
  await controlPopup.SetFocus();
}
```

隐藏定制元素

通过 `Visible` 属性，可以设置隐藏或显示元素。

- 效果演示
  - 显示

  - 隐藏

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).Visible
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';

  //添加1个下拉框定制元素
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';

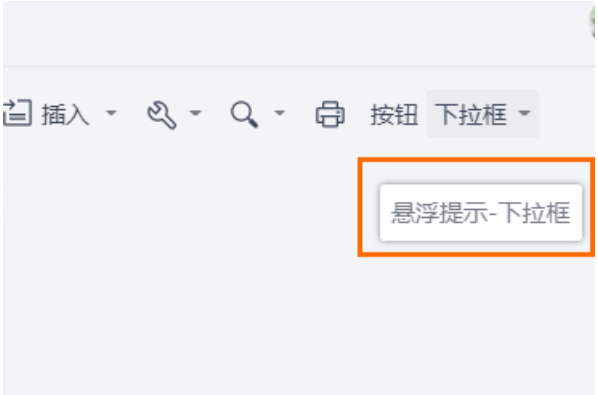
  //6000 ms后隐藏定制元素
  setTimeout(() => {
    controlButton.Visible = false;
    controlPopup.Visible = false;
  }, 6000);
}
```

定制元素悬浮提示

通过 `TooltipText` 属性，可以设置定制元素的悬浮提示。

- 效果演示
  - 按钮

  - 下拉框



- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).TooltipText
```

表达式：文档类型应用对象

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置悬浮提示
  const controlButton = await controls.Add(1);
  controlButton.Caption = '按钮';
  controlButton.TooltipText = '悬浮提示-按钮';

  //添加1个下拉框定制元素并设置悬浮提示
  const controlPopup = await controls.Add(10);
  controlPopup.Caption = '下拉框';
  controlPopup.TooltipText = '悬浮提示-下拉框';
}
```

批量定制可控元素属性

通过 `With()` 属性，可以批量设置定制元素的属性，具体各个属性的使用，可以参考单个属性的设置。

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(Type).With(Properties)
```

表达式：文档类型应用对象

- 参数

属性	数据类型	是否必填	描述
Caption	String	是	标题。
Enabled	Boolean	否	是否禁用。
Picture	String	否	图标。
Visible	Boolean	否	显示或隐藏。

属性	数据类型	是否必填	描述
TooltipText	String	否	悬浮提示。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个按钮定制元素并设置标题和禁用
  const controlButton = await controls.Add(1);
  controlButton.With({
    Caption: 按钮,
    Enabled: false,
  });

  //添加1个下拉框定制元素并设置标题和禁用
  const controlPopup = await controls.Add(10);
  controlPopup.With({
    Caption: '下拉框',
    Enabled: false,
  });
}
```

下拉框内定制元素

在 `CommandBars(CommandBarId).Controls.Add()` 中，如果 `Enum` 的值为10，可以添加一个下拉框定制元素。如果需要使用该下拉框，则需要进一步设置。例如下图的下拉框中新增了一个按钮**下拉按钮1**，关于如何对该按钮进行更多设置（例如标题、图表等），请参见单个定制元素。

新增和获取下拉框

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(10).Controls
```

其中表达式为文档类型应用对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个下拉框定制元素
  const control = await controls.Add(10);
  control.Caption = '下拉框';

  //获取下拉框
  const popupControls = await control.Controls;
}
```

新增下拉框定制元素

通过 `Add()` 属性，您可以为下拉框内添加新的定制元素。

- 语法

```
表达式.CommandBars(CommandBarId).Controls.Add(10).Controls.Add()
```

其中表达式为文档类型应用对象。

- 示例

```
async function example() {
  await instance.ready();

  const app = instance.Application;

  //获取定制元素对象开始Tab
  const controls = await app.CommandBars('StartTab').Controls;

  //添加1个下拉框定制元素
  const control = await controls.Add(10);
  control.Caption = '下拉框';

  //获取下拉框
  const popupControls = await control.Controls;

  //新增下拉框内定制元素
  const button = await popupControls.Add(1);
  button.Caption = '下拉按钮 1';
}
```

