阿里云
DataWorks

应用开发

文档版本：20191113

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. **阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	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。
	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。
	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。	执行`cd /d C:/window`命令，进入Windows系统文件夹。
*## *	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 App Studio

1.1 App Studio概述

App Studio为您提供丰富的前端组件，通过自由拖拽即可快速搭建前端应用。本文将为您介绍App Studio的基本概念和产品优势。

App Studio是一款数据产品的开发工具，您无需下载安装本地IDE或配置维护环境变量，只需一个浏览器即可编写、运行和调试应用程序，体验和本地IDE一样的编程效果，并且可以在线发布应用。

产品优势

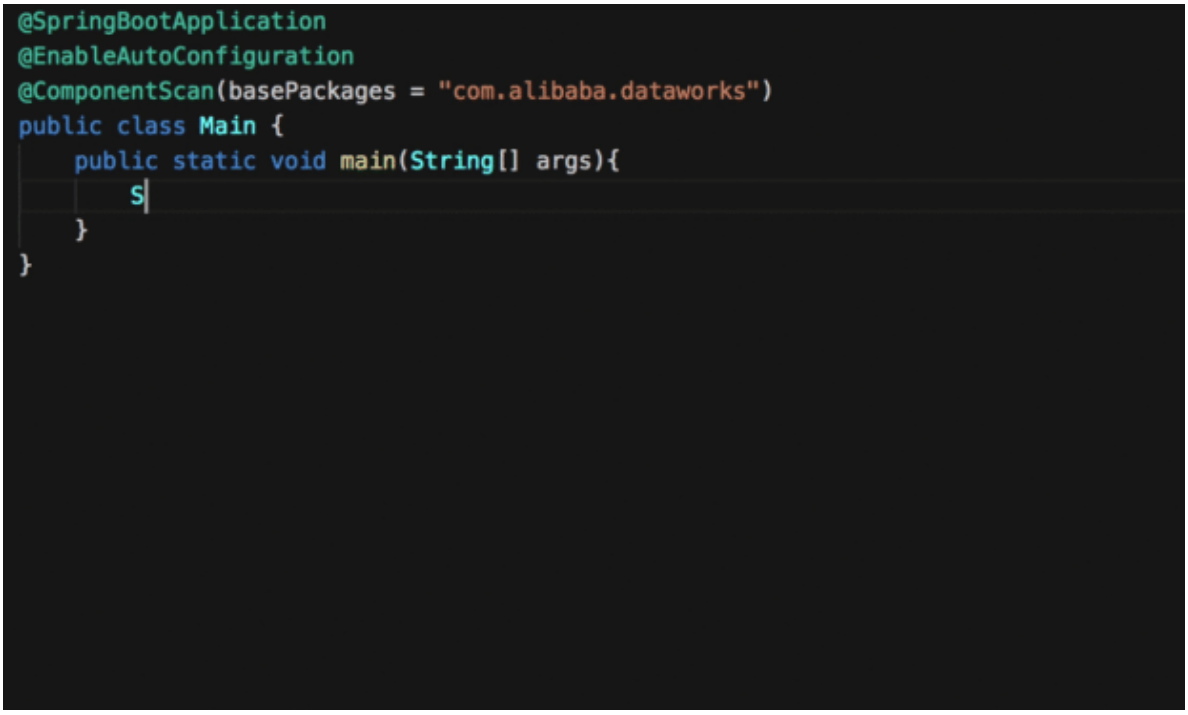
App Studio的核心优势如下：

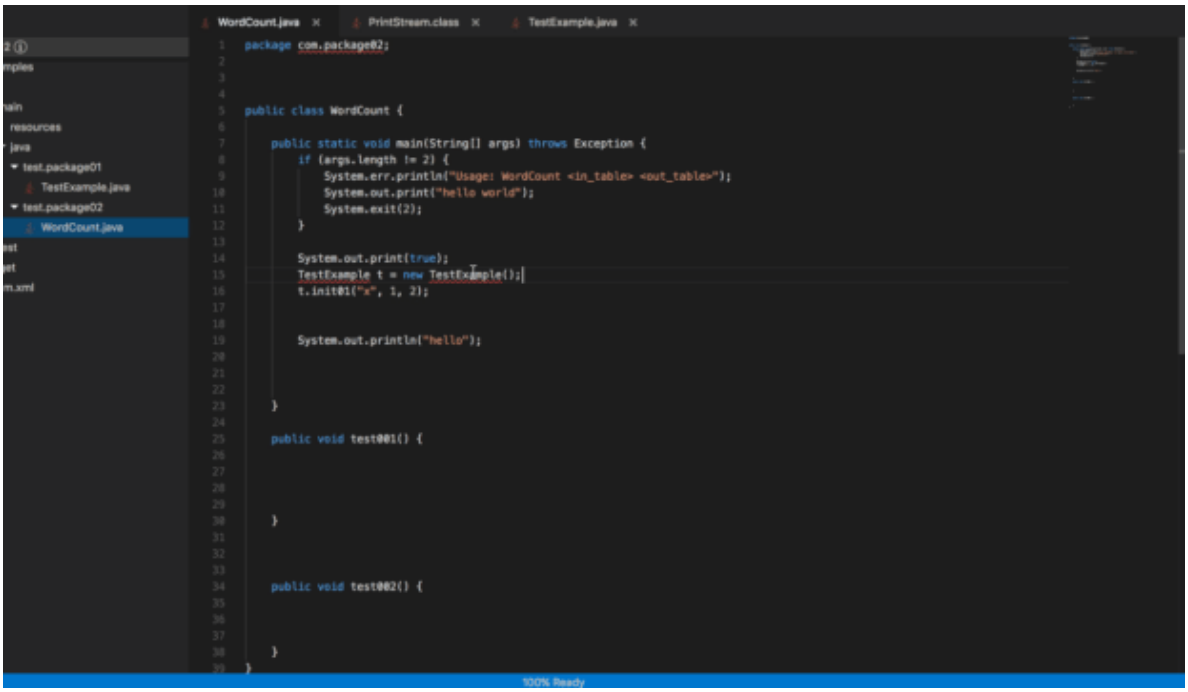
- **随时随地开发**

  您无需下载安装本地IDE和配置维护环境变量，只需要一个浏览器，即可在办公室、家或任何可以连接网络的地方，进行您的数据开发工作。

- **功能完备的编辑器**

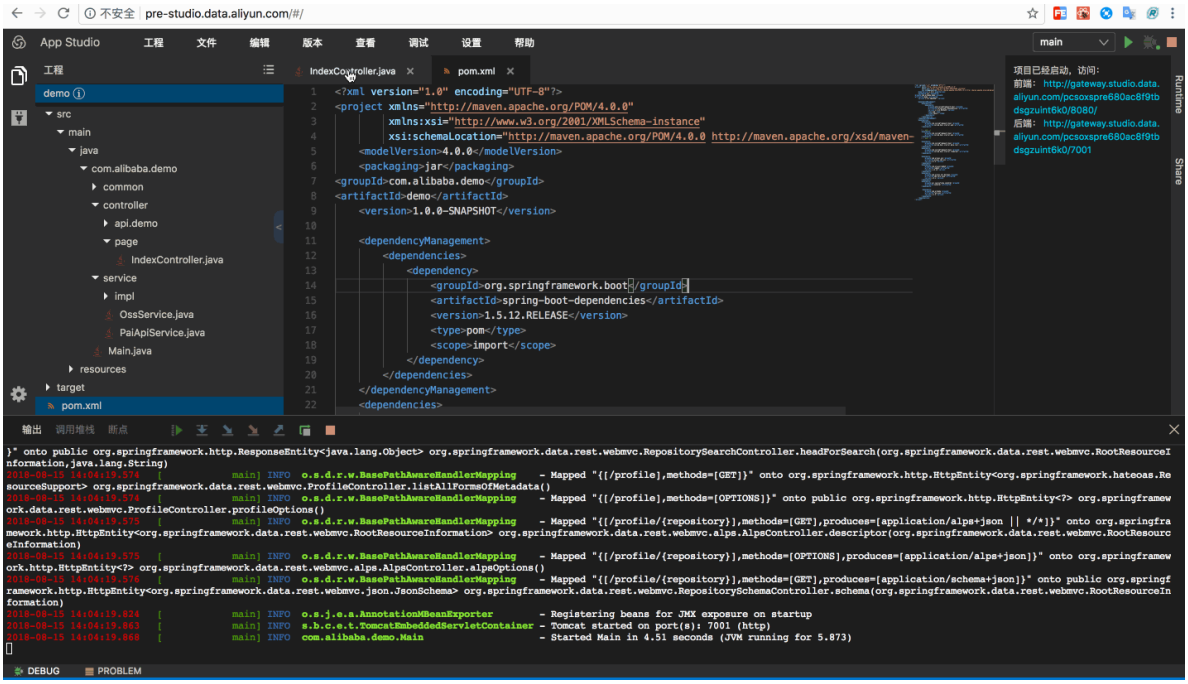
  App Studio提供一个基于浏览器的编辑器，您可以使用它轻松地编写、运行和调试项目。当您输入代码时，App Studio会提供智能提示、补全代码和修复建议等功能。您还可以查找方法的引用和定义，自动生成代码。





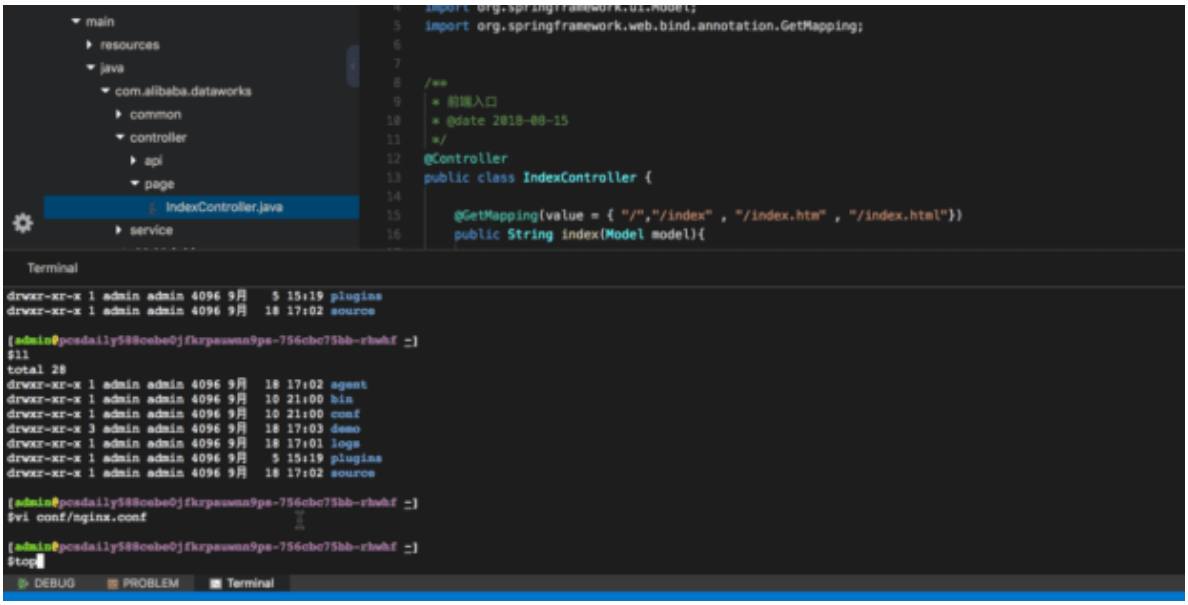
- **在线调试功能**

  在线调试包括本地IDE所有的断点类型和断点操作，支持线程切换、过滤，支持变量值查看、监视，支持远程调试和热部署。



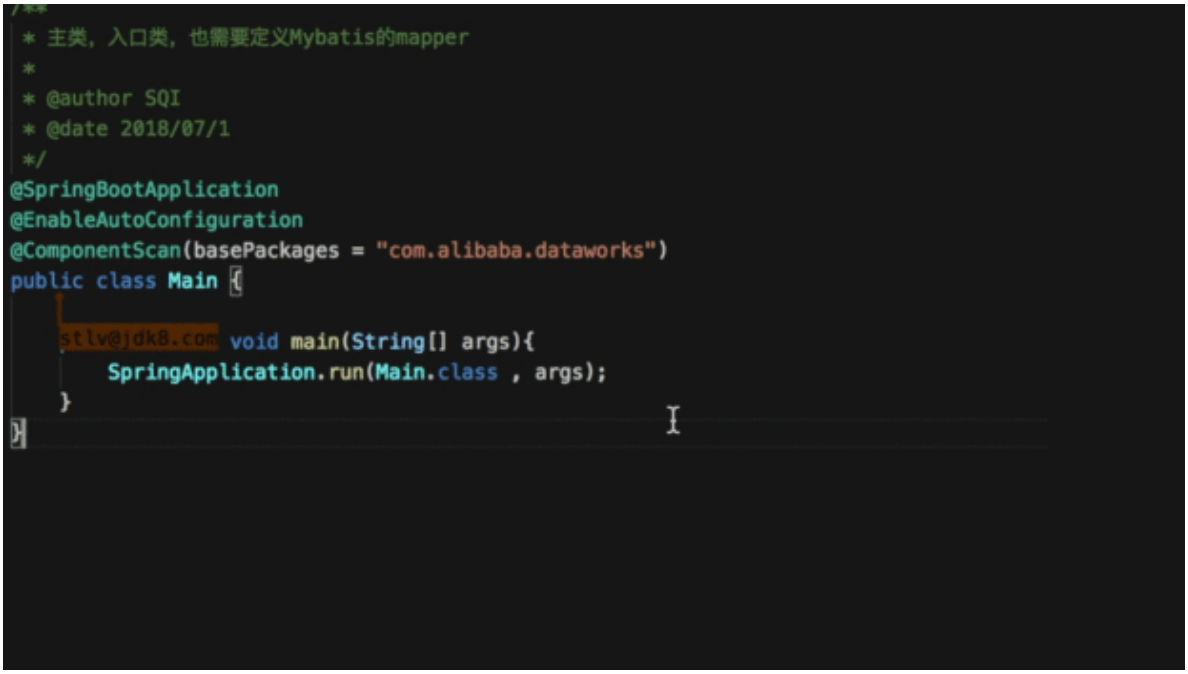
- **多功能终端**

  开发者可以直接进入运行环境，目前的运行环境基于CentOS作为基础镜像来构建。终端可以支持任意的bash命令，包括VIM等具有交互功能的命令。



- 协同编辑

  您和您的团队成员可以借助App Studio共享开发环境，进行团队协作编程。目前支持8人同时在线编辑同一个工程的同一个文件，提高工作效率。后续协同编辑组建还会支持聊天、弹幕、代码批注、视频等功能，让团队合作更加轻松。



- 插件体系

  App Studio支持业务、工具和语言3种插件：

  - 您可以根据业务在App Studio上定制任意的菜单栏，在界面增加业务入口。
  - 您可以定制专属于您业务的项目管理过程、工程类型和模板。
  - 您可以开发通用工具，例如GIT功能增强、代码规则扫描、快捷键、编辑功能增强、代码片段等集成到App Studio中。
  - 您可以通过语言插件扩展App Studio支持的语言，满足自身需求的同时帮助App Studio建设服务更多的语言用户。

- 可视化搭建

  App Studio提供丰富的组件，并且深度打通数据服务和数据开发。您能且仅能在App Studio中调用DataWorks的OpenAPI，并且通过可视化拖拽、配置的方式快速搭建前端应用，真正实现零代码开发Web应用。

- 丰富灵活的项目管理

  App Studio提供了丰富多样的模板工程，您可以基于模板工程进行再次开发，节省人力提高效率。您也可以将您自己的工程保存为模板，供您自己后续开发使用，或分享给其他人使用。

1.2 App Studio版本历史

本文将为您及时同步App Studio的版本更新。

App Studio 1.0

发布日期：2019年4月3日

发布内容：在Function Studio的基础上实现一个能做应用发布的IDE，核心功能如下所示：

- **LSP语言服务**

  支持语法高亮、智能提示、智能补全、智能诊断、查找定义、查找引用等本地编辑体验。
- **支持Debug功能**

  具有本地IDE所有的断点类型和断点操作，支持线程切换、过滤，支持变量值查看、监视，支持远程调试、热部署和多功能终端。
- **支持基于接口定义的前后端开发**

  前端可视化的组件可以通过配置后端接口进行前端联动。
- **前端可视化搭建**

  您可以通过拖拽组件搭建前端应用，具有很大的灵活性，支持没有前端经验的用户开发前端应用。支持前端模板管理，支持可视化模式和代码模式互转，可兼顾开发者更高的开发需求。
- **具备代码版本管理能力。**
- **可以在线部署和实时预览应用。**
- **具备协同编辑功能**

  支持8人同时在线编辑同一个工程同一个文件。
- **支持用户自定义工程模板，提供强大的工程管理能力。**
- **具有插件开发和集成能力**

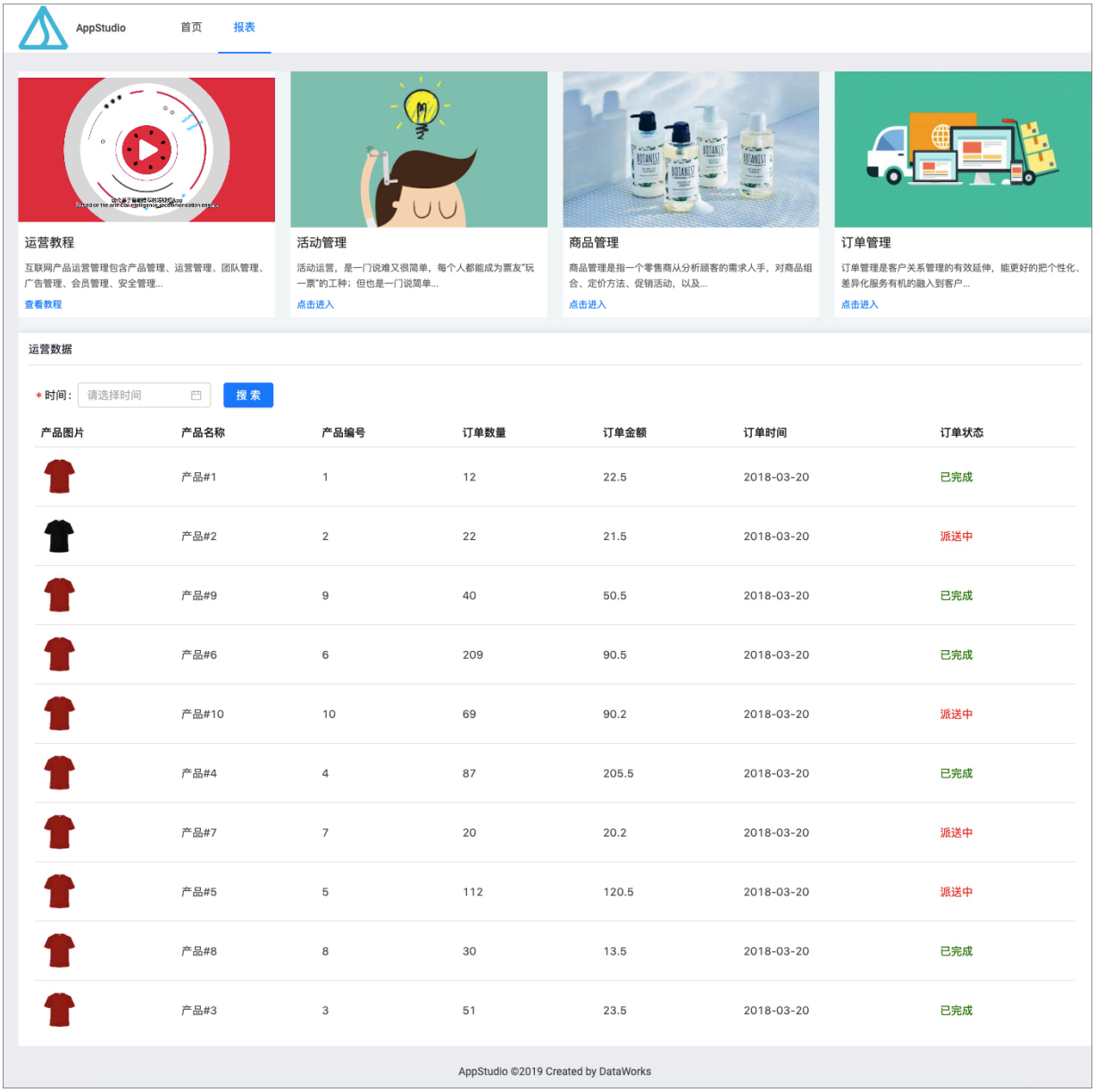
  支持用户开发插件定制专属于业务的IDE（计划在App Studio1.1版本和插件市场一起发布）。
- **支持Java、JS、CSS、HTML和Python多种语言。**
- **支持UT自动生成和运行。**
- **可设置项目为可分享状态，并通过链接分享给他人（计划在App Studio1.1版本和插件市场一起发布）。**
- **支持开发完成的应用在线发布（计划在App Studio1.2版本发布）。**

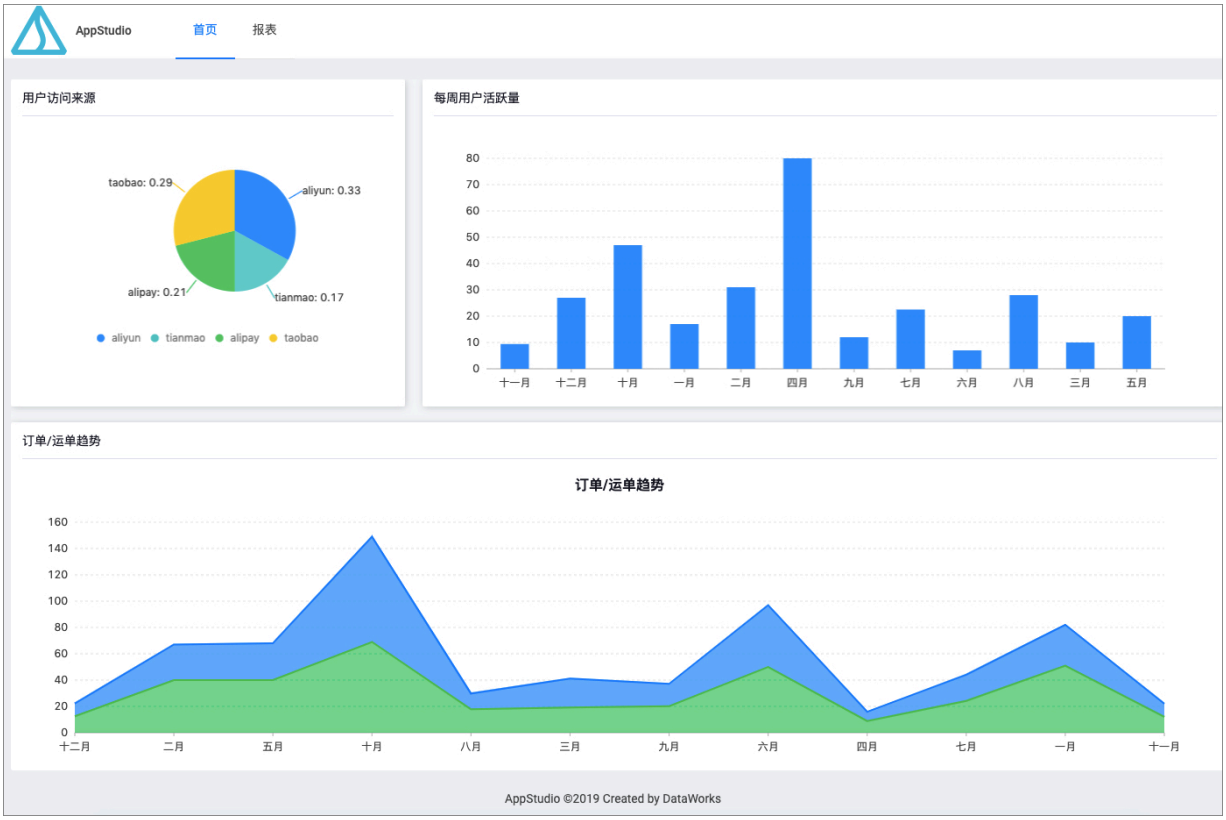
1.3 入门教程

通常，工程师搭建一个数据门户需要开发数据、搭建后端服务和开发前端页面三个环节。本文将为您介绍App Studio的基本功能及如何使用App Studio。

通常，数据工程师在DataWorks进行离线或流式数据开发。随着DataWorks的操作越来越简单，算法工程师、BI分析师、运营、熟悉SQL的产品经理等诸多角色，也逐渐可以在DataWorks进行数据开发。

针对不同种类的用户，App Studio可以助您快速搭建看数据的网页、查数据的App。



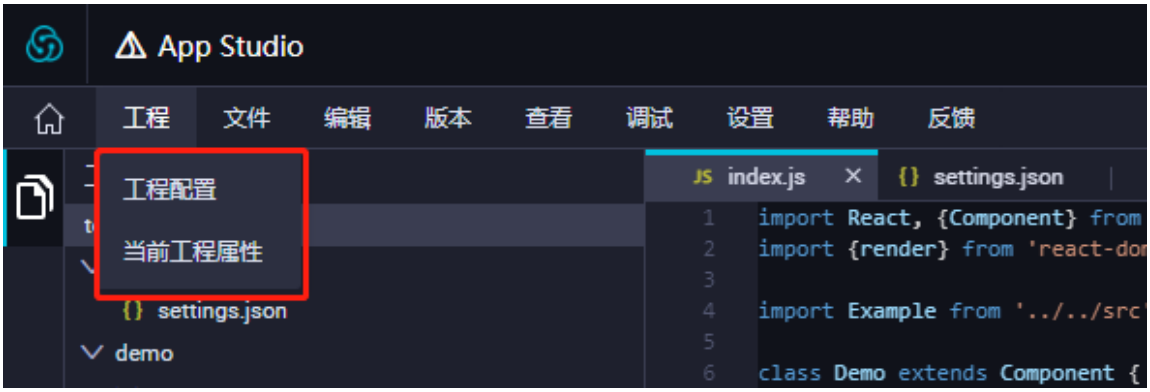

AppStudio
首页
报表
用户访问来源
taobao: 0.29
aliyun: 0.33
alipay: 0.21
tianmao: 0.17
aliyun tianmao alipay taobao
每周用户活跃量
80
70
60
50
40
30
20
10
0
十一月 十二月 十月 一月 二月 四月 九月 七月 六月 八月 三月 五月
订单/运单趋势
订单/运单趋势
160
140
120
100
80
60
40
20
0
十二月 二月 五月 十月 八月 三月 九月 六月 四月 七月 一月 十一月
AppStudio ©2019 Created by DataWorks