6.7. JS-SDK版本

本文介绍文档在线协作的版本发布信息。

> **注意**
>
> JS-SDK仅支持非模块化引用方式。引用时，请根据实际填写版本号。
>
> ```
> <script src="https://g.alicdn.com/IMM/office-js/1.1.15/aliyun-web-office-sdk.min.js"></
> script>
> ```

Release V1.1.15

发布时间：2021-10-20

更新日志：

- 新增isParentFullscreen配置项支持传入指定节点作为全屏对象。
- 优化iframe安全域校验因为www跳转导致的匹配错误问题。
- 修复“恢复历史版本”后，原有的wps.on事件监听不执行的问题。
- 修复Safari浏览器无法显示导出按钮的兼容性问题。
- 修复全屏事件在Safari浏览器中的兼容性问题。

Release V1.1.9

发布时间：2021-04-19

更新日志：

- 修复IE11下高级API调用错误问题。
- 加强JS SDK消息通讯安全。

Release V1.1.5

发布时间：2020-06-22

更新日志：

- 新增PC全屏方式控制配置，iframe内全屏和浏览器内全屏。
- 修复setter的callback问题。

Release V1.1.2

发布时间：2020-06-22

更新日志：

- 新增多个组件状态设置。
- 新增支持error事件，其中token过期的error暂不支持。
- 演示相关新增幻灯片播放状态以及幻灯片动画控制。
- PDF相关新增设置页面渲染模式。

Release V1.1.1

发布时间：2020-04-21

更新日志：

- 新增没有配置mount默认占满全屏功能。
- updateConfig()方法兼容处理，该函数即将废弃。
- 修复偶现API调用时序错乱问题。
- 修复单个接口调用catch()不生效问题。
- 修复重复调用config时文档类型没更新问题。
- 修复事件回调顺序问题。

7.图片管理

7.1. 概述

通过图片标准型项目，您可以管理图片类型数据，进行一站式图片的处理、分析、搜索、管理等操作。

注意事项

- 图片格式支持输入JPG、PNG、BMP格式的图片。除二维码检测功能外，其他功能还支持输入GIF格式的图片，GIF格式的图片均以首帧静态图作为检测依据。
- 当输入WEBP、TIFF、HEIC格式的图片时，智能媒体管理会自动尝试使用OSS提供的图片转换能力进行格式转换，然后进行图片AI处理。

  注意 此过程将可能产生一定转换费用。更多信息，请参见数据处理计费说明。

- 输入图片最大为100 MB。二维码检测功能输入图片最小为40 px*40 px。
- 对图片中的人脸数目没有限制，默认返回所有识别出来的人脸；检测时间和图片尺寸有关，图片越大耗时越长。
- 对于过大的图片，请求可能超时，请减小图片尺寸后再次尝试。

功能

名称	说明
内容识别	检测图片中的标签。更多信息，请参见内容识别。
二维码检测	检测图片中的二维码，并返回二维码的内容和位置。更多信息，请参见二维码识别。
图片裁剪建议	根据指定裁剪比例获取图片的裁剪建议。更多信息，请参见图片裁剪建议。
图片质量评分	检测图片的综合质量分数。更多信息，请参见图片质量评分。
人脸检测	检测图片中的人脸。更多信息，请参见人脸检测。
人脸搜索	从媒体集中搜索与指定图片或人脸ID最相似的前N张图片。更多信息，请参见人脸搜索。
人脸对比	比较两个人脸的相似度，可通过如下两种方式实现。更多信息，请参见人脸比对。 • 无需创建媒体集，直接传入两张图片的OSS URI，比较两张图片中最大的两个人脸的相似度。 • 创建媒体集并进行IndexImage操作将图片索引至媒体集中后，直接传入一个媒体集中的两个人脸ID即FaceId进行比较。
人脸聚类	将媒体集中存在相似人脸的多张图片进行分组。人脸聚类后，您可以根据人脸分组查询对应人员的所有图片信息。更多信息，请参见人脸聚类。
人体检测	检测图片中的人体位置信息。更多信息，请参见人体检测。