了解App Studio

- **菜单栏**

  - **工程**

    您可以通过工程菜单中的子菜单进行工程配置和查看当前工程属性。当前工程属性中包括工程ID、工程名、工程类型、创建时间和UUID等工程相关信息。



  - **文件**

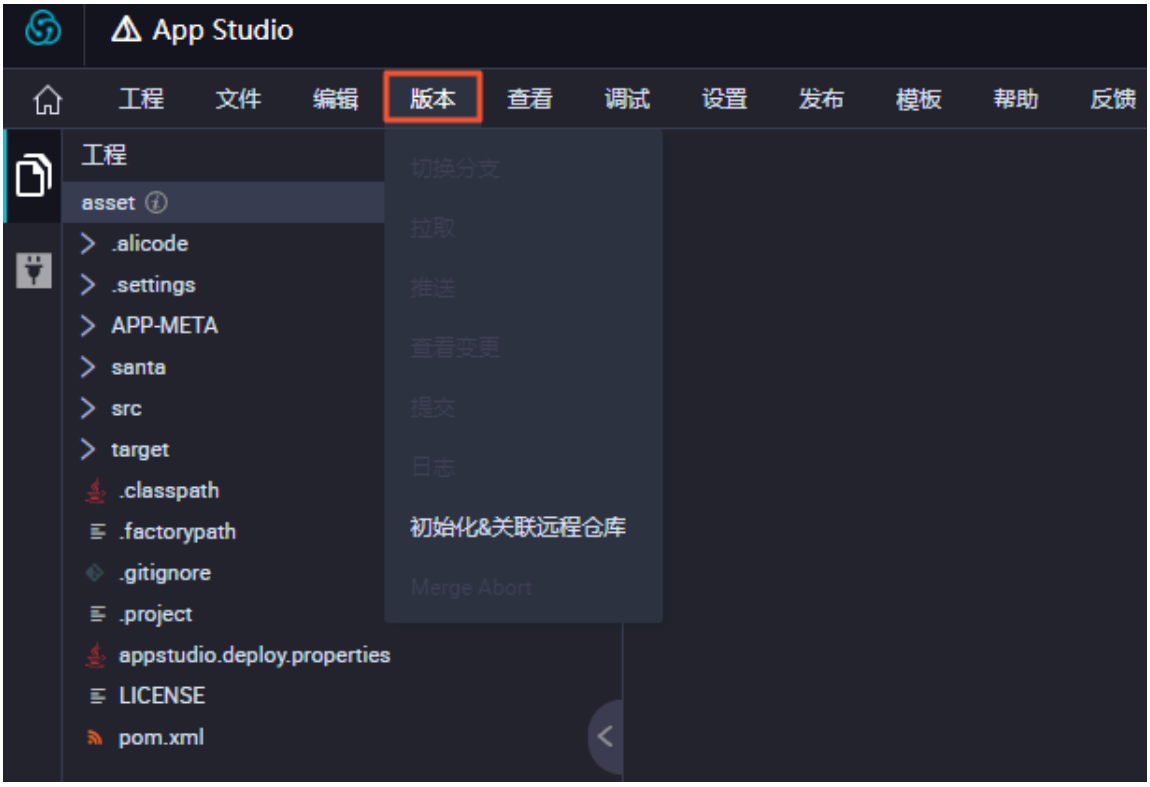
    您可以通过文件菜单中的子菜单新建文件、打开最近的文件。

  - **编辑**

    您可以通过编辑菜单栏进行常用的编辑操作，全文搜索是对工程内所有代码内容进行搜索，并可打开相关的文件。全文搜索的详情请参见全文内容搜索。

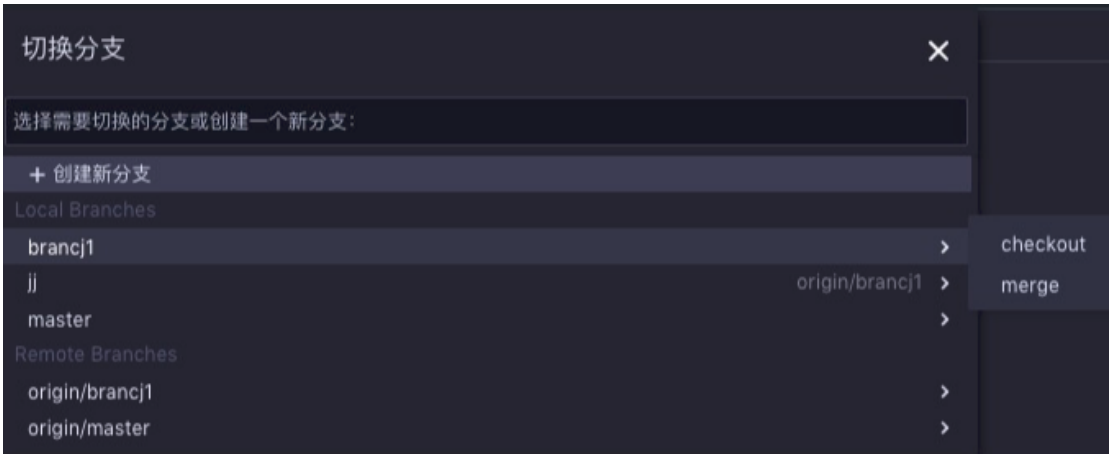
- **版本**

  您可以进行切换分支、拉取、推送、查看变更、提交、日志、初始化&关联远程仓库和Merge Abort等操作。

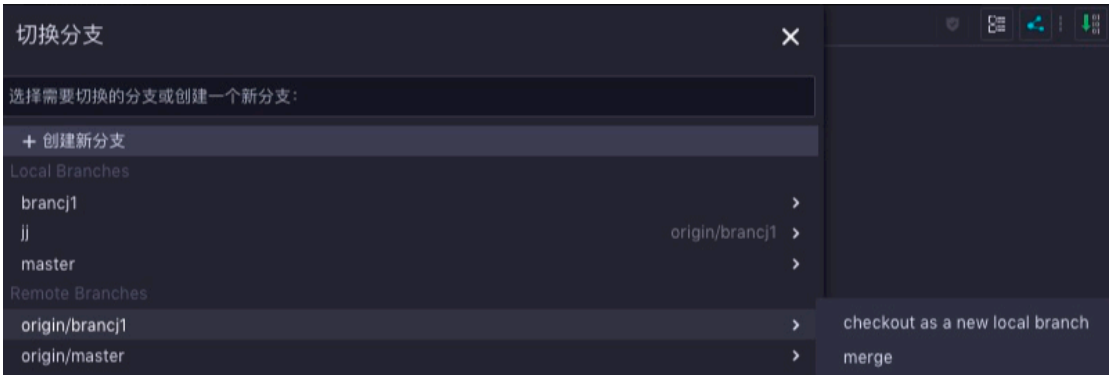


■ 切换分支

您可以通过+创建新分支创建本地新分支，然后推送到远程仓库。您可以选择一个本地分支，单击右边弹出框中的checkout。您也可以通过merge，将选中的分支合并到当前分支。



您可以选择一个远程分支，单击右边弹出框中的check out as a new local branch，将该远程分支checkout到本地并重新命名。您也可以通过merge，将选中的分支合并到当前分支。



■ **拉取**

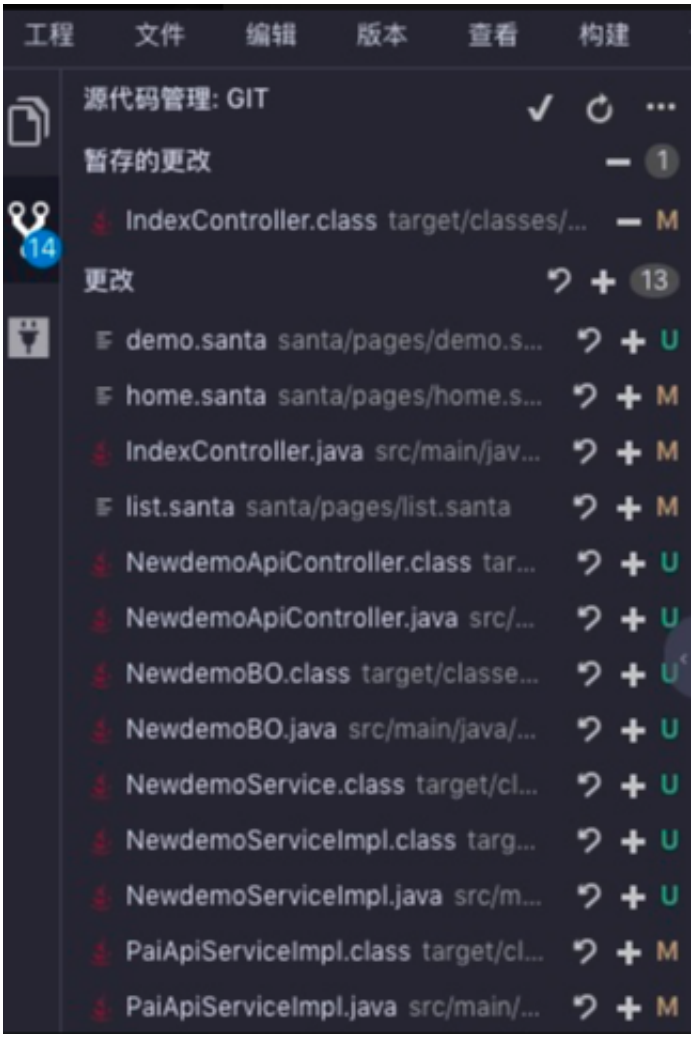
可以将远程分支的代码拉取到本地分支。

■ **推送**

可以将本地分支的变更暂存后推送到远程分支。

■ **查看变更**

单击查看变更后，右侧导航栏会弹出本地变更文件列表。



标识	说明
	代表变更文件的个数。
	代表新增的文件。
	代表变更的文件。
	单击后可以撤销更改。
	单击后可以暂存更改。
	单击后可以撤销暂存。
	单击后可以提交或提交并推送暂存的代码。

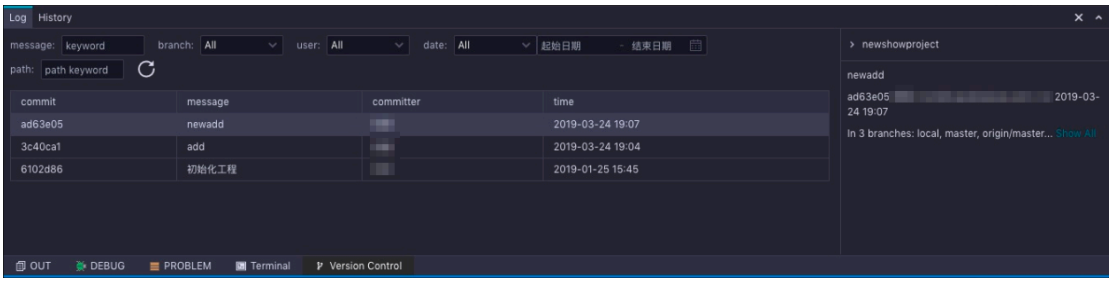
标识	说明
•••	更多中包括推送和拉取操作。

■ 提交

可以将本地分支的变更提交以暂存，需要输入commit信息。

■ 日志

可以查看分支的所有提交记录，并可以筛选。



■ 初始化&关联远程仓库

新建的工程可以关联远程仓库，从而进行版本管理。

- 查看

您可以通过切换全屏将IDE设置成全屏，然后通过Esc键退出全屏。您可以通过切换侧边栏和切换状态栏收起侧边和状态栏。

- 调试

■ 如果您建的是前端工程，调试选项如下所示。

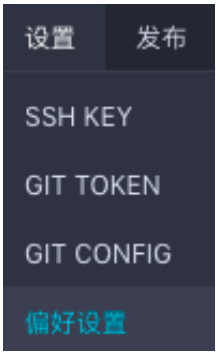
您可以配置运行参数、添加自定义镜像。

■ 如果您建的是后端工程，调试选项如下所示。

App Studio支持Java Debug，在后端工程的调试中，除配置和自定义镜像操作外，还有很多调试相关的操作。同时会有全量构建、增量构建、编译的操作入口。

- 设置

您在开始使用App Studio前，需要配置SSH KEY和GIT CONFIG。您也可以通过偏好设置，设置自己偏好的属性，目前仅支持字体大小，后续会支持颜色、样式、主题、快捷键等。



- **帮助**

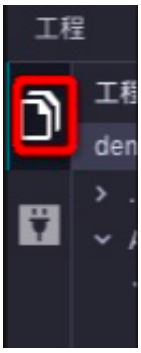
  您可以在帮助中查看产品使用文档、查看快捷键、查看版本历史和清空本地缓存。

- **反馈**

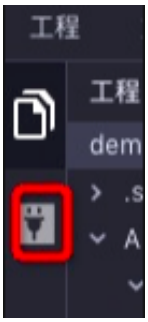
  您可以通过反馈提交问题和需求。

- 左边栏
  - 入口

    单击下图中的图标，即可展开工程区。

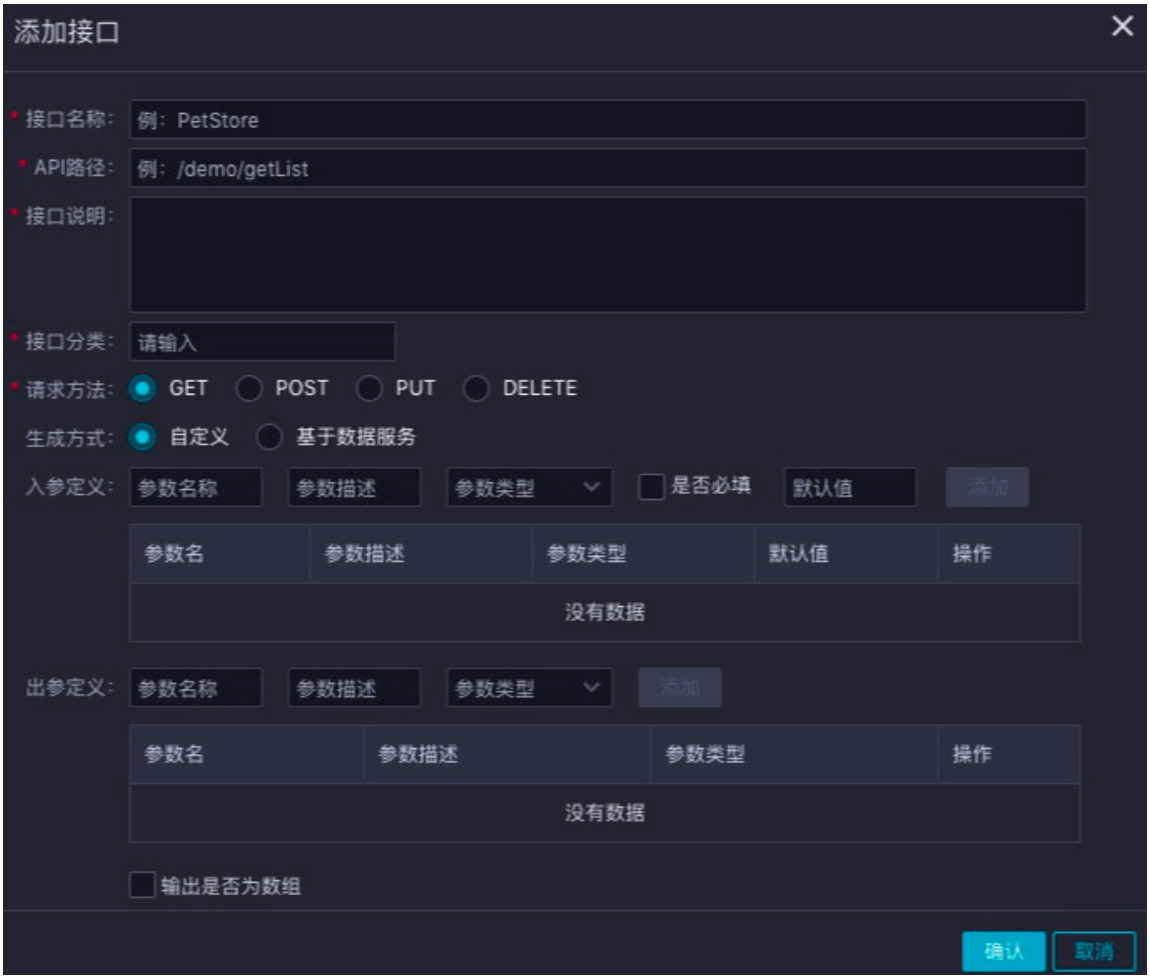
    单击下图中的图标，即可展开接口定义区。

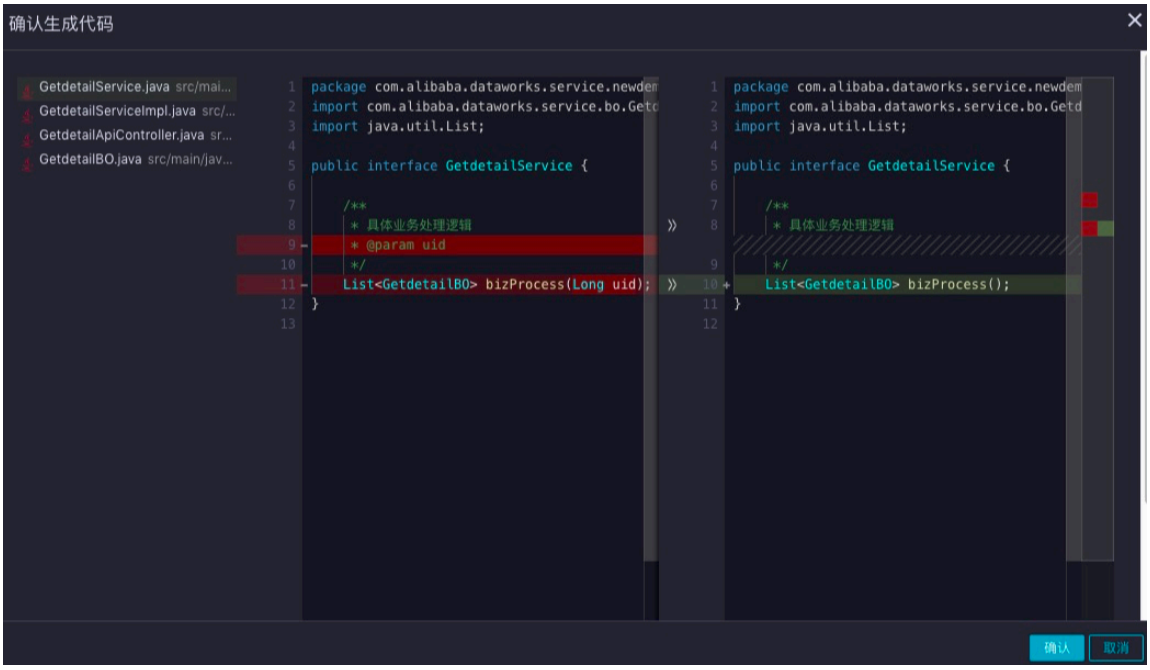
    

  - 接口定义区

    您可以添加接口并自动生成接口类代码，还可通过箭头，将左边的新代码同步到右边的本地代码中。

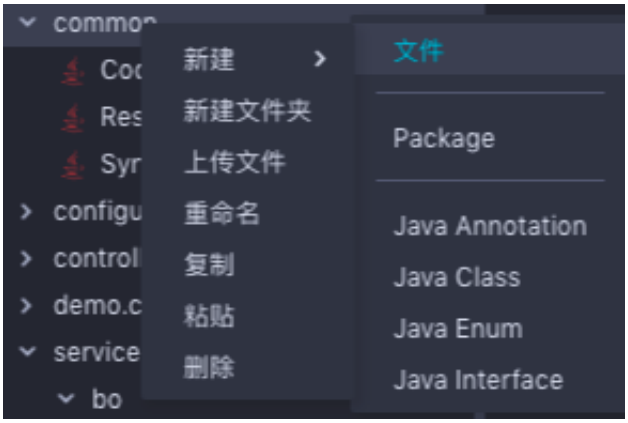




- **工程区**

■ 文件夹操作

如果您创建的是后端工程，文件模板新建后，会帮您自动生成一些框架代码。

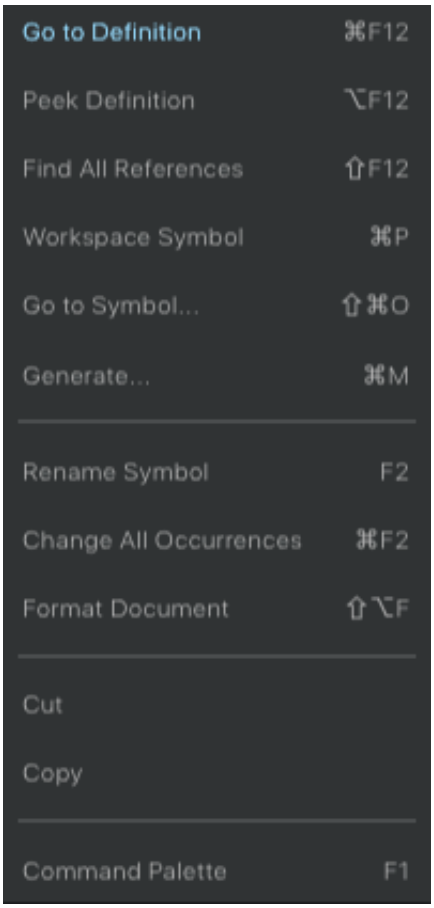


■ **文件操作**

如果您创建的是前端工程，则新建操作只有文件一个选项。

您可以重命名、复制和删除文件，也可以查看文件的GIT提交历史并进行版本对比。

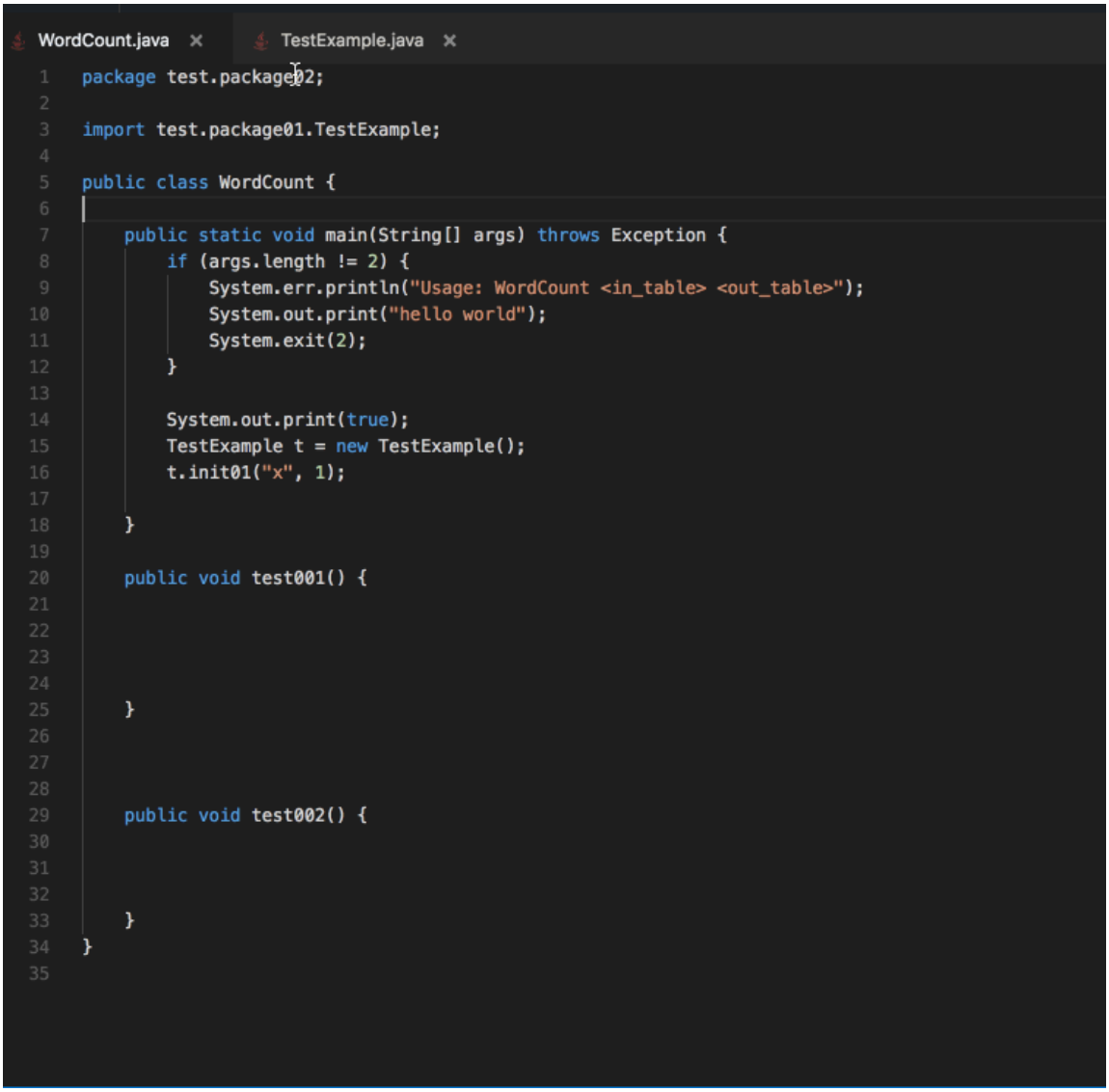
- 编辑区
  - 右键操作



操作	说明
Go to Definition	单击后跳转至定义。
Peek Definition	单击后可以预览定义。
Find All References	单击后可以查找所有引用。
Workspace Symbol	单击后可以在项目中查找符号。
Go to Symbol...	单击后可以跳转至符号。
Generate...	单击后可以生成代码。
Rename Symbol	单击后可以重命名符号。
Change All Occurrences	单击后可以修改当前文件中的所有该符号名字。
Format Document	单击后可以格式化文件。
Cut	剪切。
Copy	复制。