名称	说明
图片盲水印	为图片添加盲水印。更多信息，请参见图片盲水印。 盲水印添加后，用户不能直接看到图片中的水印信息，如果需要恢复图片中的水印信息，请使用解析图片盲水印功能。
标签搜索	创建媒体集并将图片索引到媒体集后，您可以根据时间、地点、标签、人物特征等搜索媒体集中的图片信息。更多信息，请参见标签搜索。

7.2. 图片分析

7.2.1. 内容识别

基于图片AI技术，识别图片中的场景、物体和事件等内容，实现图片的自动打标，可用于相册分类、图库分类检索等场景。

功能概述

内容识别可以识别图片中的场景、物体和事件等内容，实现图片的自动打标。目前支持的标签种类包含三十多个分类、数千个标签，如下图所示。

使用

您可以通过不同语言的SDK调用DetectImageTags接口进行内容识别。更多信息，请参见SDK参考。

7.2.2. 二维码识别

您可以使用二维码识别功能检测图片中的二维码以及二维码的位置和内容，通常用于二维码读取、图片审核等场景。

功能概述

二维码识别可以检测图片中的二维码以及二维码的位置和内容，其中位置包含左上角横坐标、左上角纵坐标、宽度和高度的值，如下图所示。当前支持照片、截图等多种图片来源。

基于获取的元数据，您可以在应用中实现二维码扫描和读取，以及对图片中的二维码进行屏蔽、打码等操作。

使用

您可以通过不同语言的SDK调用DetectImageQRCodes（推荐）或DetectQRCodes接口进行二维码识别。

7.2.3. 图片裁剪建议

您可以使用图片裁剪建议功能按照指定裁剪比例获取图片的裁剪建议，通常用于图片裁剪等场景。

功能概述

图片裁剪建议可以根据指定比例返回图片的裁剪建议框和裁剪方案的美学分数。如果指定了多个裁剪比例，系统会按照裁剪比例分别给出图片的裁剪建议。

使用方式

您可以通过不同语言的SDK调用GetImageCroppingSuggestions接口获取图片裁剪建议。

7.2.4. 图片质量评分

您可以使用图片质量评分功能检测图片的综合质量分数，通常用于封面图、低质量图片过滤等场景。

您可以通过不同语言的SDK调用GetImageQuality接口获取图片的综合质量分数。

- Java SDK
- Node.js SDK
- PHP SDK
- Python SDK
- Go SDK
- .NET SDK

7.3. 人脸人体

7.3.1. 人脸检测

基于图片AI技术，检测图片中的人脸以及人脸信息，可用于相册分类等场景。

功能概述

人脸检测可以检测图片中的人脸以及人脸信息，如果图片中有多张人脸，系统会检测多张人脸以及人脸信息。人脸信息包括人脸ID、年龄、性别、心情、吸引力、人脸质量、人脸属性等，其中人脸属性包括人脸位置、头部朝向、眼镜、胡子、面罩等。

您可以针对人脸信息进行搜索，例如搜索特定年龄、性别等类型的照片。

使用

您可以通过不同语言的SDK调用DetectImageFaces接口进行人脸检测。

7.3.2. 人脸搜索

创建媒体集并将图片索引到媒体集后，您可以使用人脸搜索功能从媒体集中搜索与指定图片或人脸ID最相似的前N张图片，可用于VIP用户的识别等场景。

您可以通过不同语言的SDK依次调用如下接口实现人脸搜索功能。

1. 调用PutProject接口创建项目。

   如果已创建项目，请忽略此步骤。您可以调用ListProjects接口列出指定地域下已创建的所有项目信息。

   > **说明** 您也可以通过智能媒体管理控制台创建项目。具体操作，请参见创建项目。

2. 调用CreateSet接口创建媒体集，并获取媒体集ID（即SetId）。

   如果已存在符合使用需求的媒体集，请忽略此步骤。您可以调用ListSets接口获取指定项目中已创建的所有媒体集信息。

3. 调用IndexImage接口将图片索引到媒体集中。

如果图片较多，可多次执行此操作。

4. 调用FindSimilarFaces接口从媒体集中搜索与指定图片或人脸ID最相似的前N张图片。

7.3.3. 人脸比对

使用人脸对比功能，您可以通过比较媒体集中的两个人脸ID或者两张图片的OSS URI实现两个人脸的相似度比较，可用于身份核查及验证等场景。

您可以通过不同语言的SDK依次调用如下接口实现人脸对比功能。

1. 调用PutProject接口创建项目。

   如果已创建项目，请忽略此步骤。您可以调用ListProjects接口列出指定地域下已创建的所有项目信息。

   > **说明** 您也可以通过智能媒体管理控制台创建项目。具体操作，请参见创建项目。

2. 根据实际业务选择相应对比方式。
   - 通过比较媒体集中的两个人脸ID实现。
     a. 调用CreateSet接口创建媒体集，并获取媒体集ID（即SetId）。