操作	说明
Command Palette	单击后可以进入命令面板。

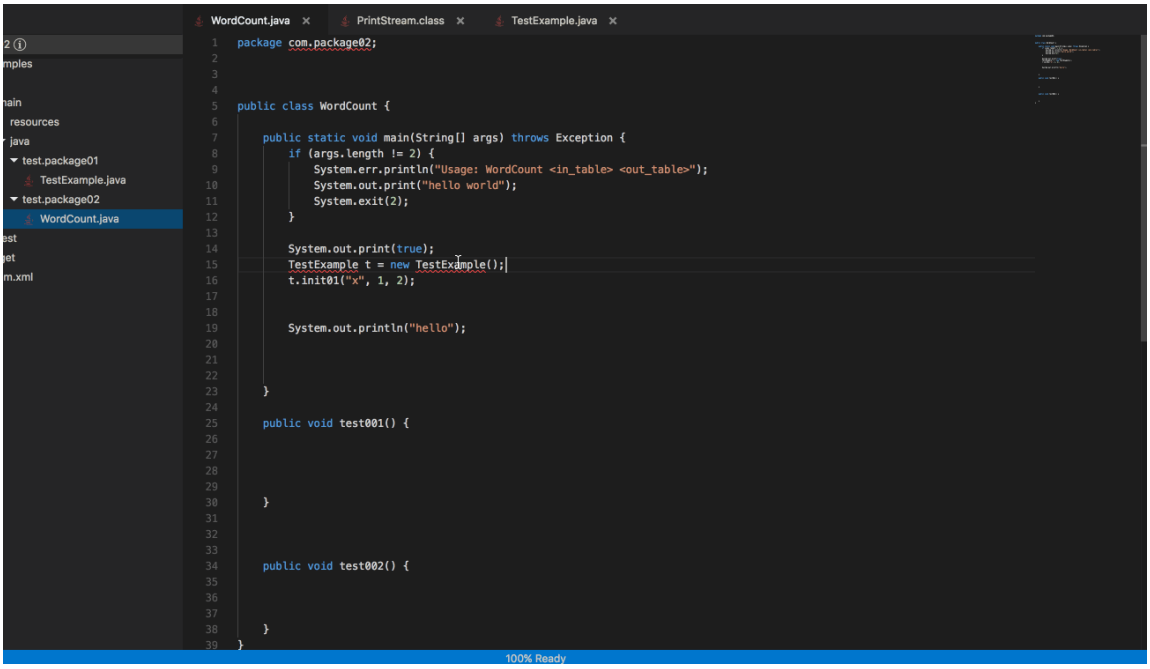
- 智能提示



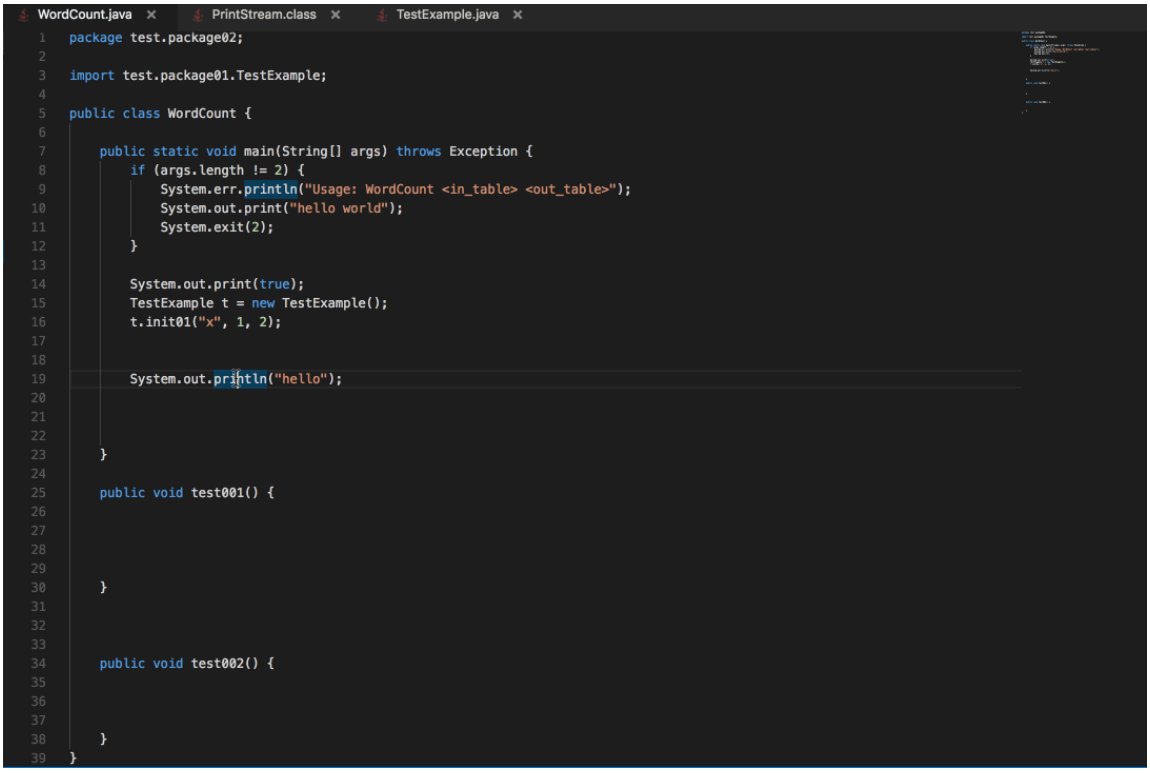
- 智能补全

```java
@SpringBootApplication
@EnableAutoConfiguration
@ComponentScan(basePackages = "com.alibaba.dataworks")
public class Main {
    public static void main(String[] args){
        S
    }
}
```

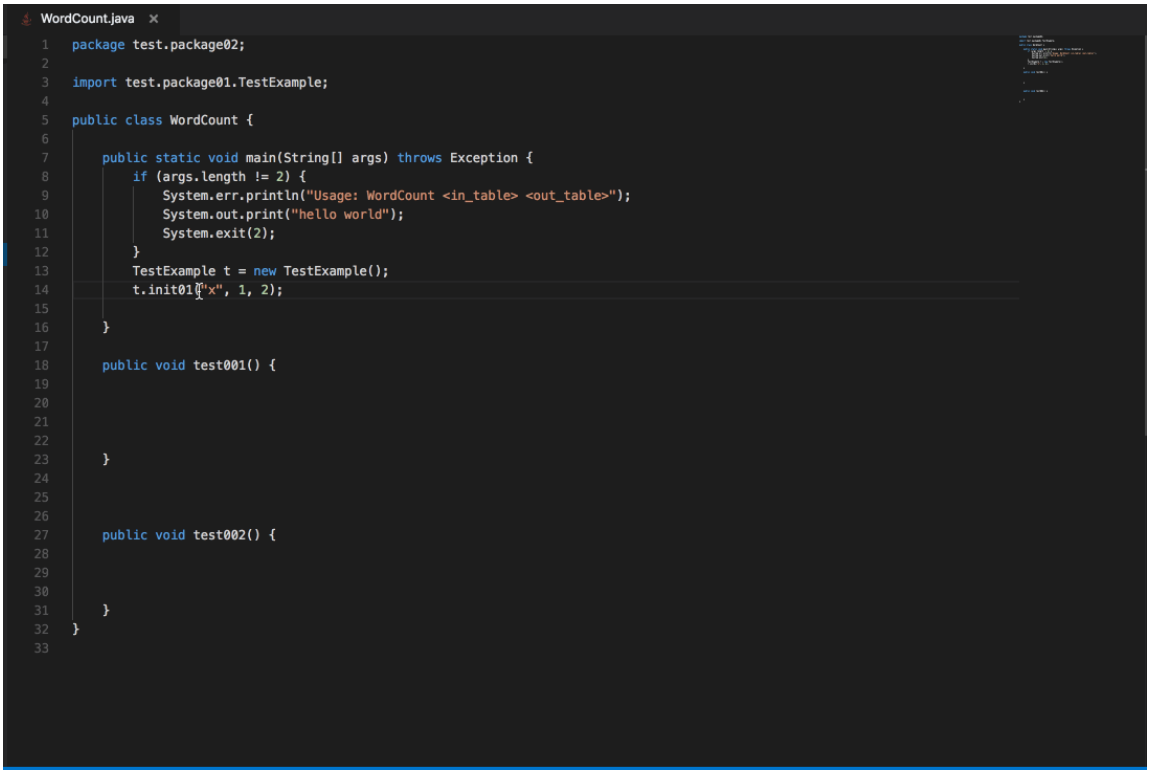
- 智能诊断



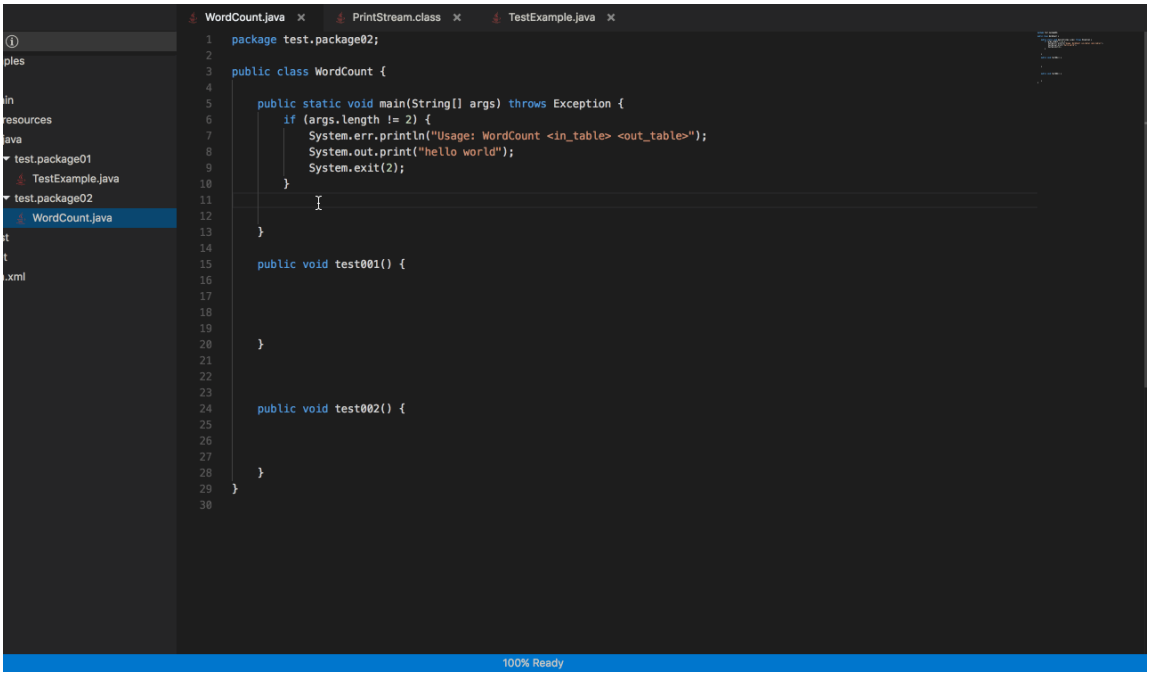
- 查找定义


WordCount.java
PrintStream.class
TestExample.java
package test.package02;
import test.package01.TestExample;
public class WordCount {
public static void main(String[] args) throws Exception {
if (args.length != 2) {
System.err.println("Usage: WordCount <in_table> <out_table>");
System.out.print("hello world");
System.exit(2);
}
System.out.print(true);
TestExample t = new TestExample();
t.init01("x", 1, 2);
System.out.println("hello");
}
public void test001() {
}
public void test002() {
}
}

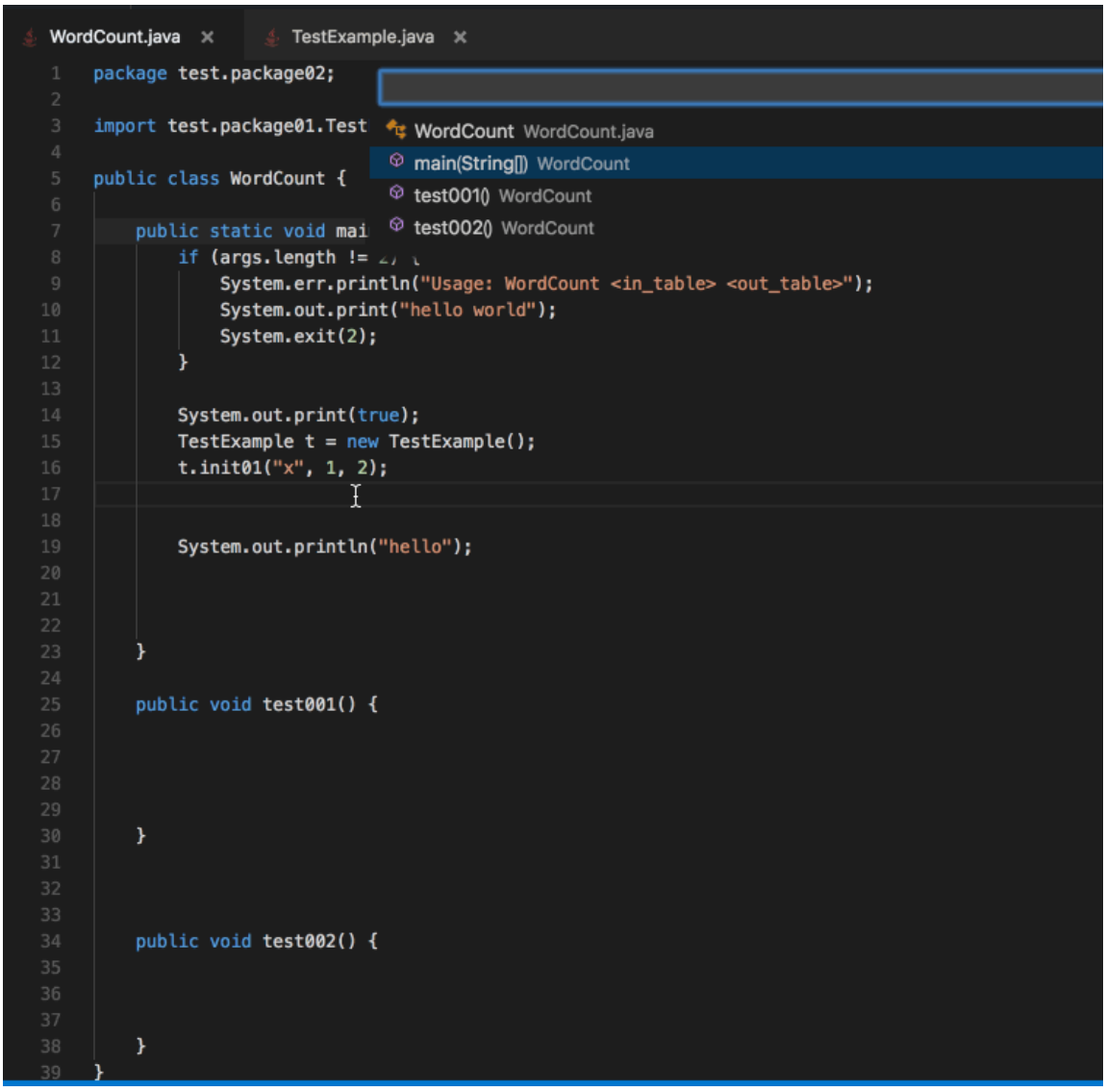
- **查找引用**


WordCount.java
package test.package02;
import test.package01.TestExample;
public class WordCount {
public static void main(String[] args) throws Exception {
if (args.length != 2) {
System.err.println("Usage: WordCount <in_table> <out_table>");
System.out.print("hello world");
System.exit(2);
}
TestExample t = new TestExample();
t.init01("x", 1, 2);
}
public void test001() {
}
public void test002() {
}
}

- **自动导入**


WordCount.java
PrintStream.class
TestExample.java
test.package01
TestExample.java
test.package02
WordCount.java
100% Ready

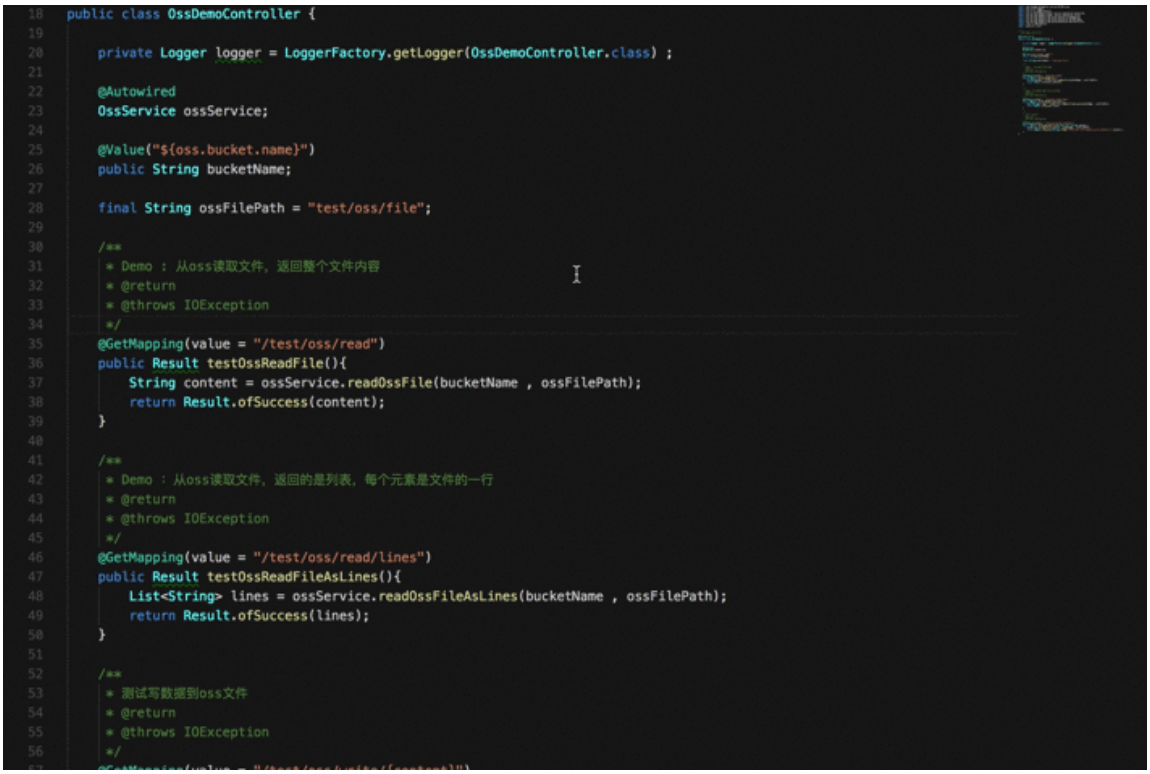
- **查找符号**


WordCount.java
TestExample.java
WordCount WordCount.java
main(String[]) WordCount
test001() WordCount
test002() WordCount

- **多光标编辑**

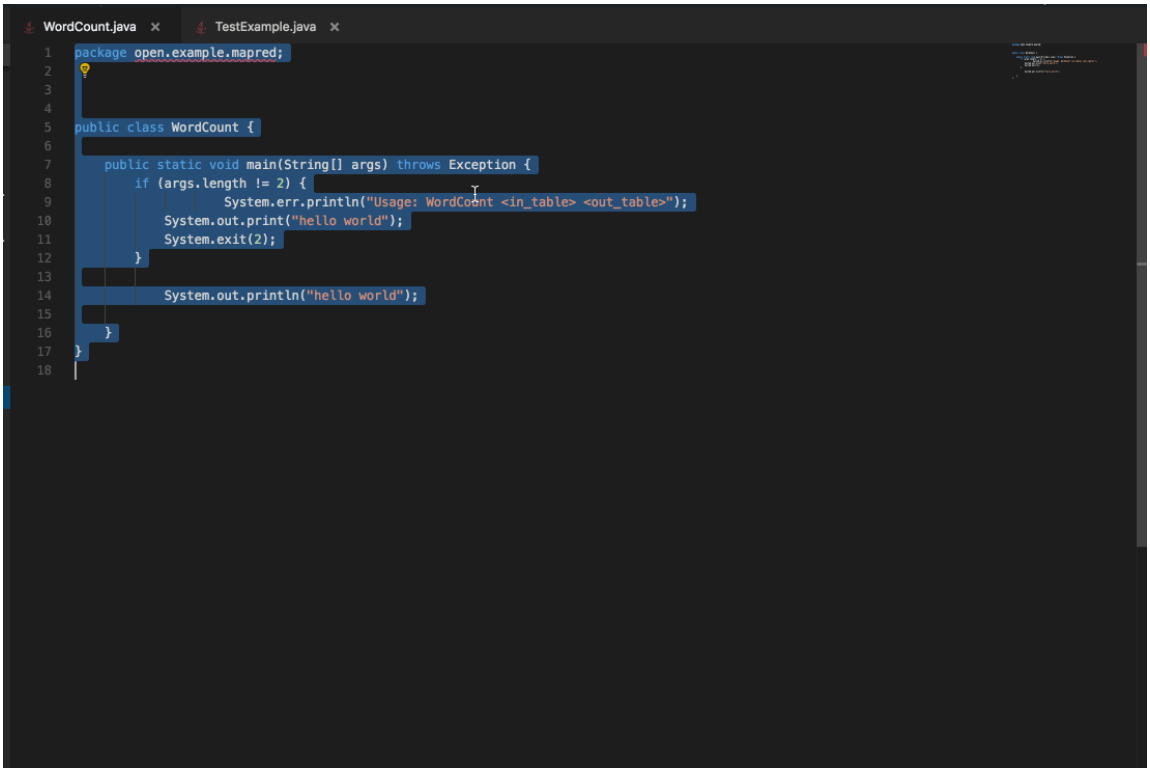
```java
@Controller
public class IndexController {

    @GetMapping(value = { "/","/index" , "/index.htm" , "/index.html"})
    public String index(Model model){
        return "index";
    }

}
```

- **查找、替换**

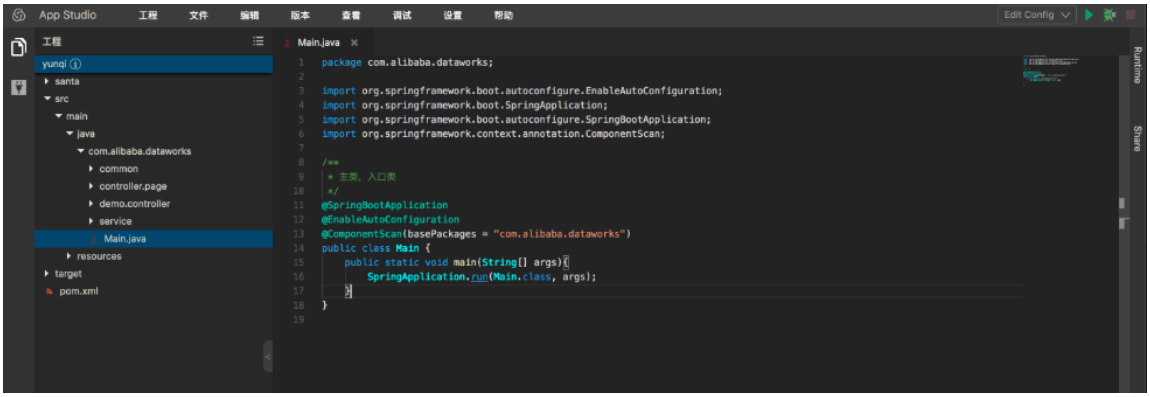


```java
public class OssDemoController {

    private Logger logger = LoggerFactory.getLogger(OssDemoController.class) ;

    @Autowired
    OssService ossService;

    @Value("${oss.bucket.name}")
    public String bucketName;

    final String ossFilePath = "test/oss/file";

    /**
     * Demo : 从oss读取文件，返回整个文件内容
     * @return
     * @throws IOException
     */
    @GetMapping(value = "/test/oss/read")
    public Result testOssReadFile(){
        String content = ossService.readOssFile(bucketName , ossFilePath);
        return Result.ofSuccess(content);
    }

    /**
     * Demo : 从oss读取文件，返回的是列表，每个元素是文件的一行
     * @return
     * @throws IOException
     */
    @GetMapping(value = "/test/oss/read/lines")
    public Result testOssReadFileAsLines(){
        List<String> lines = ossService.readOssFileAsLines(bucketName , ossFilePath);
        return Result.ofSuccess(lines);
    }

    /**
     * 测试写数据到oss文件
     * @return
     * @throws IOException
     */
```

- **代码格式化**

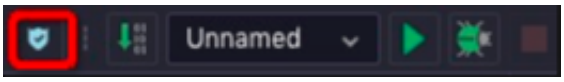


- 括号匹配

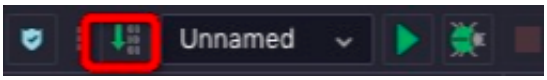


· 右上角图标区

- 编码规约



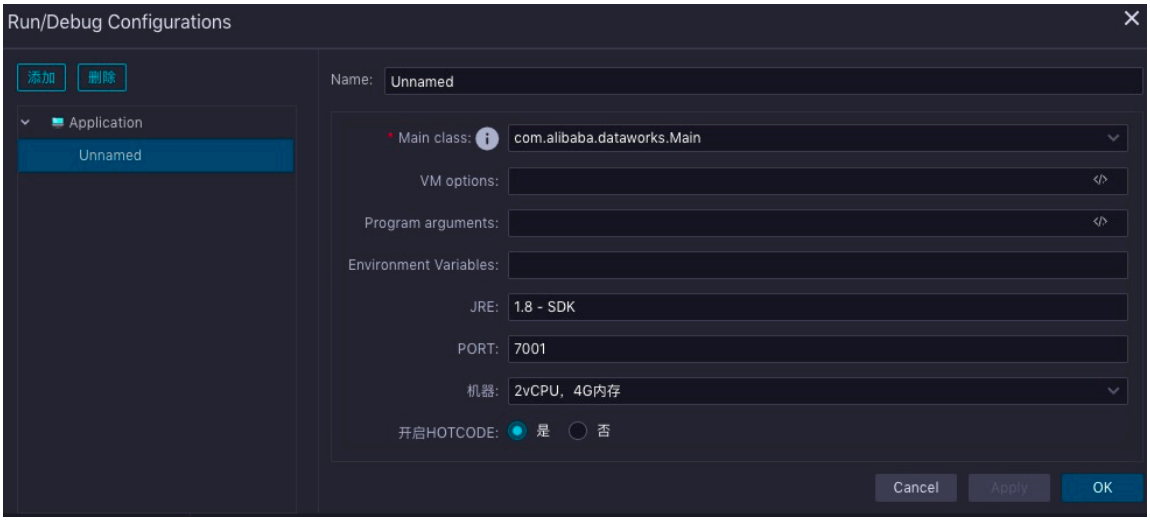
- 构建



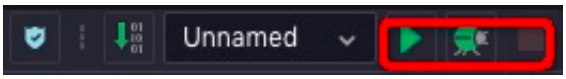
说明:

构建需要在工程运行或者debug时才能进行。

- **Run/Debug Configurations**



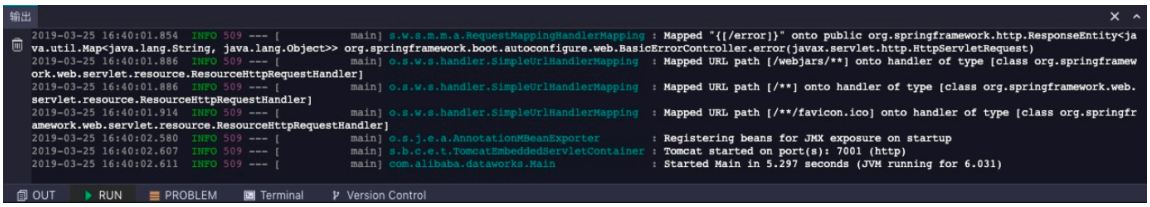
- **Debug入口**



从左到右的图标依次代表运行、Debug和停止工程。

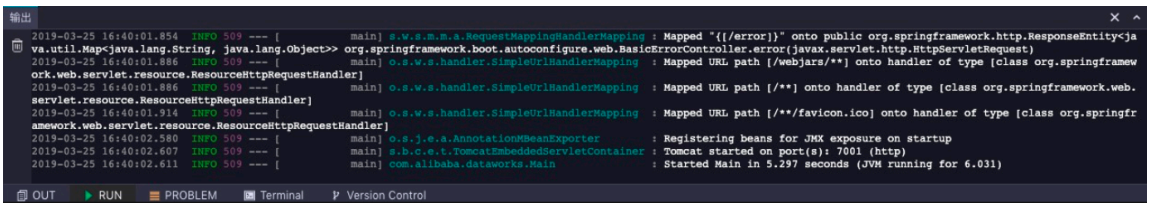
- 底边栏
  - DEBUG/RUN面板

    单击运行或Debug工程，该面板会弹出，展示进度和信息。



  - PROBLEM面板

    当工程有问题时，运行或Debug工程该面板会弹出。



  - TERMINAL面板

    当工程运行或Debug时，可以通过Terminal触达机器进行bash、vim命令操作。

  - VERSION CONTROL面板

    该面板展示Git history和Git log两部分内容。

- 右边栏
  - Runtime

    工程运行完成时会展开这个面板，并展示机器信息和访问链接。

    - 如果是后端工程，仅展示后端访问链接。
    - 如果是前端工程，仅展示前端链接。
    - 如果是可视化搭建工程，可展示前端访问链接和后端访问链接。
  - Share

    您可以邀请他人协同编程，目前支持8人同时编辑同一工程同一文件。

- **Data**

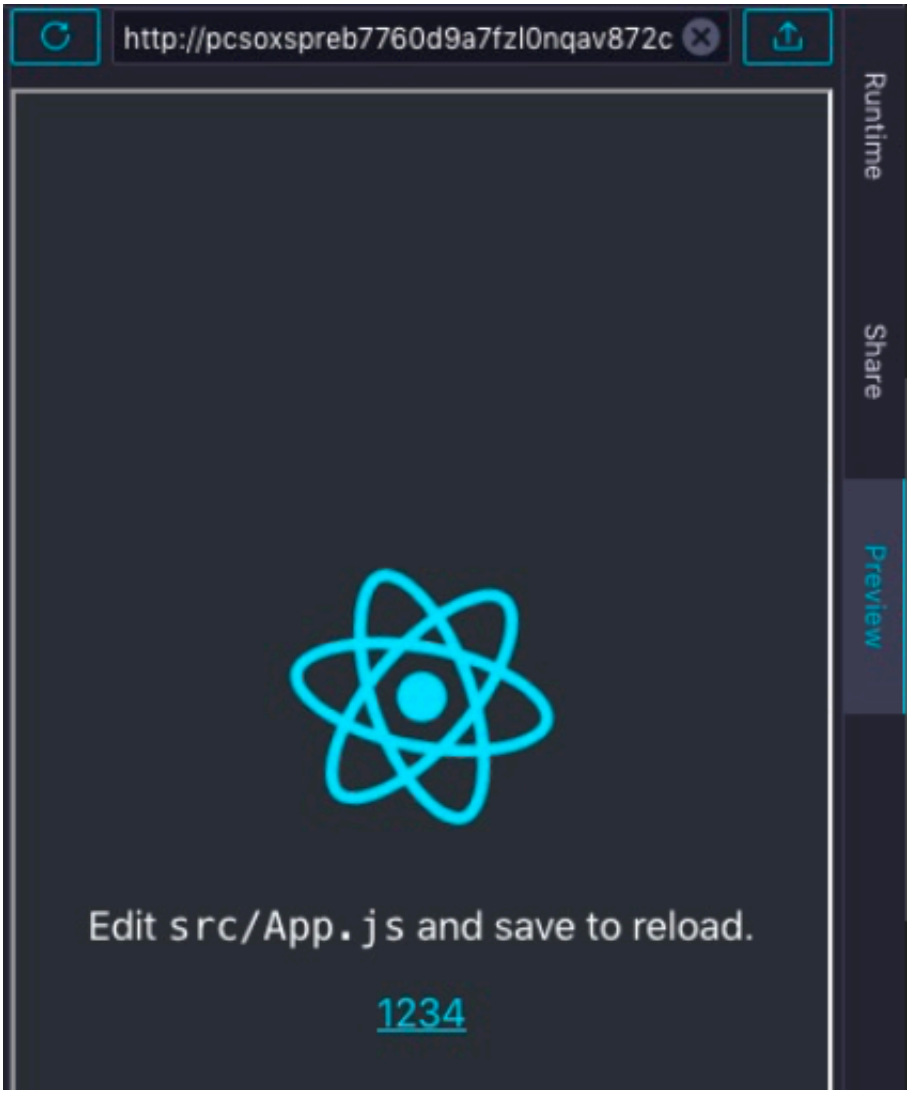
  数据服务是承接DataStudio和App Studio的重要一环。

数据服务在App Studio中有两种使用方式，更多详情请参见数据服务。

- **可以在代码中直接使用，或者对接口结果进行再加工。**
- **可以在可视化搭建中直接配置为组件的数据源。**

- **Preview**

  如果您创建的是前端工程，在右边栏会有Preview入口，在运行工程时可以实时预览前端页面。

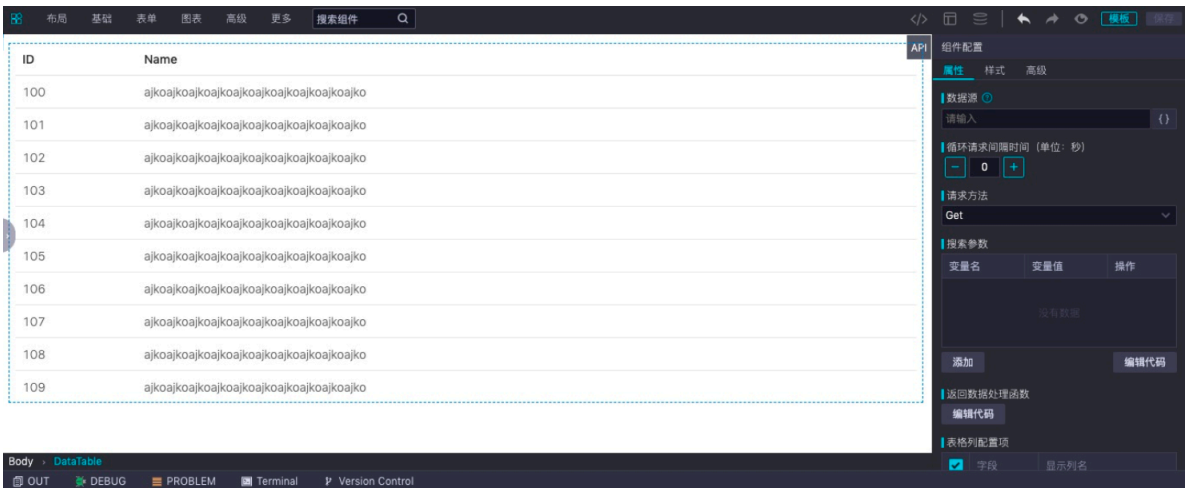

http://pcsoxspreb7760d9a7fzl0nqav872c
Runtime
Share
Preview
Edit src/App.js and save to reload.
1234

- 可视化搭建区

  您的工程必须是可视化搭建工程，在工程中找到santa-pages下的.santa文件，双击打开。

  左上角是组件区，您可以选择需要的组件，也可以通过搜索组件名称选择需要的组件。右上角的图标分别是切换为代码模式、导航配置、全局数据流配置、撤销、重做、预览、保存为模板和保存。

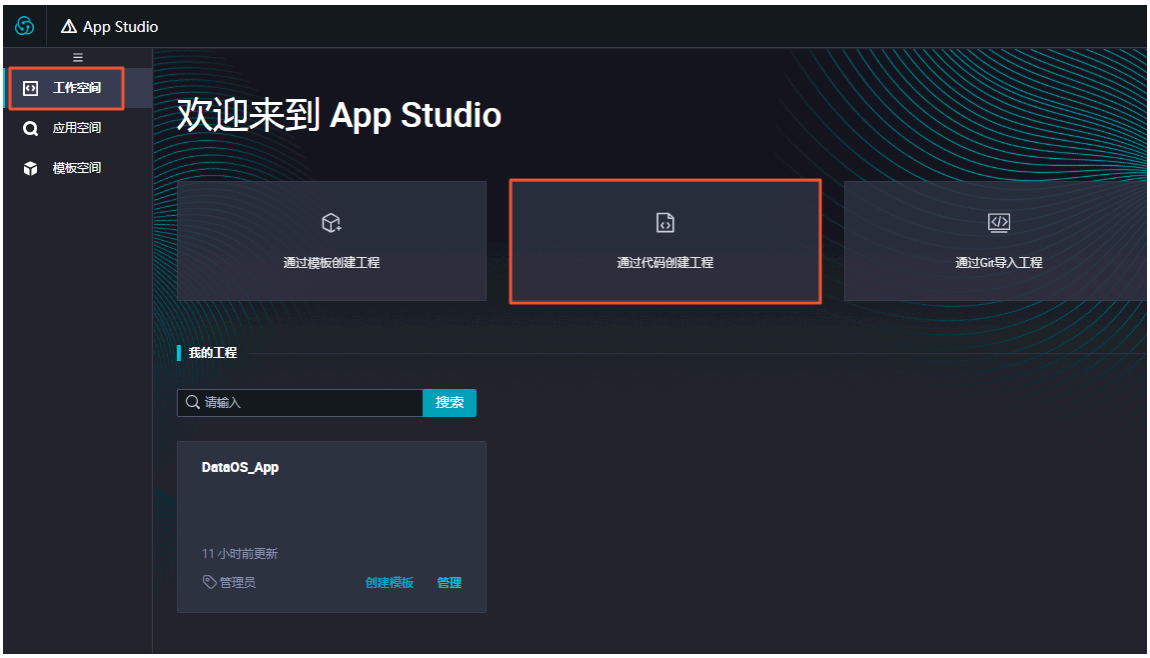
  拖动一个表格组件到画布，单击表格组件，右边会弹出组件配置区，可以对组件的属性、样式进行配置，也可以进行组件联动配置。



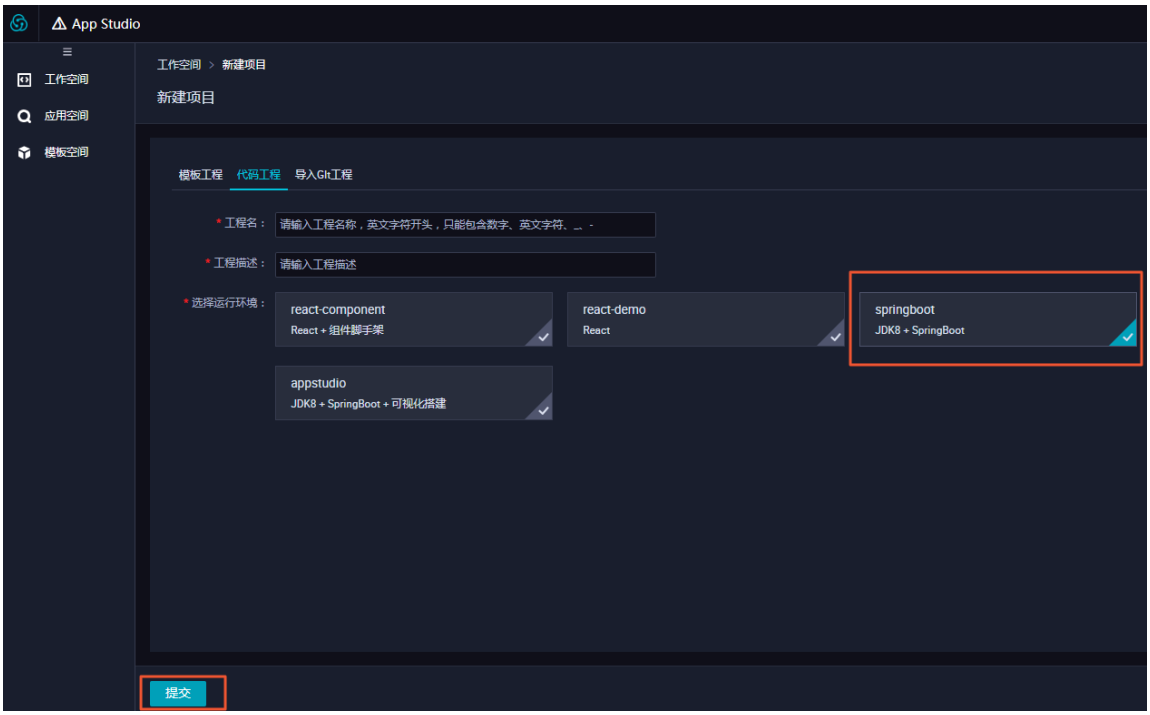
创建后端工程

1. 基于样例工程新建工程。

   a. 进入App Studio页面，单击工作空间页面的通过代码创建工程。



   b. 填写新建项目对话框中的工程名和工程描述，选择运行环境为springboot样例模板。

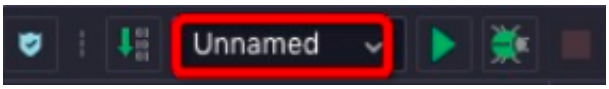


   c. 配置完成后，单击提交。

2. 配置运行参数。

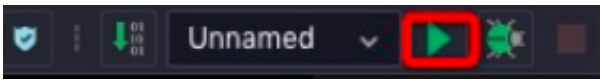
填写好配置的名称，选择运行的main函数，选择机器规格，单击OK即可完成配置。

您可以通过左边的添加按钮添加多个配置，运行时选择不同的配置运行。

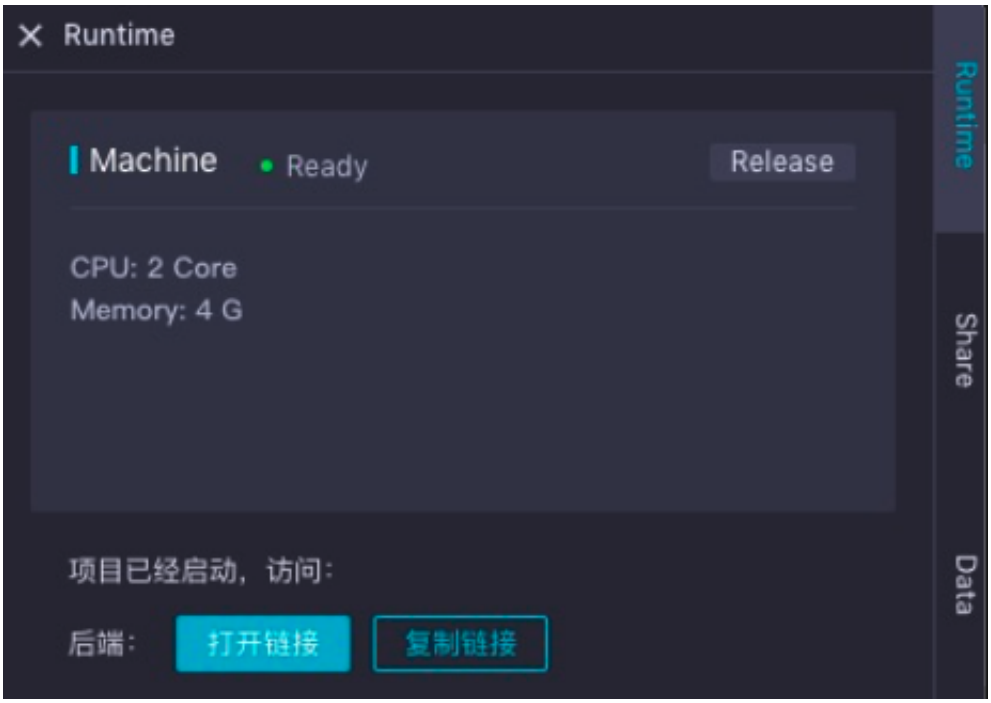


3. 运行工程

单击红框中的运行图标开始运行工程。



第一次运行需要分配机器、初始化语言服务，需要较长时间运行完成，完成后会弹出runtime窗口，展示访问链接。



4. 访问工程

单击打开链接，即可访问工程。

在链接中加上/testapi并刷新页面。

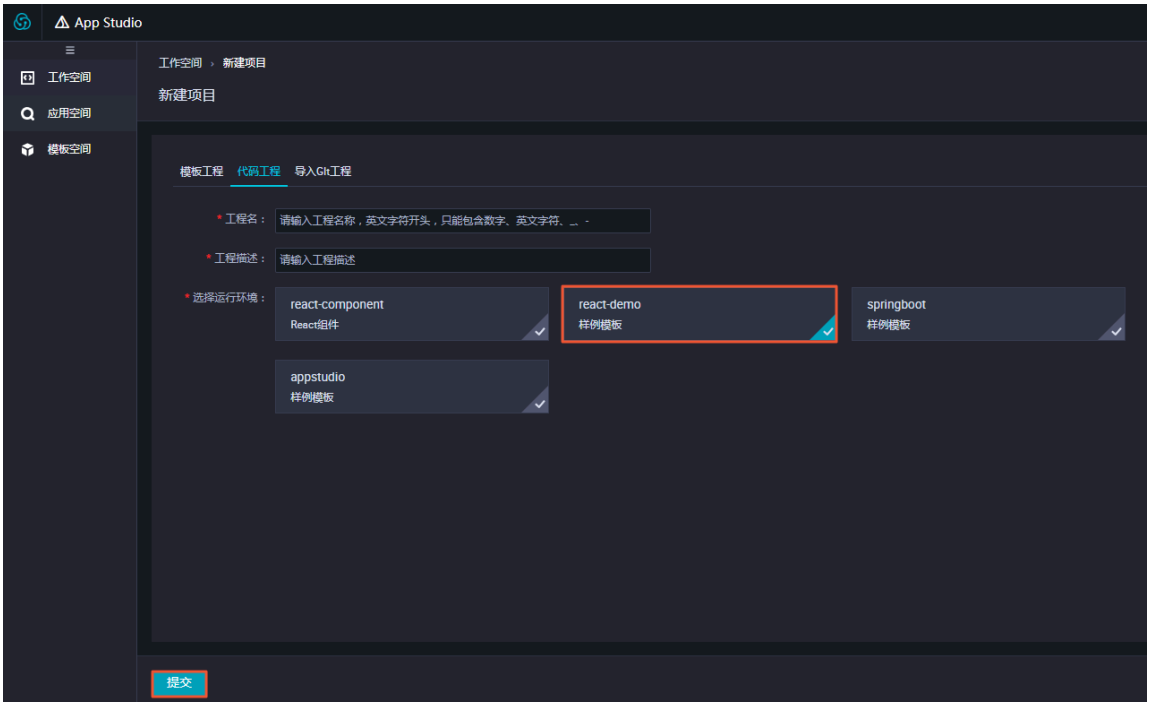
创建前端工程

App Studio提供完善的前端开发能力，支持与本地一致的前端开发体验。您可以在App Studio中创建前端工程，以自己熟悉的方式进行HTML、CSS、JS和React的开发，并且您在开发过程中不需要掌握和理解新的概念。

1. 基于样例工程新建工程。

   a. 进入App Studio页面，单击工作空间页面的通过代码创建工程。

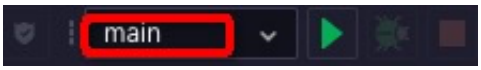
   b. 填写新建项目对话框中的工程名和工程描述，选择运行环境为react-demo样例模板。

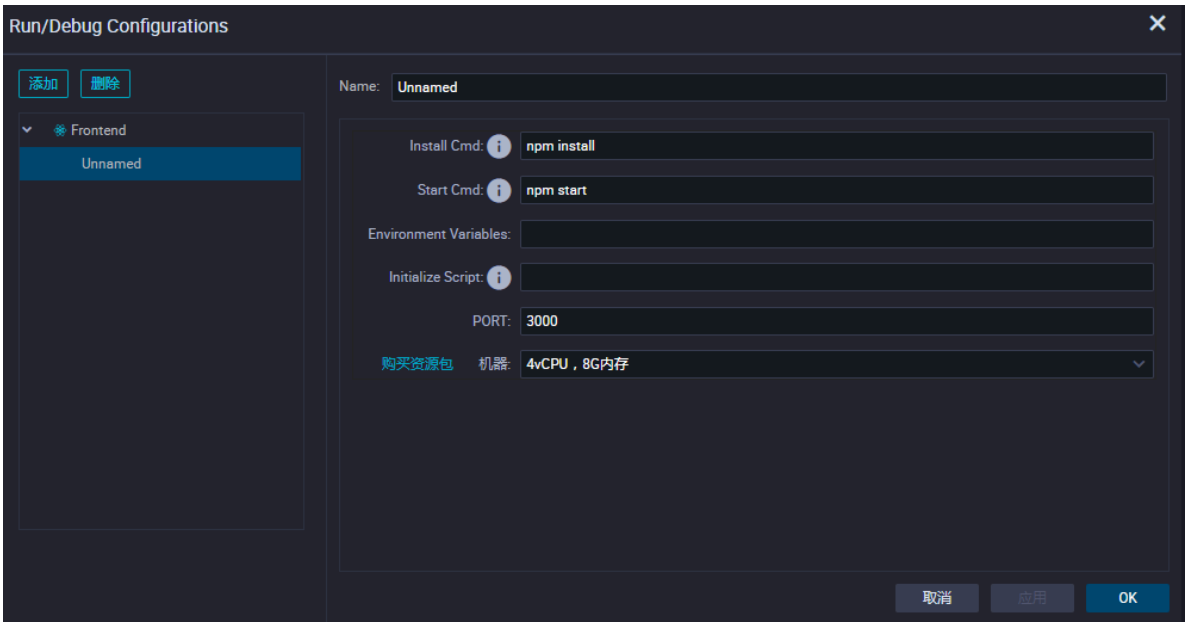


   c. 填写工程名和工程描述，单击确认。

2. 配置运行参数。

   您可以选择机器规格、配置端口，如果没有特殊需求可以直接使用默认的配置，单击OK即可。

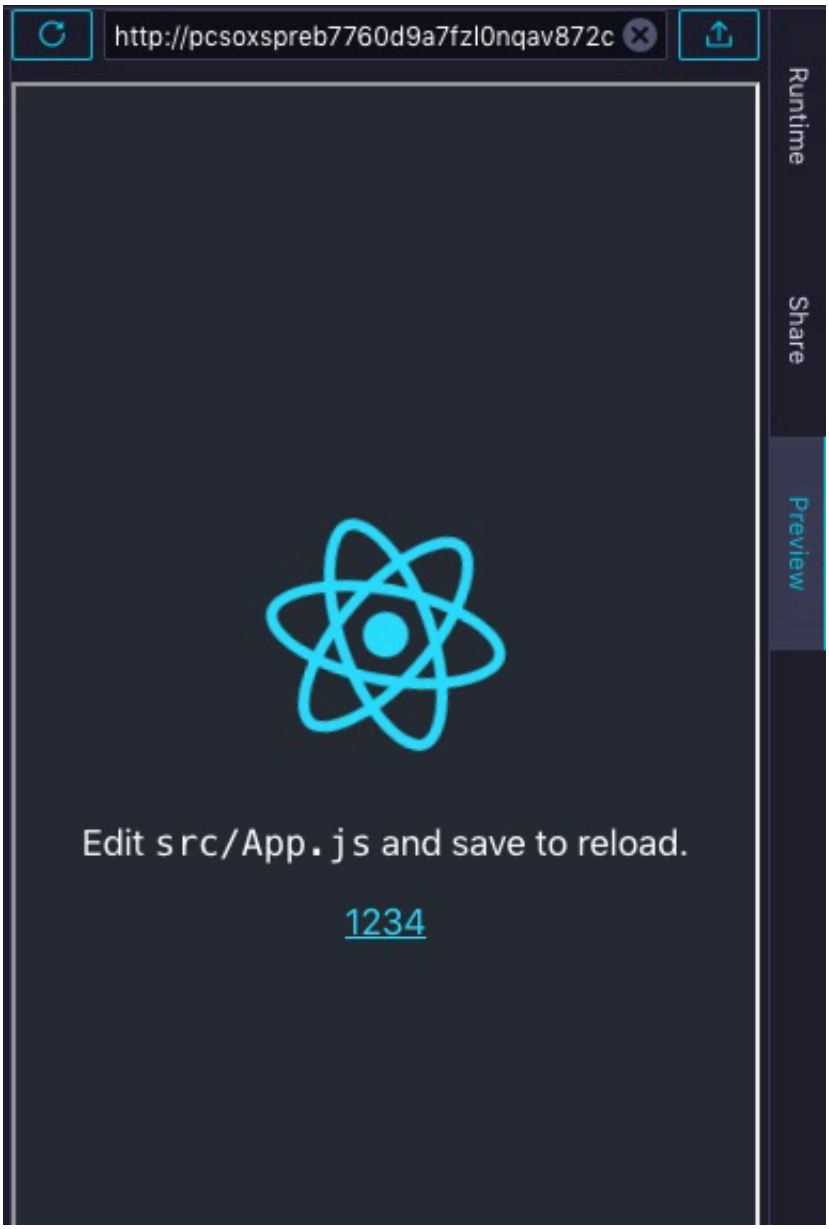




3. 运行工程

单击右上角的运行图标开始运行工程，目前支持以`tnpm start`的方式启动前端工程，配置了*webpack-dev-server*的工程可以无缝对接运行。

启动运行后，开发者可以在日志中看到依赖安装及应用启动的日志，运行完成后右边会弹出页面的预览页面。您可以实时修改代码并进行保存，便可实时生效。



4. 访问工程

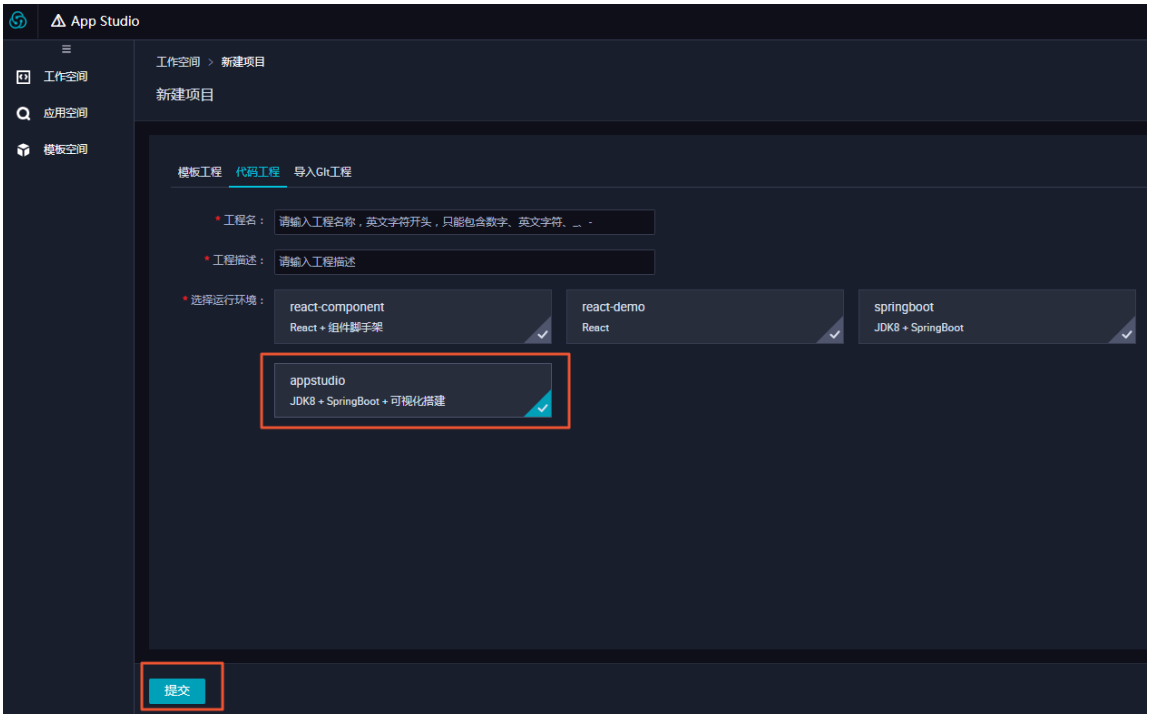
单击链接边的箭头即可打开访问页面，App Studio对于前端工程的编辑开发提供了与本地IDE一致的开发体验，包括HTML、CSS、LESS、SCSS、JavaScript、TypeScript、JSX和TSX等文件的智能补全、函数签名、重构和跳转等功能。同时您不需进行搭建环境、下载依赖等操作，可以在模板基础上进行前端开发。