        如果已存在符合需求的媒体集，请忽略此步骤。您可以调用ListSets接口获取指定项目中已创建的所有媒体集信息。

     b. 调用IndexImage接口将图片索引到媒体集中。

        如果图片较多，可多次执行此操作。

     c. 调用GetImage或ListImages接口获取需要进行搜索的人脸ID。
   - 通过比较两张图片的OSS URI实现。

     将两张图片上传到OSS的Bucket中。具体操作，请参见上传文件。

3. 调用CompareImageFaces接口比较两个人脸的相似度。

7.3.4. 人脸聚类

使用人脸聚类功能，你可以将媒体集中存在相似人脸的多张图片进行分组，可用于网盘的人脸相册、家庭监控的陌生人检测、甚至新零售的顾客管理等场景。人脸聚类后，您可以根据人脸分组查询对应人员的所有图片信息。

您可以通过不同语言的SDK依次调用如下接口实现人脸聚类功能。

1. 调用PutProject接口创建项目。

   如果已创建项目，请忽略此步骤。您可以调用ListProjects接口列出指定地域下已创建的所有项目信息。

   > **说明** 您也可以通过智能媒体管理控制台创建项目。具体操作，请参见创建项目。

2. 调用CreateSet接口创建媒体集，并获取媒体集ID（即SetId）。

   如果已存在符合需求的媒体集，请忽略此步骤。您可以调用ListSets接口获取指定项目中已创建的所有媒体集信息。

3. 调用IndexImage接口将图片索引到媒体集中。

   如果图片较多，可多次执行此操作。

4. 调用CreateGroupFacesJob接口对媒体集中的人脸进行聚类操作。

   重复步骤3~步骤4可进行增量人脸聚类的处理。

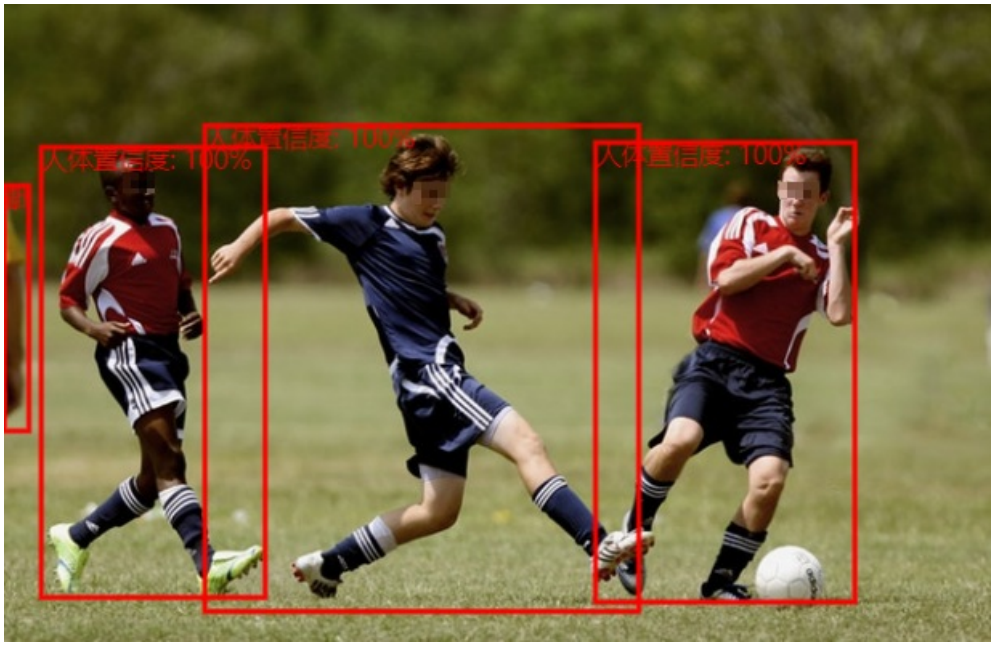
5. 根据人脸分组查询对应人员的所有图片信息。
   i. 调用ListFaceGroups接口获取媒体集中人脸分组的信息。
   ii. 调用FindImages接口查询人脸分组对应人员的所有图片信息。