搭建前端可视化工程

1. **基于样例工程新建工程。**

   a. **进入App Studio页面，单击工作空间页面的通过代码创建工程。**

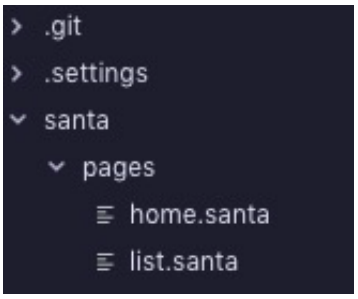
   b. **填写新建项目对话框中的工程名和工程描述，选择运行环境为App Studio样例模板。**



   c. **配置完成后，单击提交。**

2. 打开home.santa文件。

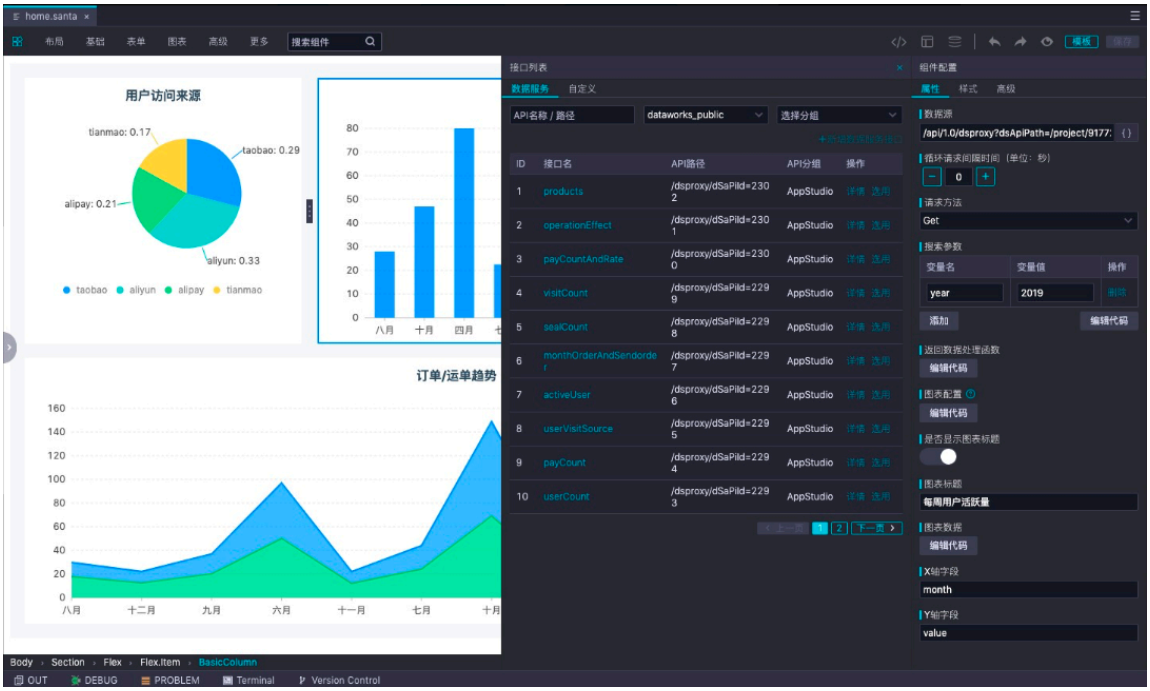
在santa-pages目录下找到.santa文件，有home和list两个样例页面。



a. 打开home.santa，是一个简单的报表页面。

b. 选中一个组件，右边会弹出组件配置。

c. 单击数据源输入框，会弹出接口列表。



App Studio为您提供一些数据服务接口，以便您入门使用。您可以单击+新增数据服务接口前往数据服务中新增接口，通过API路径查看现在的组件对应的接口。

说明:

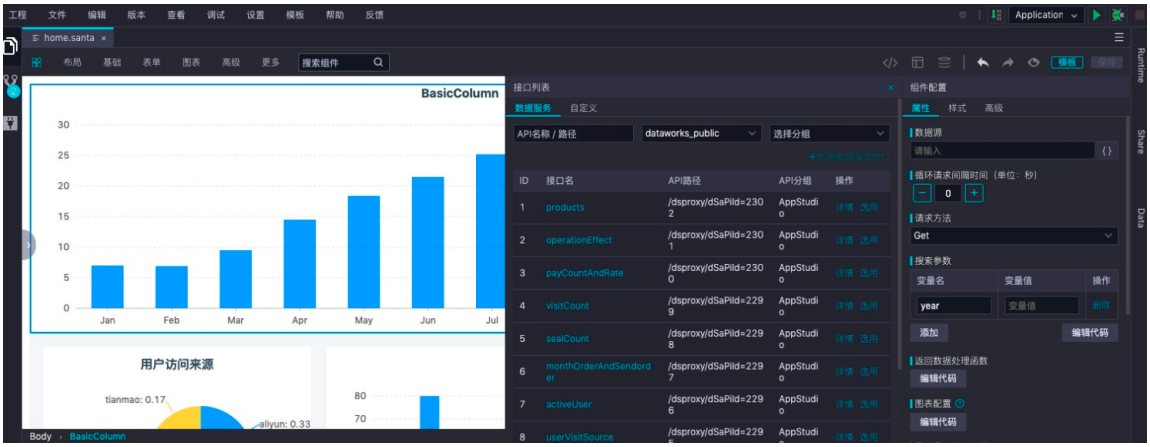
您可以尝试去掉接口自行配置，体验组件配置数据源的效果，也可以对样式进行修改。

3. 添加组件&配置接口。

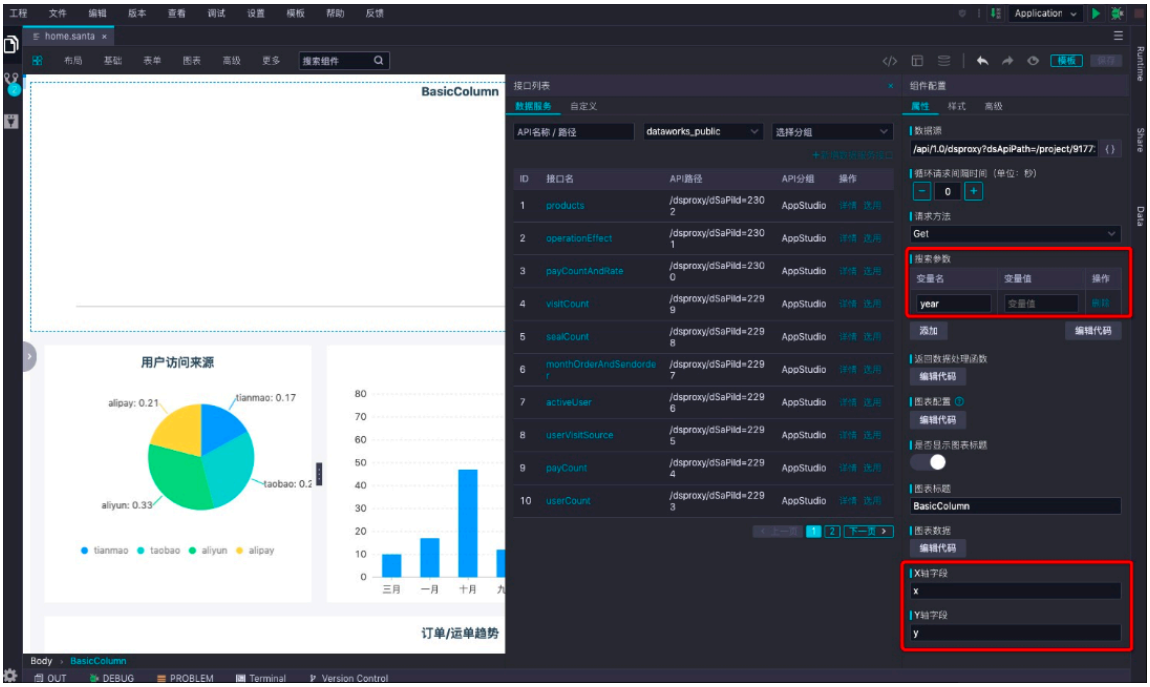
a. 从图表中拖动一个柱状图到画布上。

b. 选中组件，单击弹出的组件配置框中的数据源输入框。

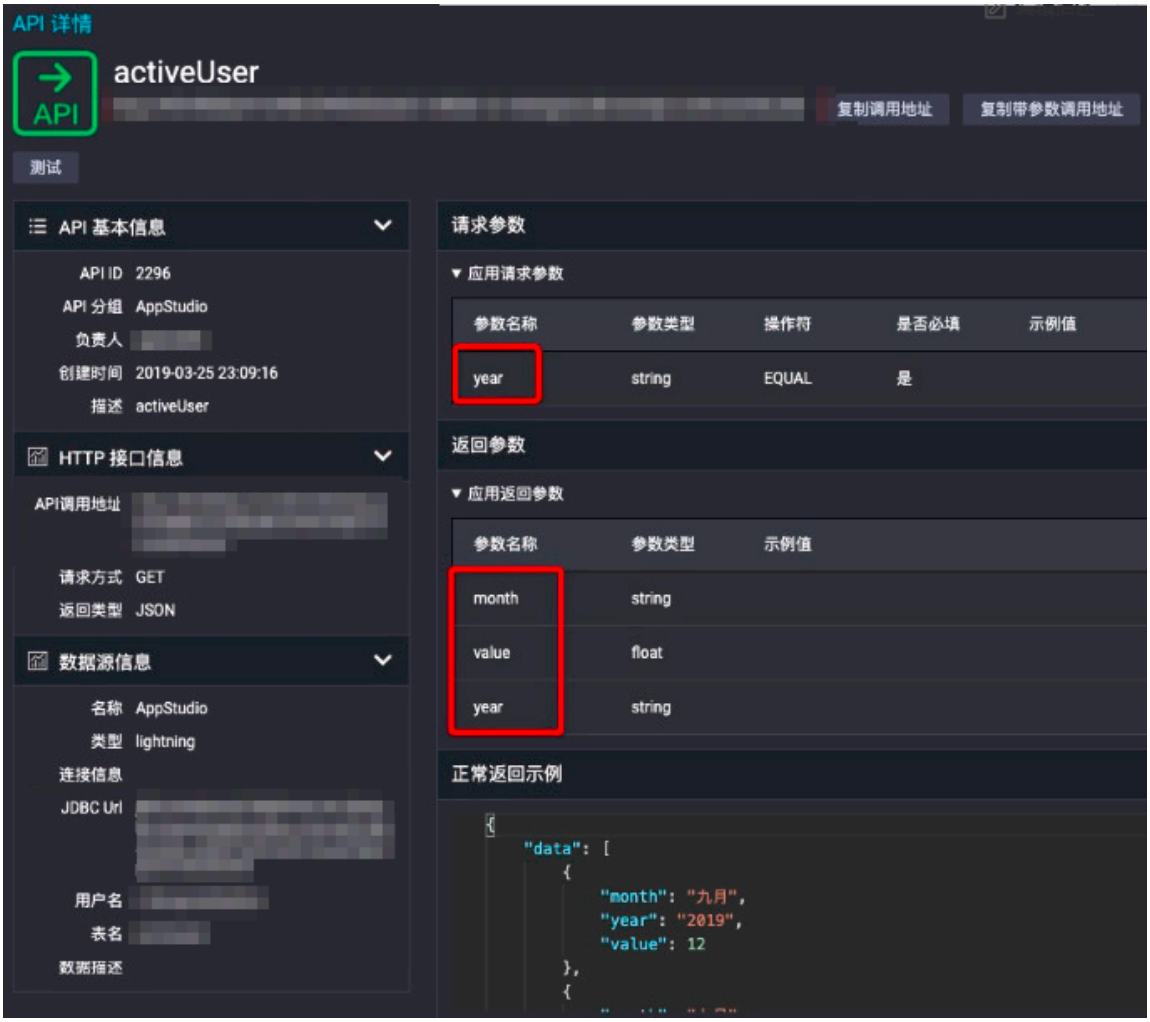
c. 选择第7个接口，单击选用，便成功配置接口。



d. 此时组件中没有返回结果，是因为此接口需要填写入参和返回的列。



您可以单击第7个接口的详情，查看请求和返回的内容。



> **说明:**
> 由于这是示例项目，您会无权访问，建议您搭建可视化工程时，使用自己的账号到数据服务创建接口。

e. 按照下图填写组件配置。

配置完成后，即可看到组件中已有配置好的数据。

4. 打开list.santa文件。

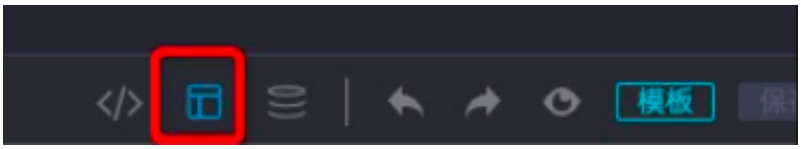
   App Studio可视化搭建不仅可以搭建报表，还可以搭建应用。

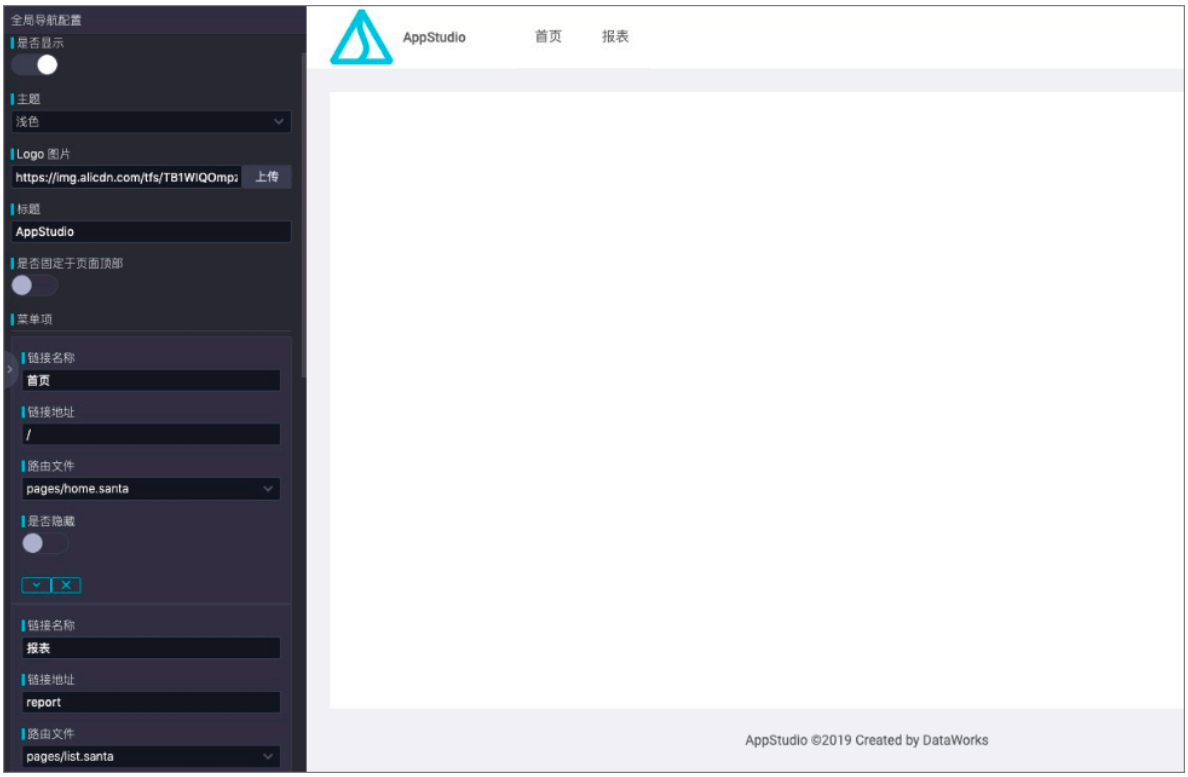
   打开`list.santa`文件，是一个简单的数据应用。其中包括图标、链接、视频、列表和搜索等组件，详情请参见可视化搭建概述。

5. 导航配置。

   您搭建一个应用，通常不会只有一个页面，多个页面之间需要一个导航配置。

   单击右上角的导航配置标识，即可打开导航配置页面。





6. 配置运行参数，可参见搭建后端工程的操作。
7. 运行工程。

   单击右上角的运行标识即可开始运行工程，运行后会弹出Runtime面板，单击里面的前端链接即可访问工程。

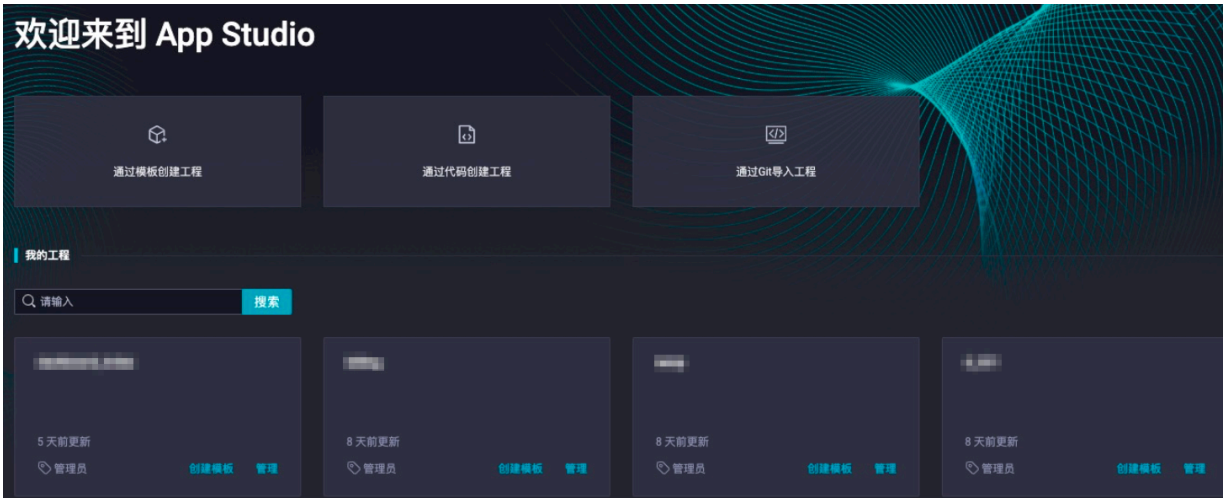
1.4 功能介绍

1.4.1 导航页

1.4.1.1 工作空间

您可在工作空间页面创建和管理工程。

App Studio的工作空间页面，将为您展示当前创建的工程列表，并提供三种创建工程的方式，详情请参见工程管理。



单击工程卡片，即可进入工程开发页面。您也可单击创建模板或管理，进行相关操作。

创建模板

1. **单击相应工程下的创建模板。**
2. **填写生成模板对话框中的配置。**

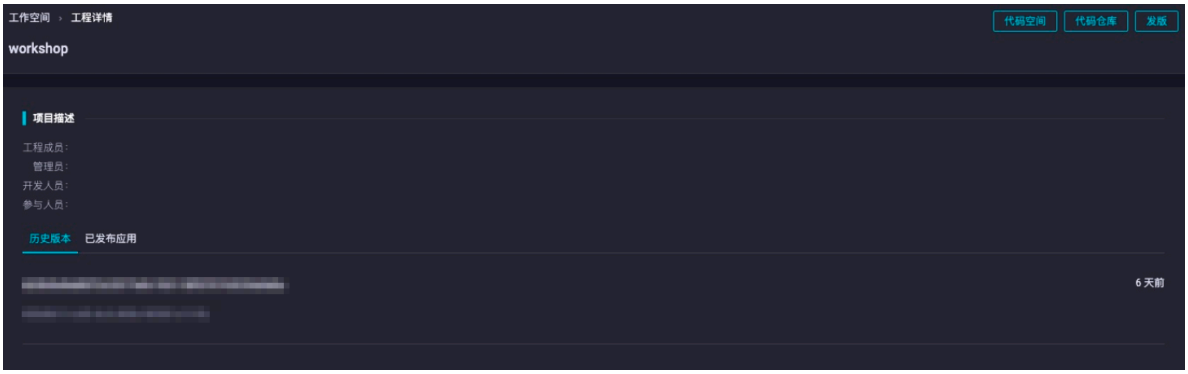
配置	说明
模板名称	输入模板的名称。
模板描述	对模板进行简单描述。
分类	选择模板的分类。

3. **填写完成后，单击生成。**

管理

您的工程可以发布为一个应用，为方便您的版本管理，您可将工程发布成不同的版本，然后再进行应用发布。

1. 单击相应工程下的管理，即可进入工程管理页面。



2. 单击右上角的发版，选择要更新版本的应用。

> **说明:**
> 您需要将工程绑定Git仓库后，方可进行发版。

3. 配置完成后，单击更新，即可产生一个新的版本。

1.4.1.2 应用空间

应用空间包括我开发的应用、我分享的应用和第三方应用三大模块。

> **说明:**
> - 仅购买旗舰版的用户，可以查看我分享的应用。
> - 仅购买企业版和旗舰版的用户，可以查看第三方应用。

我开发的应用

我开发的应用页面为您展示已开发的应用，您可以对应用进行发布，也可以通过部署控制台进入应用运维页面。

说明:

分享入口仅对购买旗舰版的用户可见。

部署控制台

单击相应应用下的部署控制台，即可进入运维页面。

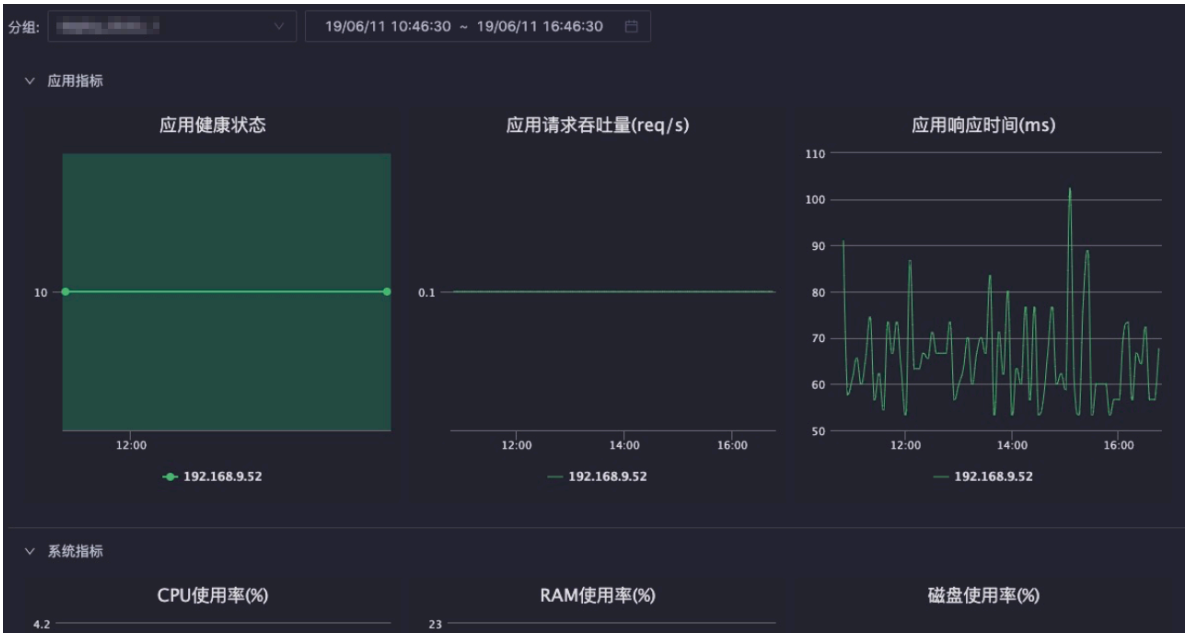
运维页面为您展示所有应用的运维情况，您可以在左侧下拉框选择需要查看的应用。

- **概览**

  概览页面为您展示应用信息、应用状态、分组信息、机器信息、分组列表和机器列表等信息。

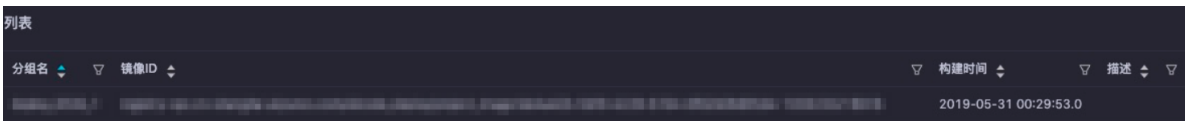
- **监控**

  监控页面为您展示应用的详细运维指标，包括3个应用指标、8个系统指标和7个JVM指标。



- **镜像**

  镜像页面为您展示每个分组使用的镜像和构建时间。



- 变更

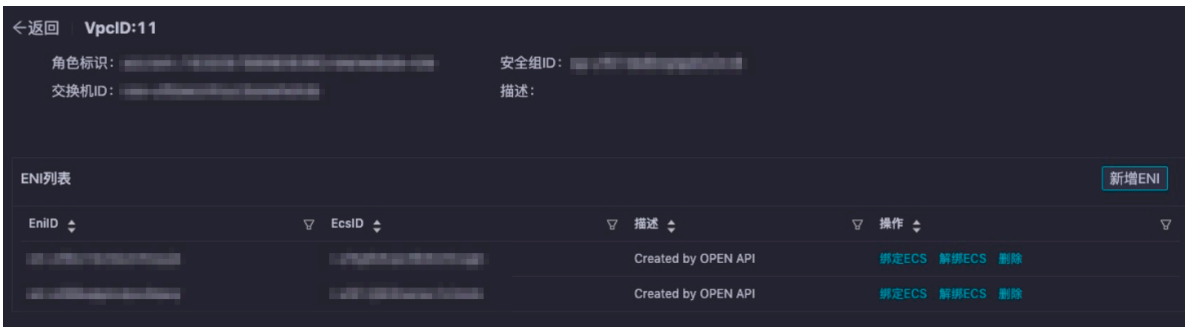
  变更页面为您展示进行的部署、应用扩容或机器下线等操作。单击变更单ID，即可查看变更详情。

  当应用正在部署时，可以在此查看详细的部署信息和日志。

- 资源

  资源页面为您展示您所购买的VPC。购买VPC后，需要在此进行新增操作。

  单击相应的ID，即可进入VPC详情页。



- 操作

  您可以进行应用重启、机器重启、机器下线和应用扩容四项操作。

  - 应用重启

    在应用重启对话框中，对当前操作进行简要描述。单击执行，即可重启整个应用。

  - 机器重启

    在机器重启对话框中，选择分组和机器，并进行简要描述。单击执行，即可对当前应用的某个分组下的某台机器进行重启。

- **机器下线**

  在机器下线对话框中，选择分组和机器，并进行简要描述。单击执行，即可将当前应用的某个分组下的某台机器移除，放回资源池中。

- **应用扩容**

  在应用扩容对话框中，选择扩容分组、可用机器，并进行简要描述。单击执行，即可将您的资源池中的机器，加入到当前应用的某个分组下。

发布

单击相应应用下的发布，即可进行发布操作，详情请参见应用部署。

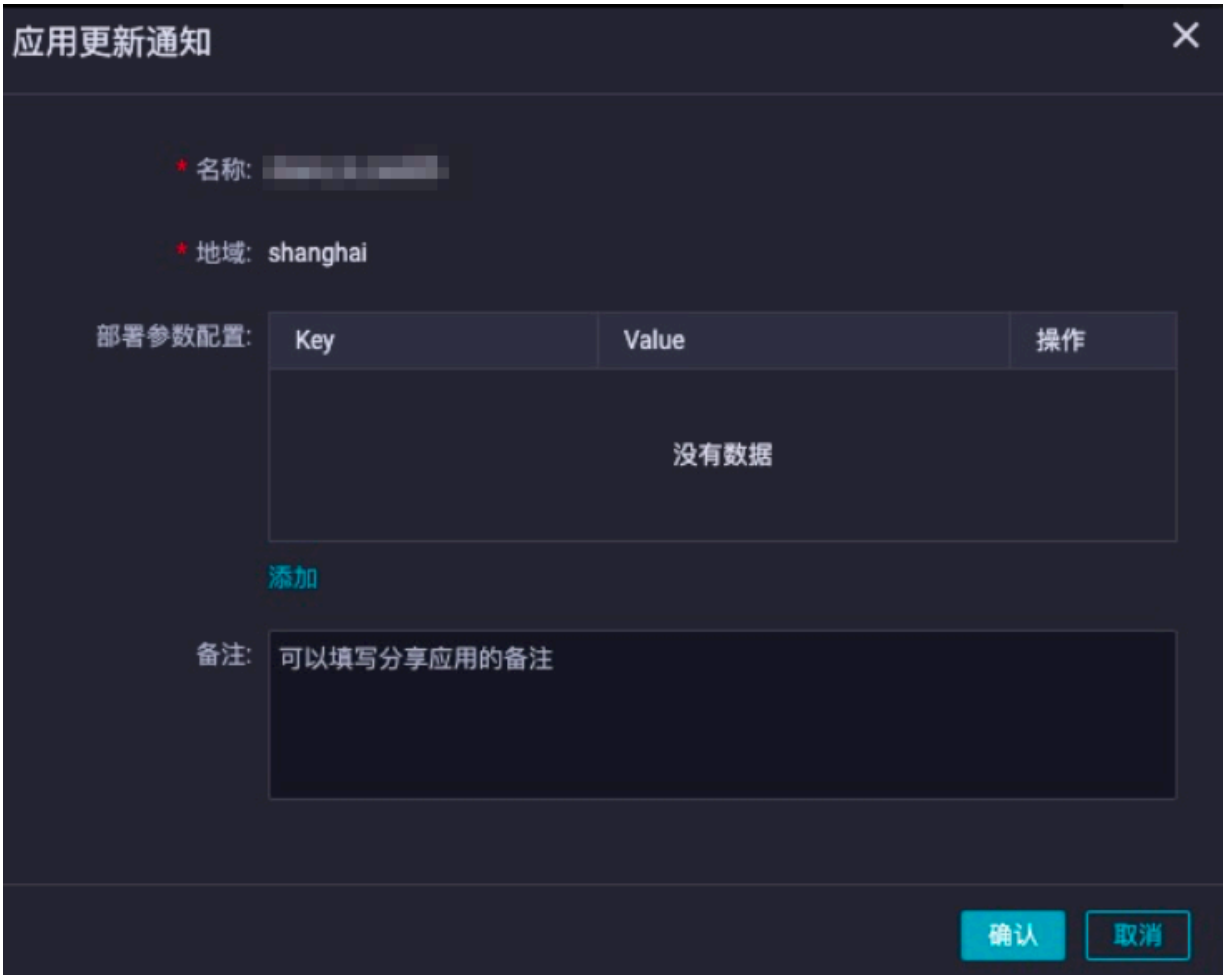
分享

单击相应应用下的分享，购买企业版及以上版本的用户，可以将应用分享给其他用户。分享成功后，您可以在我分享的应用列表中进行查看，对方可以在第三方应用的列表中进行查看。

我分享的应用

进入应用空间 > 我分享的应用页面，即可查看分享过的应用。

单击相应应用下的部署通知，可以将应用的代码更新推送给被分享的用户，进行应用部署。



第三方应用

进入应用空间 > 第三方应用页面，可以查看别人分享给您的应用，并进行部署和发布等操作，操作方式和应用空间一致。

1.4.1.3 模板空间

模板空间为您展示所有通过工程创建的模板。

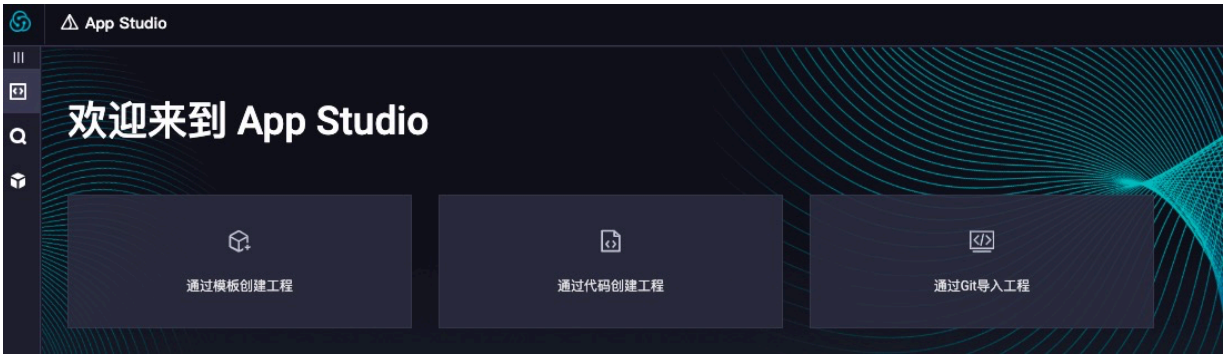
您可以单击相应的模板卡片，进入模板详情页面。然后单击代码空间，即可查看模板相应的工程代码。

您也可以直接单击相应模板下创建工程，即可跳转至通过模板创建工程页面，基于当前模板创建工程。

1.4.2 工程管理

本文将为您介绍如何新建和管理工程。

您可以通过模板、代码和Git导入三种方式新建工程。



通过模板创建工程

1. 进入App Studio页面，单击工作空间页面的通过模板创建工程。

2. 填写新建项目对话框中的工程名和工程描述，选择相应的模板。

说明:

- 您可以选择自己定义的模板，也可以选择系统提供的模板创建工程。
- 通过模板创建的都是可视化工程。

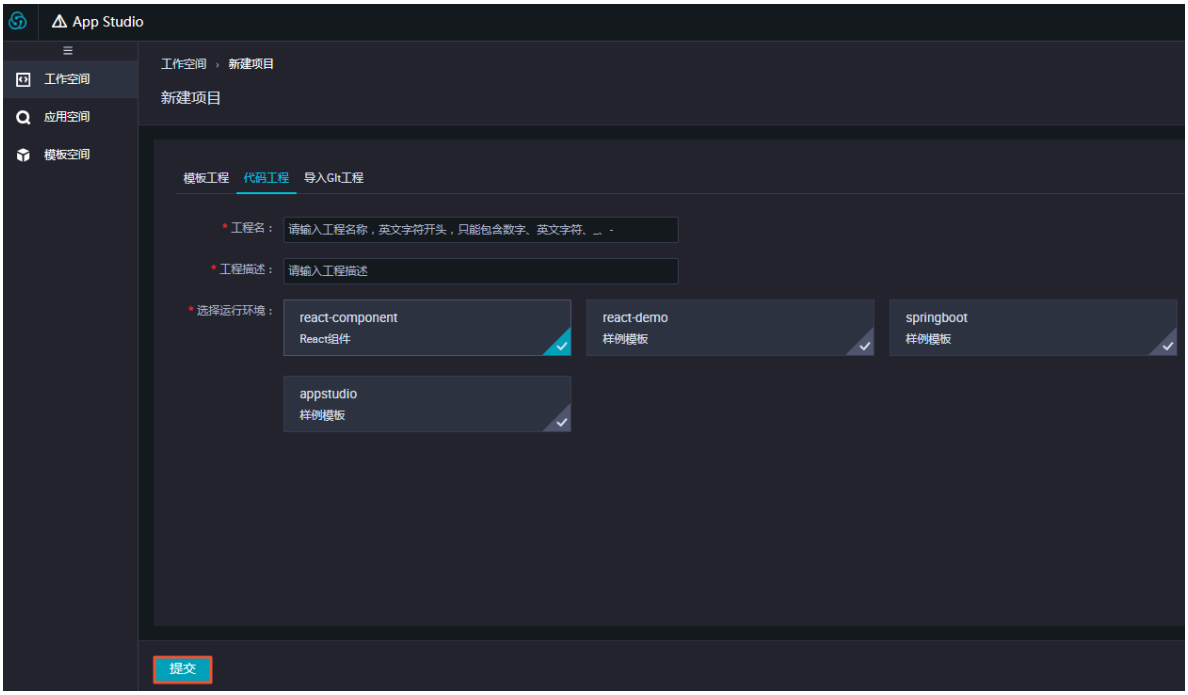
3. 配置完成后，单击提交。

通过代码创建工程

如果想进行纯代码开发的工程，可以通过代码创建工程。App Studio提供了4种运行环境的代码模板，您可以根据自身需求进行选择。

1. 进入App Studio页面，单击工作空间页面的通过代码创建工程。

2. 填写新建项目对话框中的工程名和工程描述，选择相应的模板。

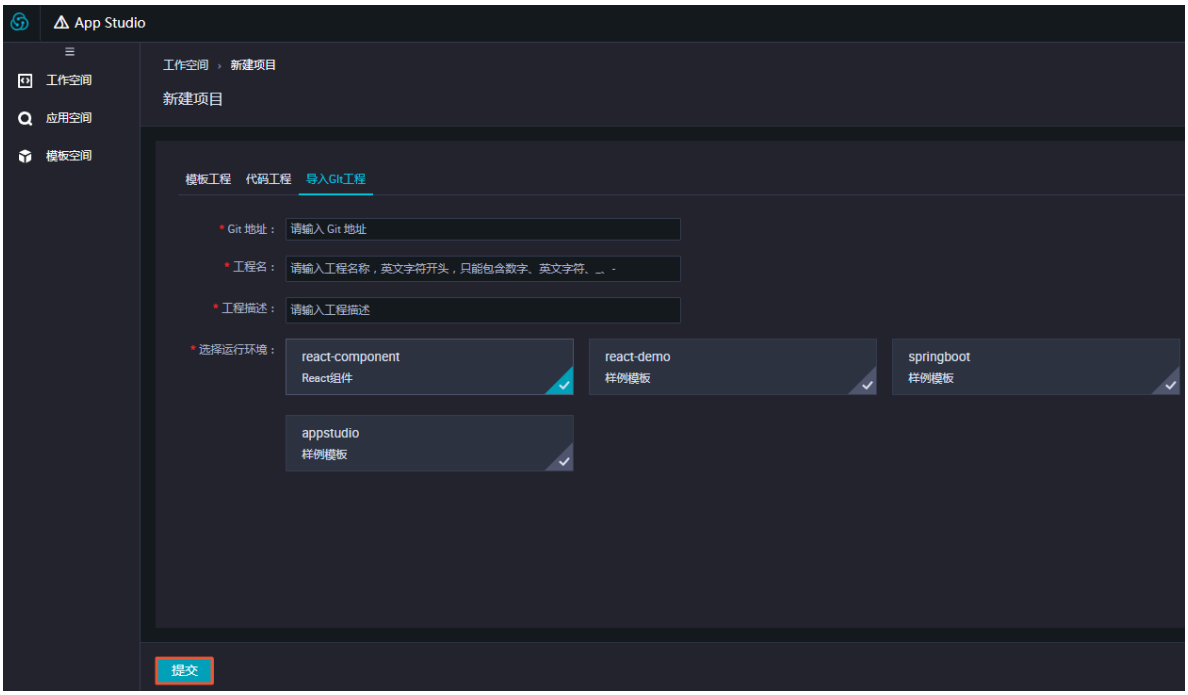


3. 配置完成后，单击提交。

通过Git导入工程

如果您已经有Git代码，可以直接导入Git代码创建工程。此处仅支持*Code*中的Git代码导入。

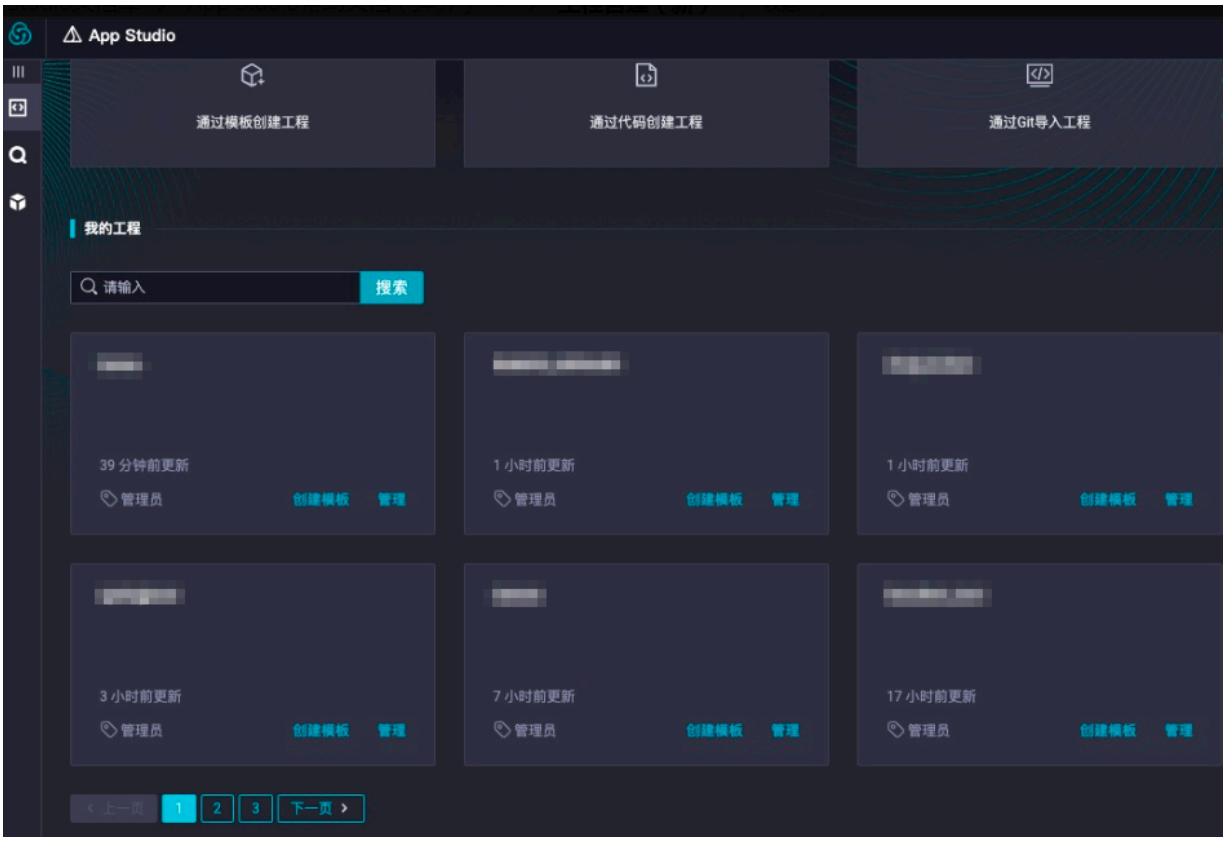
1. 进入App Studio页面，单击工作空间页面的通过Git导入工程。
2. 填写新建项目对话框中的Git地址、工程名和工程描述，选择相应的运行环境。



3. 配置完成后，单击提交。

工程列表

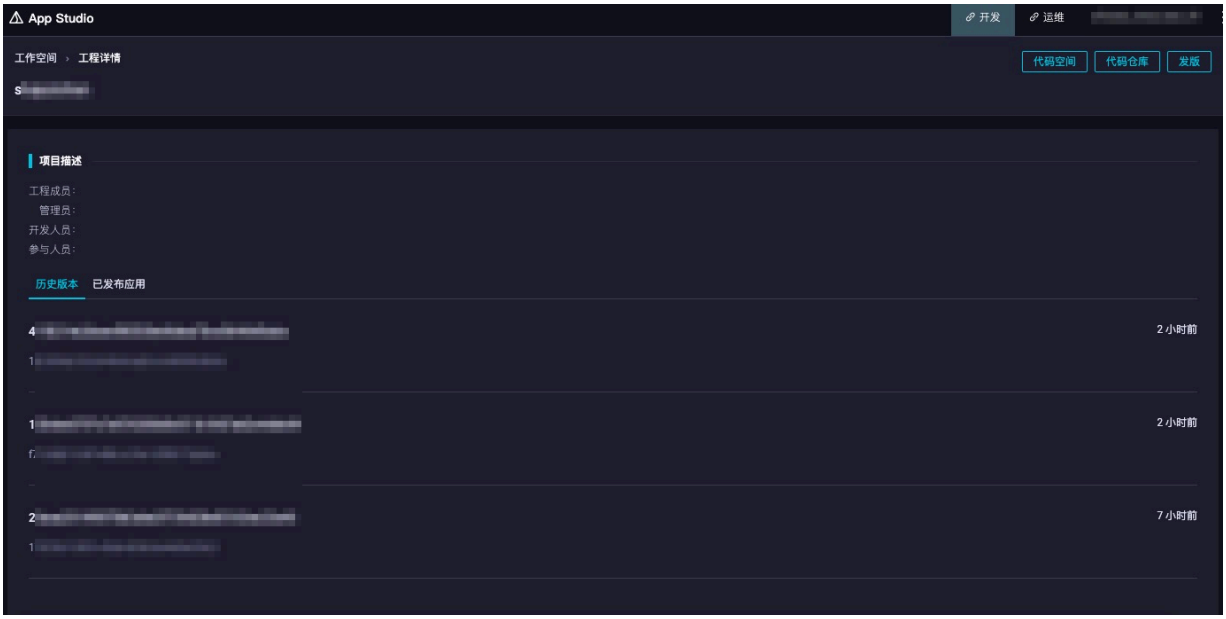
您可以在工作空间页面查看创建的工程。



您可以直接单击相应的工程名称，进入工程编辑页面。也可以单击相应工程下的创建模板，通过工程创建模板。

说明:
如果是其他人分享给您的工程，将不能进行创建模板的操作。

App Studio对工程可以进行部署的版本管理，单击相应工程下的管理，即可进入部署版本管理页面。



您可以对工程进行发版，然后进入应用空间页面，部署相应的工程版本。

说明:

工程需要关联Git才能进行发版。

1.4.3 版本管理

App Studio集成了通用的Git服务，本文将为您介绍在App Studio中如何使用VCS-git。

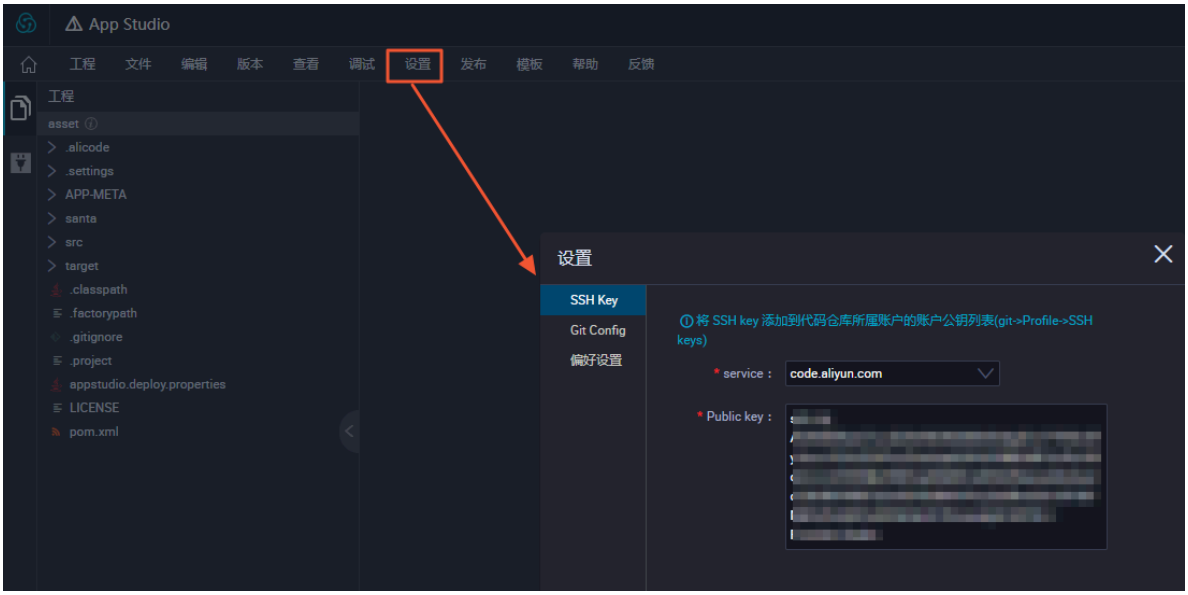
新建工程关联Git系统

1. 新建工程。

2. 录入用户基本信息。

关联Git操作前，需要首先录入用户基本信息。

打开已导入的Git工程，单击菜单栏中的设置，生成一个SSH Key，并根据提示添加到代码仓库所属的账户公钥列表中。



说明:

新创建的工程默认未关联Git服务。如果需要Git服务，请关联当前项目至自己的Git仓库。

3. 新建Git仓库。

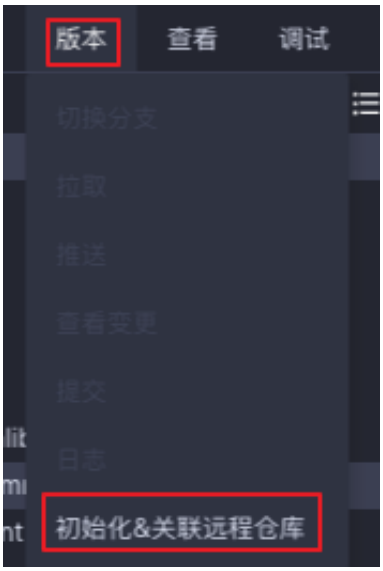
4. 获取当前仓库的SSH地址。

a. 单击SSH，即可获取当前仓库的SSH地址。

b. 单击右侧的复制按钮，即可复制SSH地址至剪贴板。

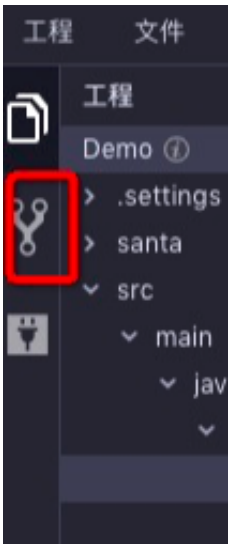
5. 关联Git仓库。

a. 选择菜单栏中的版本 > 初始化&关联远程仓库。



b. 填写关联远程仓库对话框中的Git地址，单击提交。

c. 关联完成后，在App Studio页面的左侧导航栏增加版本控制标签。



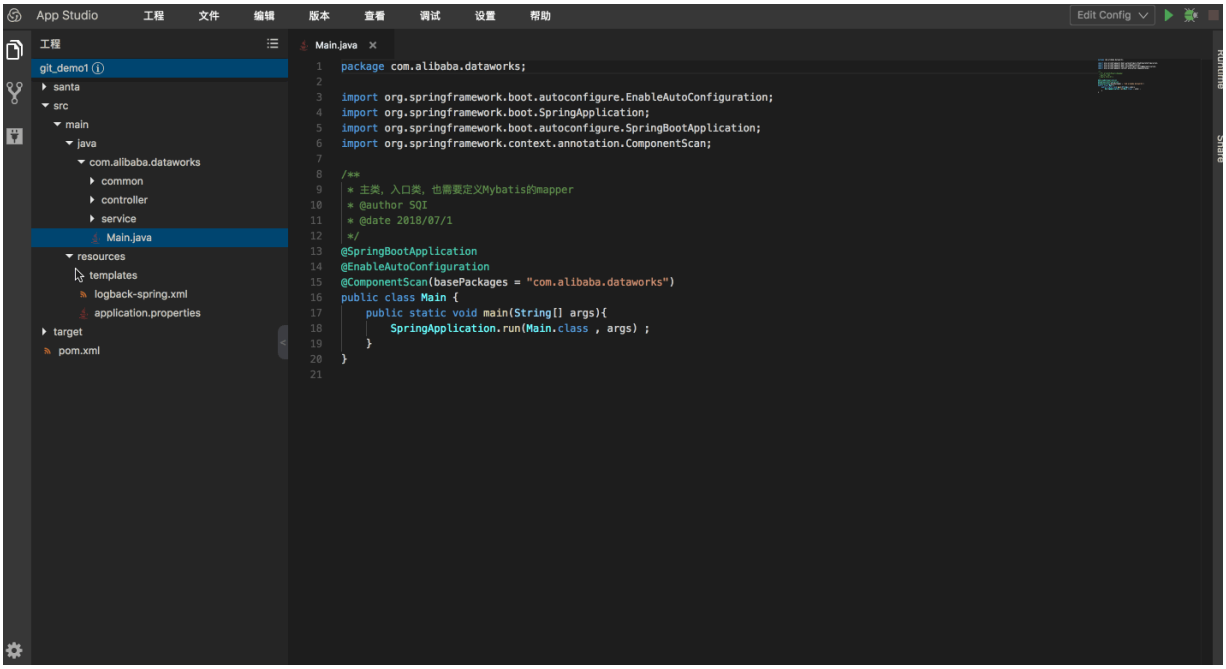
d. 单击版本下拉列表中的推送，即可将本地代码推送至远程仓库。

Git操作入口

Git相关的操作均集成在左侧的Git面板，以及顶端的版本控制菜单栏中。

Git控制面板

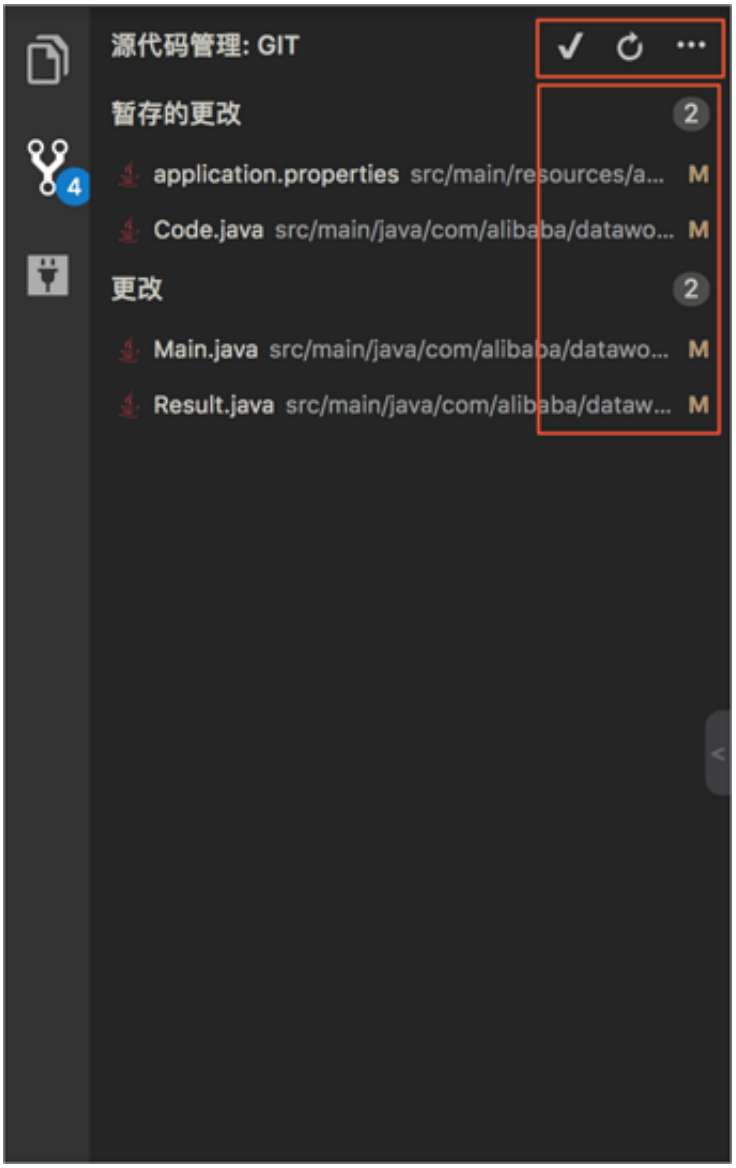
Git控制面板会动态更新文件的编辑状态。



您可以在Git控制面板中，完成基本的`git add/rm/commit/revert`等操作。

Git基本操作

Git面板中以列表形式展示变动的文件，包括文件名、路径、以及右侧支持的基本操作。



如上图所示，红框中包含了可操作按钮，以及文件标识（icon）。

- 源代码管理

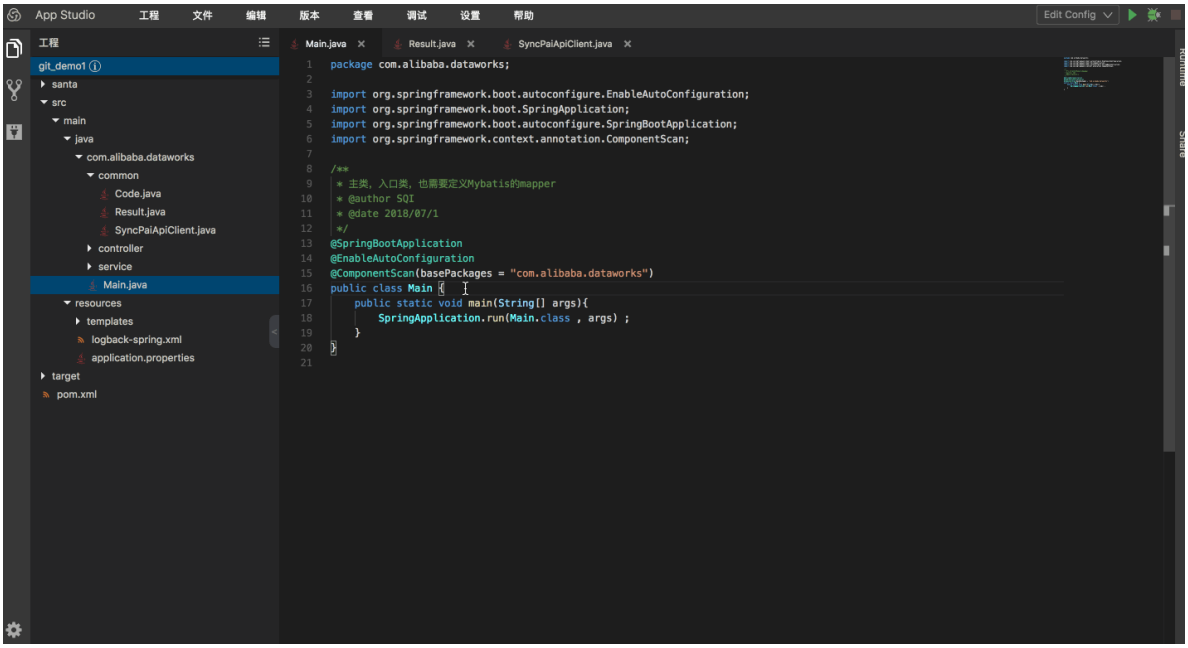
  您可在此处进行commit、refresh、pull和push等操作。

  - commit操作：选择 下的commit&push操作。
  - refresh操作：单击 ，刷新当前控制面板内容，相当于执行`git status`，并刷新界面。
  - pull/push操作：单击 ，根据自身需求选择拉取或推送。

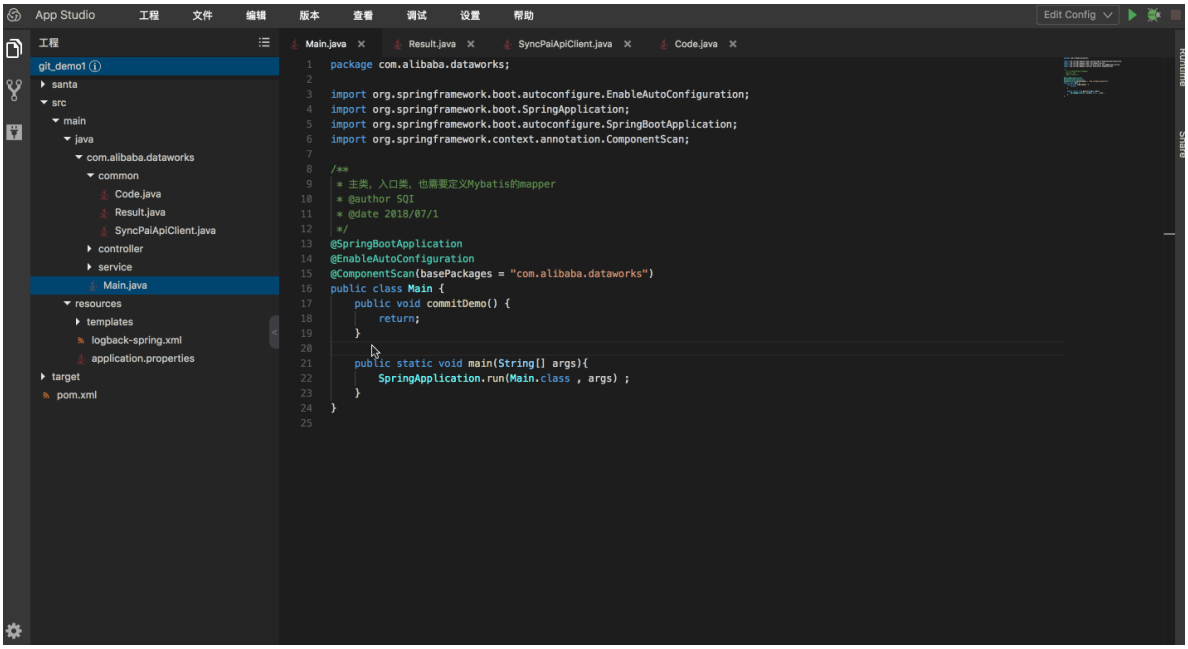
- 暂存的更改
  - ：放弃所有修改，相当于执行`git reset`。
  - ：文件数量标签。
  - ：显示文件更改。
- 更改
  - ：放弃所有更改。
  - ：将所有文件添加至缓存区，相当于执行`git add`。
  - ：文件数量标签。
  - 显示条目包含的操作如下：
    - ：放弃修改（revert）。
    - ：暂存更改（add）。
    - ：文件改动标识（Modified）。

commit/push操作示例

- 示例一



- 示例二

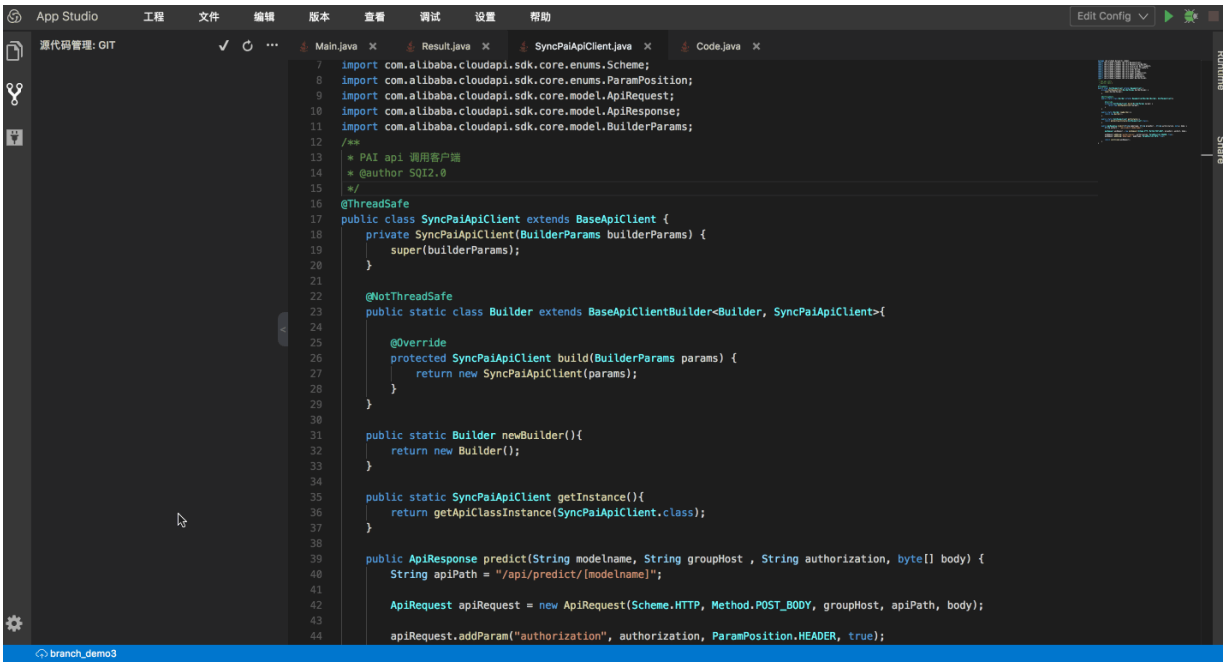


说明:

- Git客户端逻辑一致，您需要主动调用push，本地的代码才会推送至远程仓库。
- 与push同理，您需要主动调用pull，远程仓库的代码才会拉取至本地。

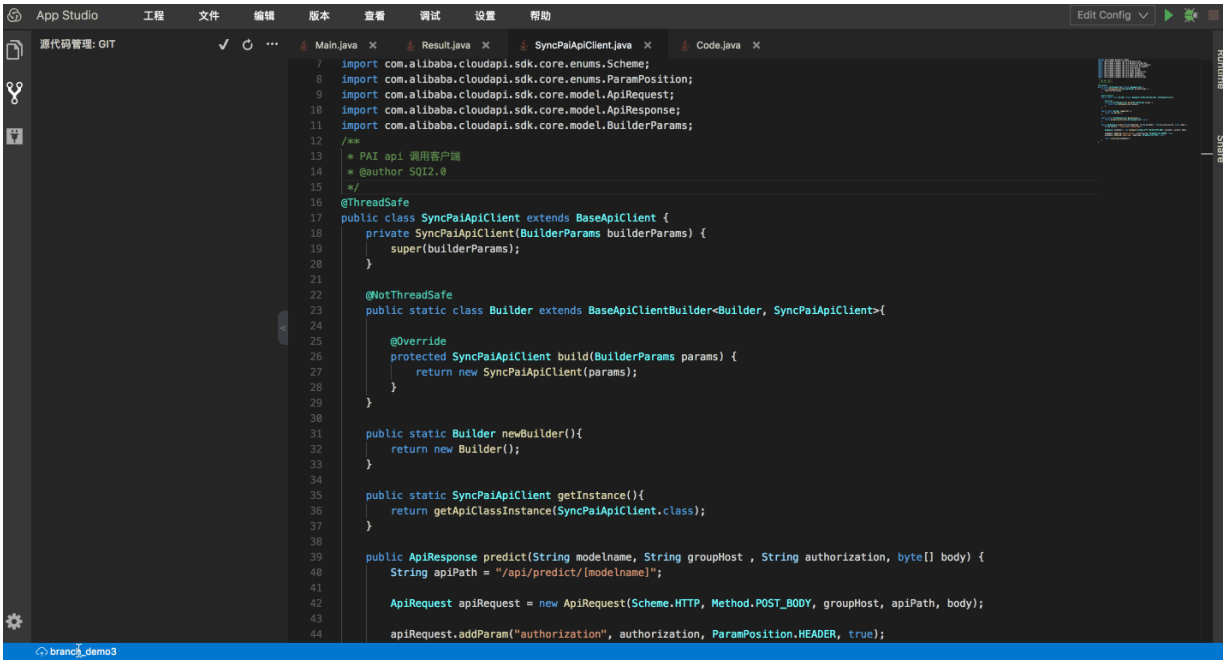
Branch管理

打开分支管理弹窗。



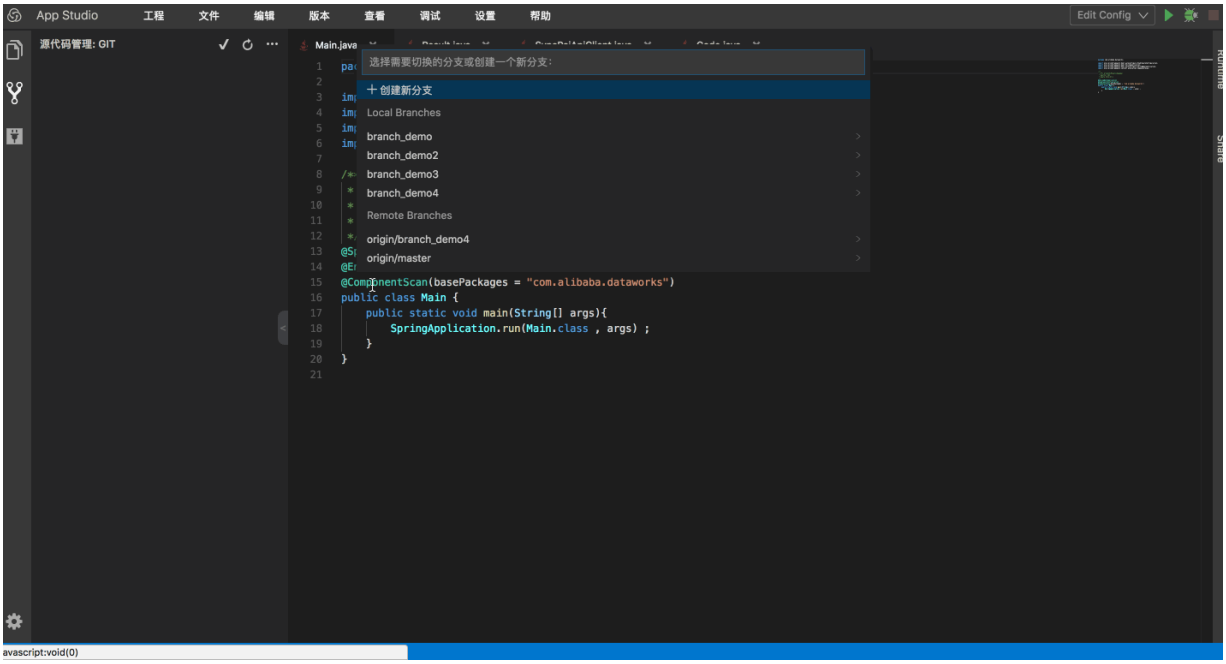
单击窗口下侧状态栏中显示的当前Branch名称，即可弹出Branch管理窗口。

新建本地分支



分支创建后，会自动切换至新创建的分支。

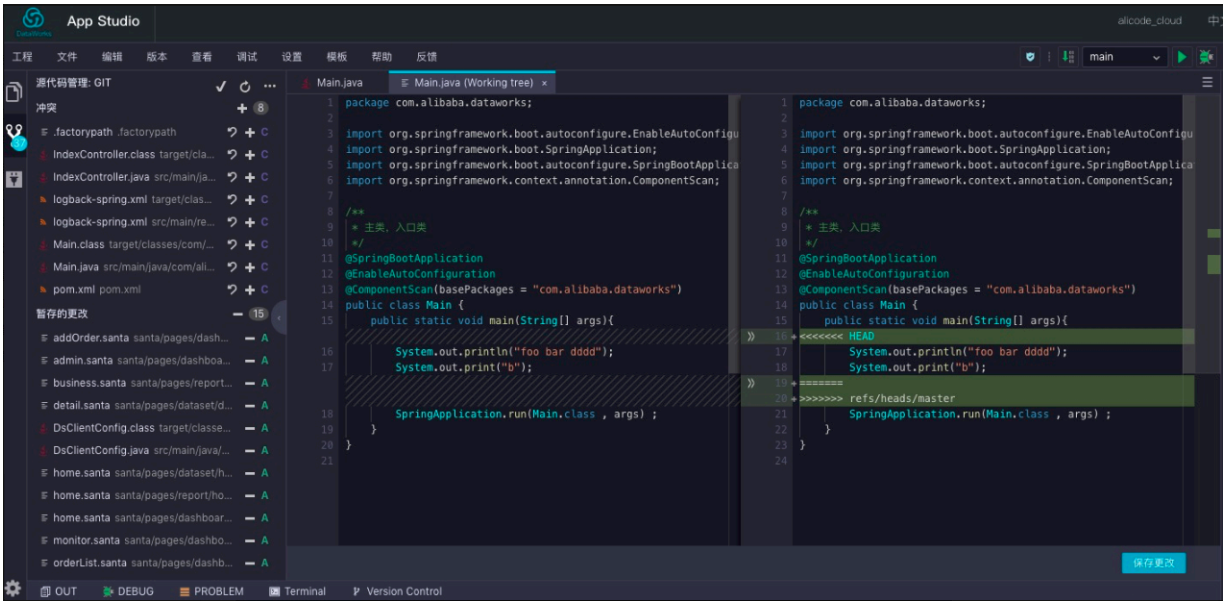
创建/切换/合并分支



说明:

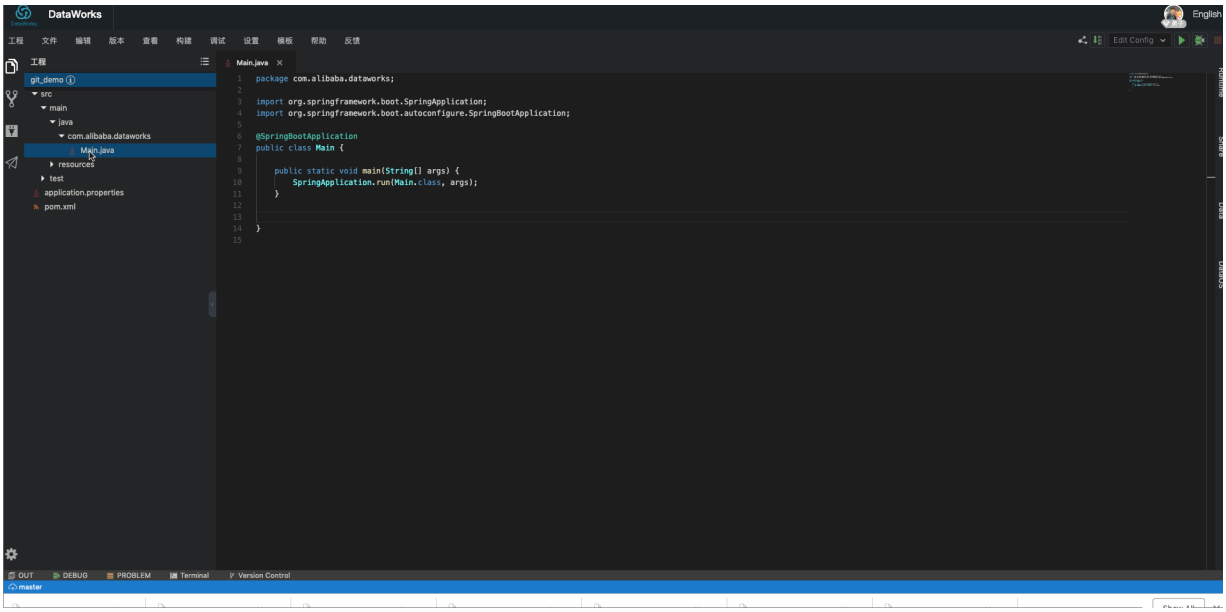
新创建的本地分支，可直接推送至远程，远程分支名与本地一致。

通过Diff页面解决merge conflict



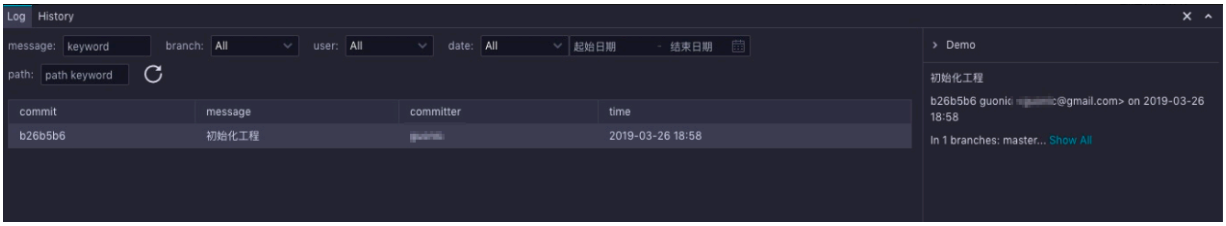
Show History

右键单击文件，选择Git > Show History，即可查看当前文件的历史记录，对特定的commit与当前version进行Diff。



Git Log

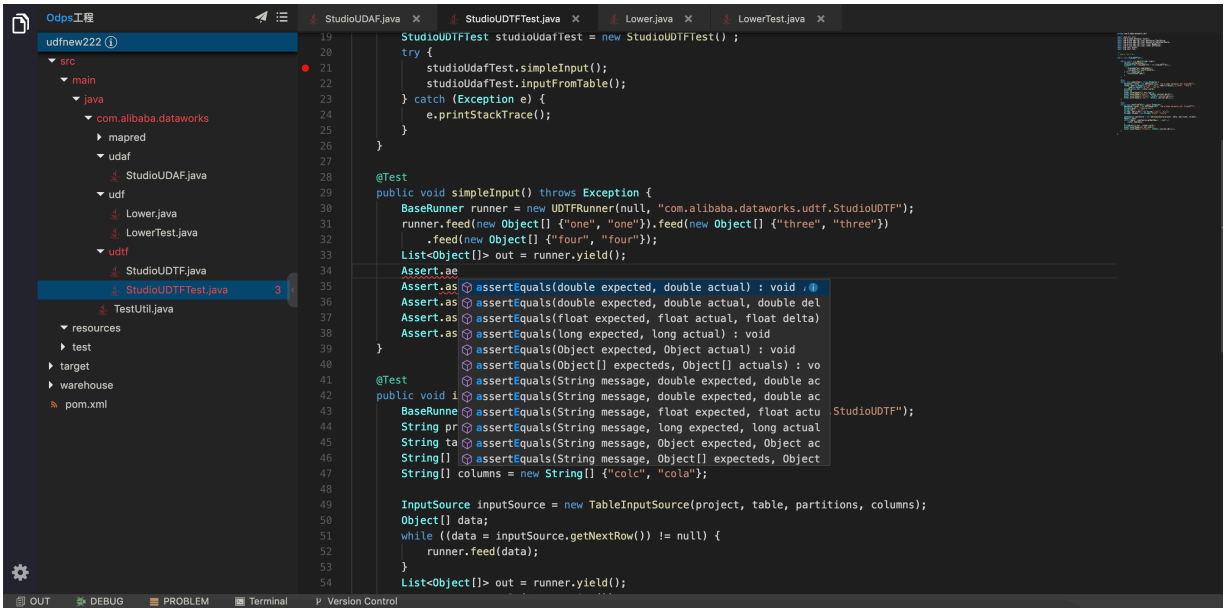
单击菜单栏中的版本 > 日志，打开Git log面板，即可查看提交的信息、时间、作者，您可以通过信息、分支、作者、时间筛选提交日志。



1.4.4 代码编辑

1.4.4.1 代码编辑概述

代码编辑包括自动补全、智能提示、语法诊断和全局内容搜索等常见的IDE具备的功能。



目前语言和对应的功能支持情况，如下表所示。

基本功能	Java	Python	JavaScript/ TypeScript
Completion自动补全	支持	支持	支持
Hover智能提示	支持	支持	支持
Diagnostics语法诊断提示	支持	支持	支持
SignatureHelp函数参数提示	支持	支持	支持
Definition跳转定义	支持	支持	支持
References查找引用	支持	支持	支持
Implementation查找实现类	支持（comming soon）	不支持	不支持