7.3.5. 人体检测

您可以使用人体检测功能检测图片中所有的人体以及人体信息。

功能概述

人体检测可以检测图片中所有的人体以及人体位置框和置信度，如下图所示。



使用方式

您可以通过不同语言的SDK调用DetectImageBodies接口进行人体检测。

7.4. 图片盲水印

您可以使用添加图片盲水印功能为图片添加图片或文字类型的盲水印。盲水印添加后，用户不能直接看到图片中的水印信息，如果需要恢复图片中的水印信息，请使用解析图片盲水印功能，通常用于图片版权追溯等场景。

为图片添加图片类型盲水印的使用流程如下图所示。

为图片添加文字类型盲水印的使用流程如下图所示。

图片盲水印的具体使用步骤如下：

1. 上传待添加水印的图片到OSS Bucket。具体操作，请参见上传文件。
2. 根据添加的盲水印类型准备水印信息。
   - 图片类型：准备水印图片并上传水印图片到OSS Bucket。
   - 文字类型：准备水印文字内容。
3. 调用Put Project接口创建项目。

   如果已创建项目，请忽略此步骤。您可以调用List Projects接口列出指定地域下已创建的所有项目信息。

> **说明** 您也可以通过智能媒体管理控制台创建项目。具体操作，请参见创建项目。

4. 调用EncodeBlindWatermark接口为图片添加盲水印。
5. 调用DecodeBlindWatermark接口解析图片盲水印。

7.5. 标签搜索

创建媒体集并将图片索引到媒体集后，您可以根据时间、地点、标签、人物特征等搜索媒体集中的图片信息，可用于图片社区、智能相册等应用中搜索特定图片的场景。

您可以通过不同语言的SDK依次调用如下接口实现图片搜索功能。

1. 调用PutProject接口创建项目。

   如果已创建项目，请忽略此步骤。您可以调用ListProjects接口列出指定地域下已创建的所有项目信息。

   > **说明** 您也可以通过智能媒体管理控制台创建项目。具体操作，请参见创建项目。

2. 调用CreateSet接口创建媒体集，并获取媒体集ID（即SetId）。

   如果已存在符合需要的媒体集，请忽略此步骤。您可以调用ListSets接口获取指定项目中已创建的所有媒体集信息。

3. 调用IndexImage接口将图片索引到媒体集中。如果图片较多，可多次执行此操作。
4. 调用ListSetTags接口获取媒体集中Top 100的标签信息列表。
5. 调用FindImages接口根据指定条件，例如时间、地点、标签、人物特征等，搜索媒体集中的图片信息。

8.通过OSS使用智能媒体管理

为OSS的存储空间（Bucket）绑定智能媒体管理项目后，通过OSS控制台或SDK可以使用智能媒体管理的功能，例如文档预览、人脸识别等。本文介绍通过OSS使用智能媒体管理的前提条件、注意事项和使用流程。

前提条件

- 已开通智能媒体管理服务。具体操作，请参见开通产品。
- 已开通OSS服务。具体操作，请参见开通OSS服务。

注意事项

- 智能媒体管理的项目和OSS Bucket必须处于同一地域。
- 创建智能媒体管理项目及使用智能媒体管理功能会产生一定的费用。使用前请了解智能媒体管理的计费信息。更多信息，请参见计费说明。

快速使用

1. 配置智能媒体管理。

   创建项目。具体操作，请参见创建项目。

2. 配置OSS。

   i. 创建Bucket。具体操作，请参见创建存储空间。

   ii. 上传文件到Bucket。具体操作，请参见上传文件。

   iii. 为Bucket中的功能模块绑定对应类型的智能媒体管理项目。具体操作，请参见绑定智能媒体管理项目。

   > **注意** 当您不需要使用智能媒体管理功能时，请及时解绑Bucket中已绑定的智能媒体管理项目，避免产生额外的费用。

模块	说明
文档功能	绑定智能媒体管理的文档标准型项目。 绑定对应项目后，使用文档预览功能，可以在线预览PPT、XLS、DOC、PDF等格式的文件。具体操作，请参见文档预览。
人脸识别	绑定智能媒体管理的图片标准型项目。 绑定对应项目后，使用人脸识别功能，可以检测图片中的人脸矩形框和属性。具体操作，请参见人脸识别。
图片识别	绑定智能媒体管理的图片标准型项目。 绑定对应项目后，使用图片识别功能，可以检测图片标签和置信度。具体操作，请参见图片识别。

3. 通过OSS控制台或SDK使用智能媒体管理的功能。具体操作，请参见快速开始。

9.授权管理

9.1. 配置用户权限

阿里云权限管理机制包括访问控制和临时安全令牌，可以根据需求使用不同权限的RAM用户来访问表格存储，也支持为用户提供访问的临时授权。使用RAM和STS能极大地提高管理的灵活性和安全性。

背景信息

RAM和STS解决的一个核心问题是如何在不暴露主账号的AccessKey的情况下安全地授权其他人访问。因为主账号的AccessKey泄露会带来极大的安全风险，其他人可以随意操作该账号下所有的资源、盗取重要信息等。

- 访问控制（RAM）是阿里云提供的管理用户身份与资源访问权限的服务。RAM允许在一个阿里云账号下创建并管理多个身份，并允许给单个身份或一组身份分配不同的权限，从而实现不同用户拥有不同资源访问权限的目的。更多信息，请参见什么是访问控制。
- 阿里云临时安全令牌（Security Token Service，STS）是阿里云提供的一种临时访问权限管理服务。通过STS服务，您所授权的身份主体（RAM用户或RAM角色）可以获取一个自定义时效和访问权限的临时访问令牌。更多信息，请参见什么是STS。

RAM提供了一种长期有效的权限控制机制，通过分出不同权限的RAM用户，将不同的权限分给不同的用户，即使RAM用户的AccessKey泄露也不会造成全局的信息泄露。RAM用户一般情况也是长期有效的。因此RAM用户的AccessKey也不能泄露。

相对于RAM提供的长效控制机制，STS提供的是一种临时访问授权，通过STS可以获取临时的AccessKey和Token，这些信息可以直接发给临时用户用来访问智能媒体管理。一般来说，从STS获取的权限会受到更加严格的限制，并且拥有时间限制，因此即使这些信息泄露，对于系统的影响也很小。

配置RAM用户权限

为了数据安全和更好的权限控制，建议使用RAM用户访问智能媒体管理。

1. 创建RAM用户。具体操作，请参见创建RAM用户。
2. 为RAM用户授权，根据实际配置RAM用户的权限。具体操作，请参见为RAM用户授权。
   - 如果需要管理智能媒体管理，例如创建项目等，请授予RAM用户AliyunIMMFullAccess权限。
   - 如果只读访问智能媒体管理，例如查看项目等，请授予RAM用户AliyunIMMReadOnlyAccess权限。

   > **说明** 如果需要更精细的权限控制，您可以创建自定义权限策略，进行策略权限的配置。更多信息，请参见基于RAM Policy的权限控制。

3. 为RAM用户设置多因素认证。具体操作，请参见为RAM用户启用多因素认证。

配置临时用户权限

1. 创建临时角色及授权。
   i. 创建可信实体为阿里云账号的RAM角色。具体操作，请参见创建可信实体为阿里云账号的RAM角色。

ii. 创建自定义策略。具体操作，请参见创建自定义权限策略。

> 说明 如果需要更精细的权限控制，您可以自定义策略的权限。更多信息，请参见基于RAM Policy的权限控制。

iii. 为临时角色授权。具体操作，请参见为RAM角色授权。

2. 临时授权访问。

i. 创建自定义策略。具体操作，请参见创建自定义权限策略。

> 说明 如果需要更精细的权限控制，您可以自定义策略的权限。更多信息，请参见基于RAM Policy的权限控制。

ii. 授权RAM用户临时角色。具体操作，请参见为RAM用户授权。

3. 从STS获取的临时访问凭证。具体操作，请参见AssumeRole。
4. 使用临时授权读写数据。

   您可以使用临时授权调用不同语言的SDK访问智能媒体管理。Java SDK请参考以下方式创建IAcsClient对象，传入从STS获取的AccessKeyId、AccessKeySecret和SecurityToken等参数。

```
DefaultProfile profile = DefaultProfile.getProfile("cn-shanghai", stsAccessKeyId, stsAc
cessKeySecret, stsToken);
IAcsClient client = new DefaultAcsClient(profile);
```

9.2. 基于RAM Policy的权限控制

RAM是阿里云提供的资源访问控制服务，RAM Policy是基于用户的授权策略。通过设置RAM Policy，您可以集中管理您的用户（例如员工、系统或应用程序），以及控制用户可以访问您名下资源的权限，例如限制您的用户只拥有对某一个Project的读权限。

自定义RAM Policy

通过自定义RAM Policy实现用户权限控制，主要包括如下配置：

- `Effect`：可选值为`Allow`或者`Deny`，分别表示允许或者拒绝用户操作。
- `Action`：智能媒体管理对外提供的API操作，格式为`imm:<action>`，例如`imm:CreateOfficeConversionTask`。Action列表的更多信息，请参见API概览。
- `Resource`：智能媒体管理目前只有Project一种资源，格式为`acs:imm:<region-id>:<uid>:projects/<project>`，例如`acs:imm:cn-shanghai:150910xxxxxxxxxx:projects/imm-test-doc-proj`。对于Project相关的操作例如`PutProject`、`ListProject`等API，如果无对应操作资源时，需要设置`Resource`为`*`。
  - `region-id`：地域ID，例如`cn-shanghai`、`cn-beijing`。
  - `uid`：账号ID，可以在管理控制台的账户管理中查看。

- `project`：项目创建时指定，可以在智能媒体控制台中查看。

更多关于权限策略的信息请参考创建自定义策略。

常见示例

- 完全授权

  RAM Policy将允许用户访问使用智能媒体管理的所有功能。

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": "imm:*",
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

- 授权通配

  RAM Policy支持通配符 `*` ，可以实现批量授权，如下示例将允许用户调用在所有地域以 `imm-test-doc` 开头的Project下的可读操作。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["imm:List*", "imm:Get*"],
            "Resource": "acs:imm:*:150910xxxxxxxxxx:projects/imm-test-doc*"
        }
    ],
    "Version": "1"
}
```

- 授权特定Project的特定操作

  如下示例表示允许用户访问 `ListProjects` 接口，允许用户访问 `CreateOfficeConversionTask` 接口，并且仅能操作华东2（上海）地域的imm-test-doc-proj项目；允许用户访问 `CreateFigureClusteringTask` 和 `CreateFigureClustersMergingTask` 接口，并且仅能操作华东2（上海）地域的imm-test-media-proj项目。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["imm:ListProjects"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["imm:CreateOfficeConversionTask"],
            "Resource": "acs:imm:cn-shanghai:150910xxxxxxxxxx:projects/imm-test-doc-proj"
        },
        {
            "Effect": "Allow",
            "Action": ["imm:CreateFigureClusteringTask", "CreateFigureClustersMergingTask
"],
            "Resource": "acs:imm:cn-shanghai:150910xxxxxxxxxx:projects/imm-test-media-pro
j"
        }
    ],
    "Version": "1"
}
```

10.用户限额（Quota）

通过本文您可以了解智能媒体管理的默认限额。

默认限额配置请参见下表。

Quota项	默认值
按次计费Project文档转换最大文档大小	200 MB
按QPS计费Project文档转换最大文档大小	
单用户Project最大数量	20
单用户通用API调用次数限制	10 req/s
单用户GetOfficeConversionTask调用限制	30 req/s
单用户CreateOfficeConversionTask调用限制	20 req/s
单媒体集内图片总数量	50000
识别出的人脸总数量	200000
视频总数量	5000
视频秒数总长度（秒）	10000000

注意 在接口描述中如果出现参数可选值、可用规格内容与官网中提供的资源规格限制不一致时，均以官网中提供的值为准。