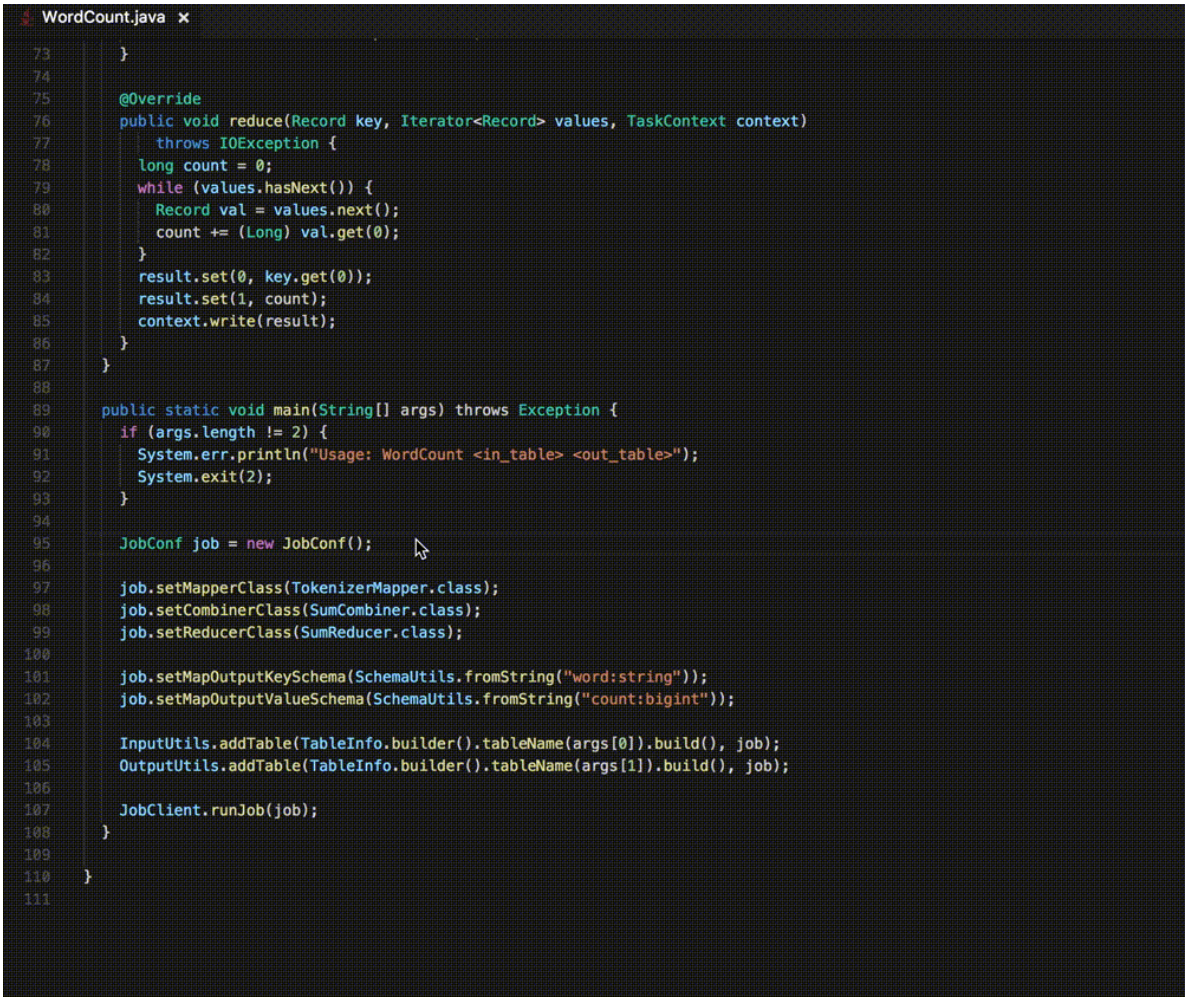
基本功能	Java	Python	JavaScript/ TypeScript
DocumentHighlight变量高亮	支持	支持	支持
DocumentSymbol查找类成员	支持	支持	支持
WorkspaceSymbol全局查找类/函数	支持	支持	支持
CodeAction修复建议	支持（Alibaba Java Guidelines is coming soon）	支持	支持
CodeLens行操作提示	References Implementation	不支持	不支持
Formatting 格式化代码	支持	支持	不支持
RangeFormatting局部格式化	支持	不支持	不支持
FindInPath全局内容搜索	支持	支持	支持

高级功能	Java	Python	JavaScript/ TypeScript
Rename重命名	支持	支持	支持
WorkspaceEdit多文件修改	支持	不支持	不支持
UnitTest单元测试（quickstart）	支持	不支持	不支持
MainClass查找main函数入口	支持	不支持	不支持
MainClassQuickStart快捷运行	不支持	不支持	不支持
ListModules查找所有模块	支持	不支持	不支持

高级功能	Java	Python	JavaScript/ TypeScript
Generate生成代码片段	Constructor Override Getter/Setter Implement	不支持	不支持

基本功能

- 自动补全



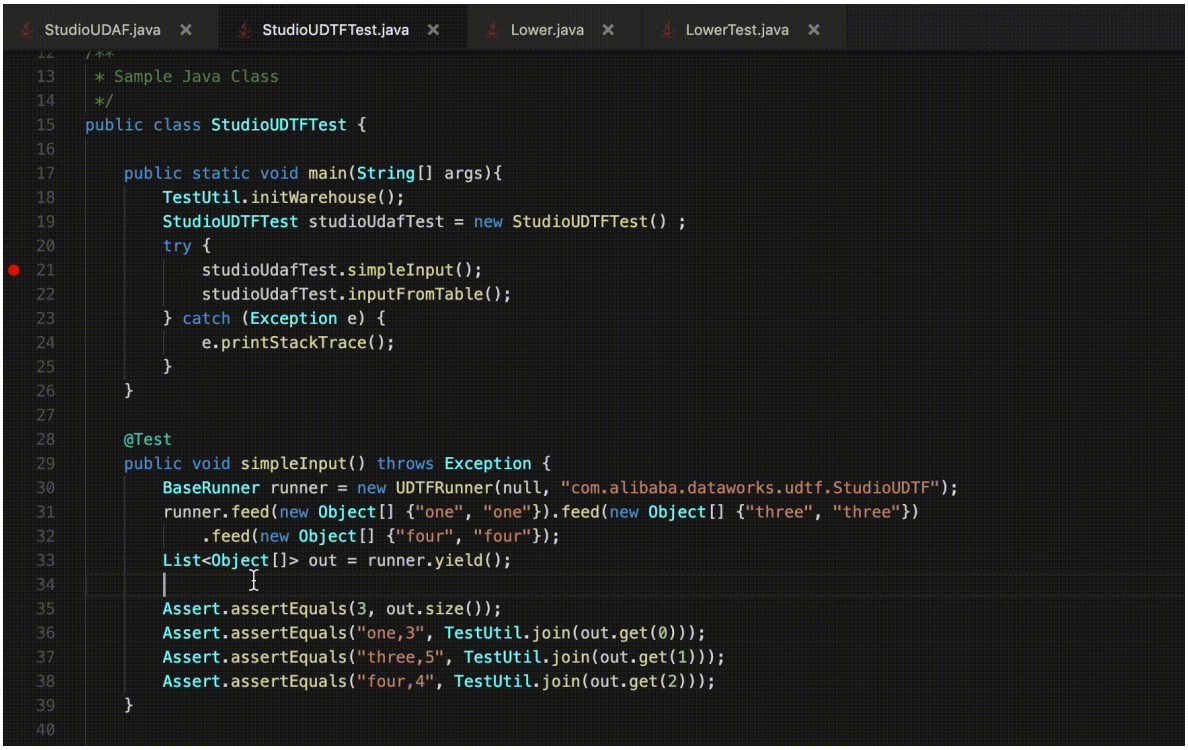
- 智能提示

```
@Resolve({"double->double"})
public class AggrAvg extends Aggregator {
    private static class AvgBuffer implements Writable {
        private double sum = 0;
        private long count = 0;
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeDouble(sum);
            out.writeLong(count);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            sum = in.readDouble();
            count = in.readLong();
        }
    }
    private DoubleWritable ret = new DoubleWritable();
    @Override
    public Writable newBuffer() {
        return new AvgBuffer();
    }
    @Override
    public void iterate(Writable buffer, Writable[] args) throws UDFException {
        DoubleWritable arg = (DoubleWritable) args[0];
        AvgBuffer buf = (AvgBuffer) buffer;
        if (arg != null) {
            buf.count += 1;
            buf.sum += arg.get();
            if (buf.count > 9) {
                throw new IllegalStateException("只能计算10个数");
            }
        }
    }
```

- 语法诊断

```
@Resolve({"double->double"})
public class AggrAvg extends Aggregator {
    private static class AvgBuffer implements Writable {
        private double sum = 0;
        private long count = 0;
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeDouble(sum);
            out.writeLong(count)
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            sum = in.readDouble();
            count = in.readLong();
        }
    }
    private DoubleWritable ret = new DoubleWritable();
    @Override
    public Writable newBuffer() {
        return new AvgBuffer();
    }
    @Override
```

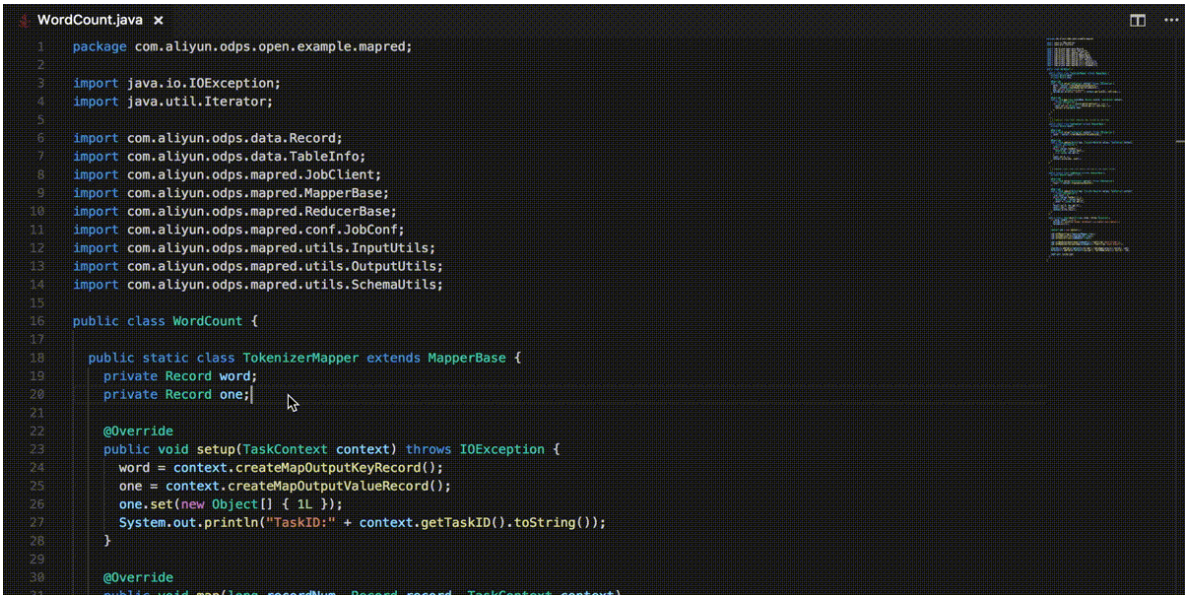
- 函数参数提示



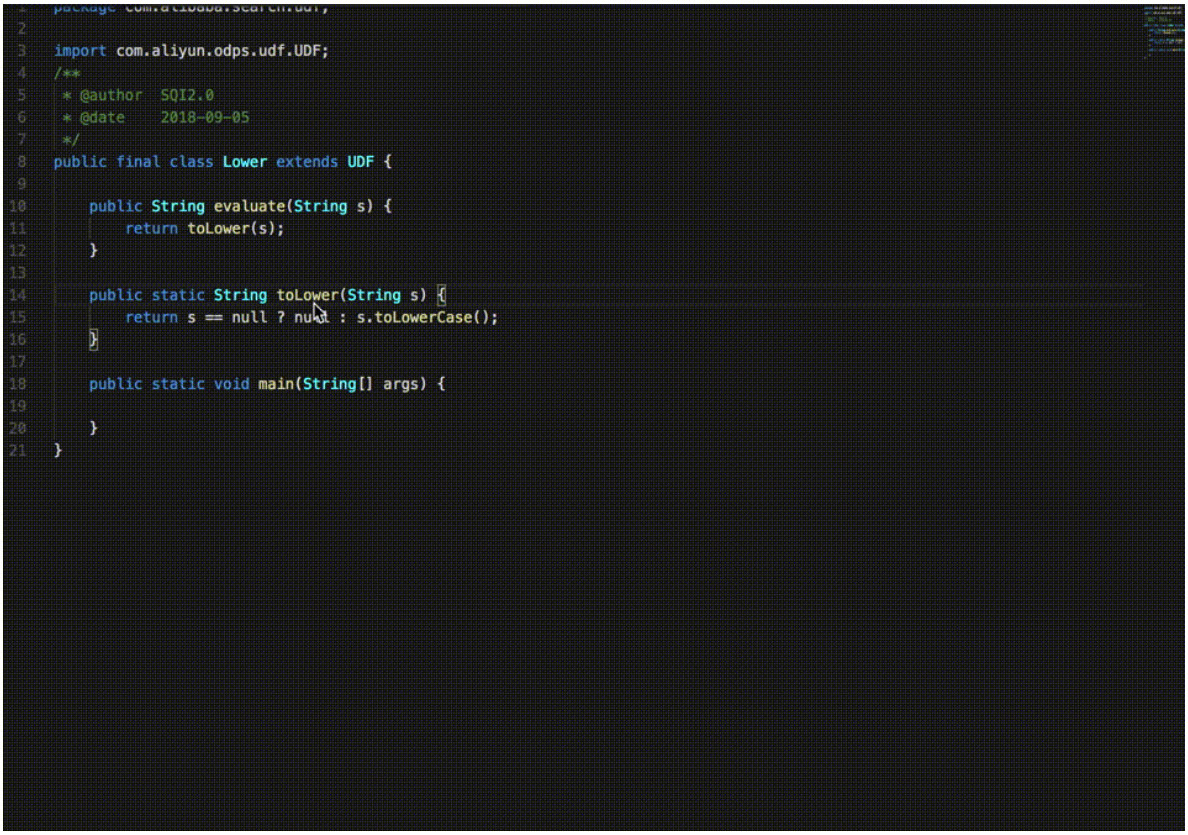
StudioUDAF.java | StudioUDTFTest.java | Lower.java | LowerTest.java

```
/**
 * Sample Java Class
 */
public class StudioUDTFTest {

    public static void main(String[] args){
        TestUtil.initWarehouse();
        StudioUDTFTest studioUdafTest = new StudioUDTFTest() ;
        try {
            studioUdafTest.simpleInput();
            studioUdafTest.inputFromTable();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Test
    public void simpleInput() throws Exception {
        BaseRunner runner = new UDTFRunner(null, "com.alibaba.dataworks.udtf.StudioUDTF");
        runner.feed(new Object[] {"one", "one"}).feed(new Object[] {"three", "three"})
            .feed(new Object[] {"four", "four"});
        List<Object[]> out = runner.yield();

        Assert.assertEquals(3, out.size());
        Assert.assertEquals("one,3", TestUtil.join(out.get(0)));
        Assert.assertEquals("three,5", TestUtil.join(out.get(1)));
        Assert.assertEquals("four,4", TestUtil.join(out.get(2)));
    }

```

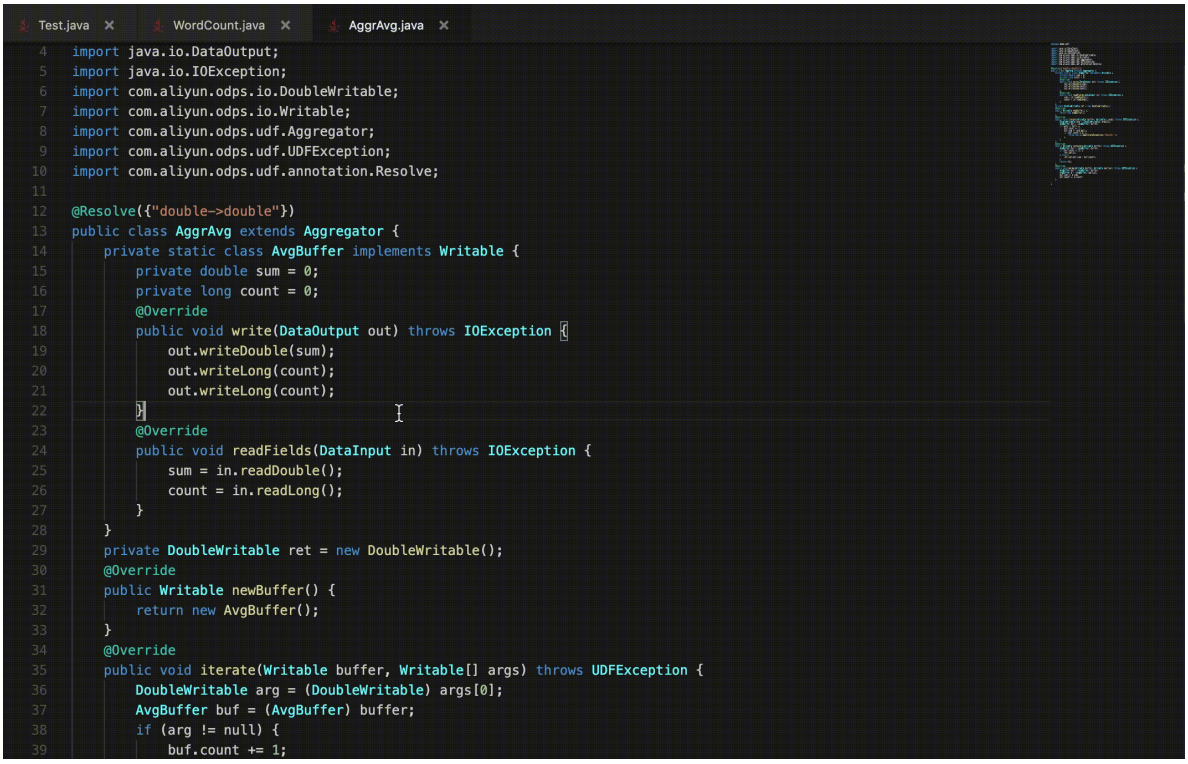
- 跳转定义


WordCount.java
package com.aliyun.odps.open.example.mapred;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
public class WordCount {
public static class TokenizerMapper extends MapperBase {
private Record word;
private Record one;
@Override
public void setup(TaskContext context) throws IOException {
word = context.createMapOutputKeyRecord();
one = context.createMapOutputValueRecord();
one.set(new Object[] { 1L });
System.out.println("TaskID:" + context.getTaskID().toString());
}
@Override

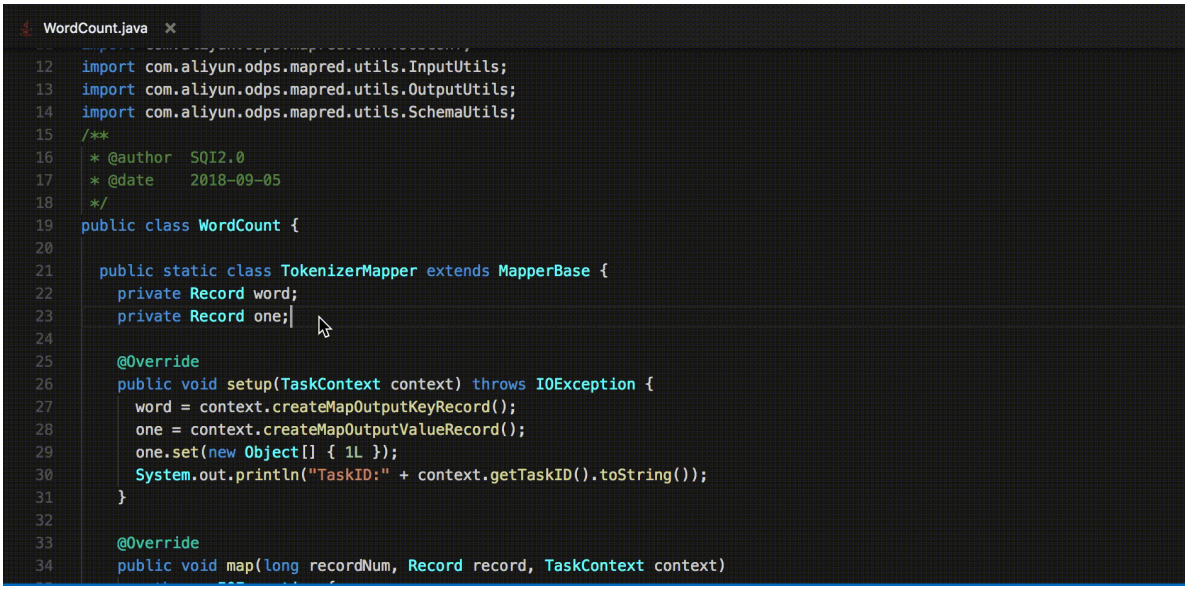
- 查找引用


import com.aliyun.odps.udf.UDF;
/**
* @author SQI2.0
* @date 2018-09-05
*/
public final class Lower extends UDF {
public String evaluate(String s) {
return toLower(s);
}
public static String toLower(String s) {
return s == null ? null : s.toLowerCase();
}
public static void main(String[] args) {
}
}

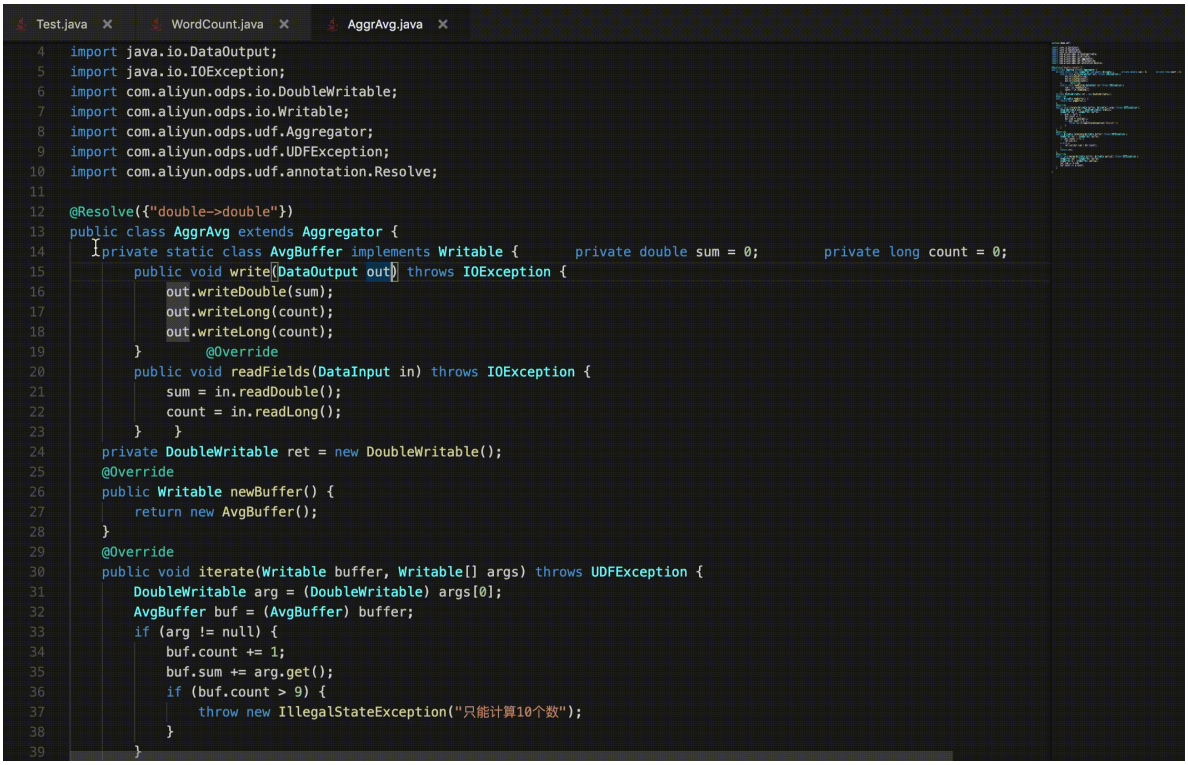
- 查找当前类成员



- 全局查找类/函数



- **代码格式化**

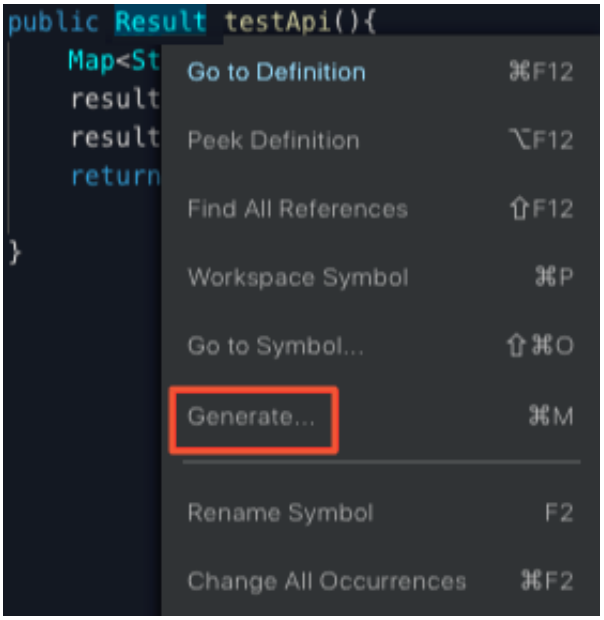


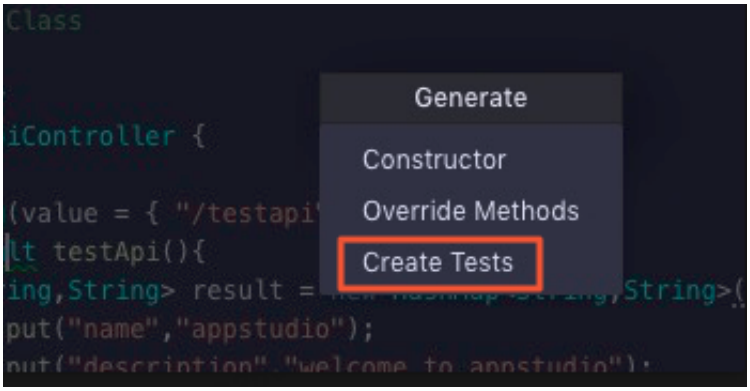
1.4.4.2 UT测试

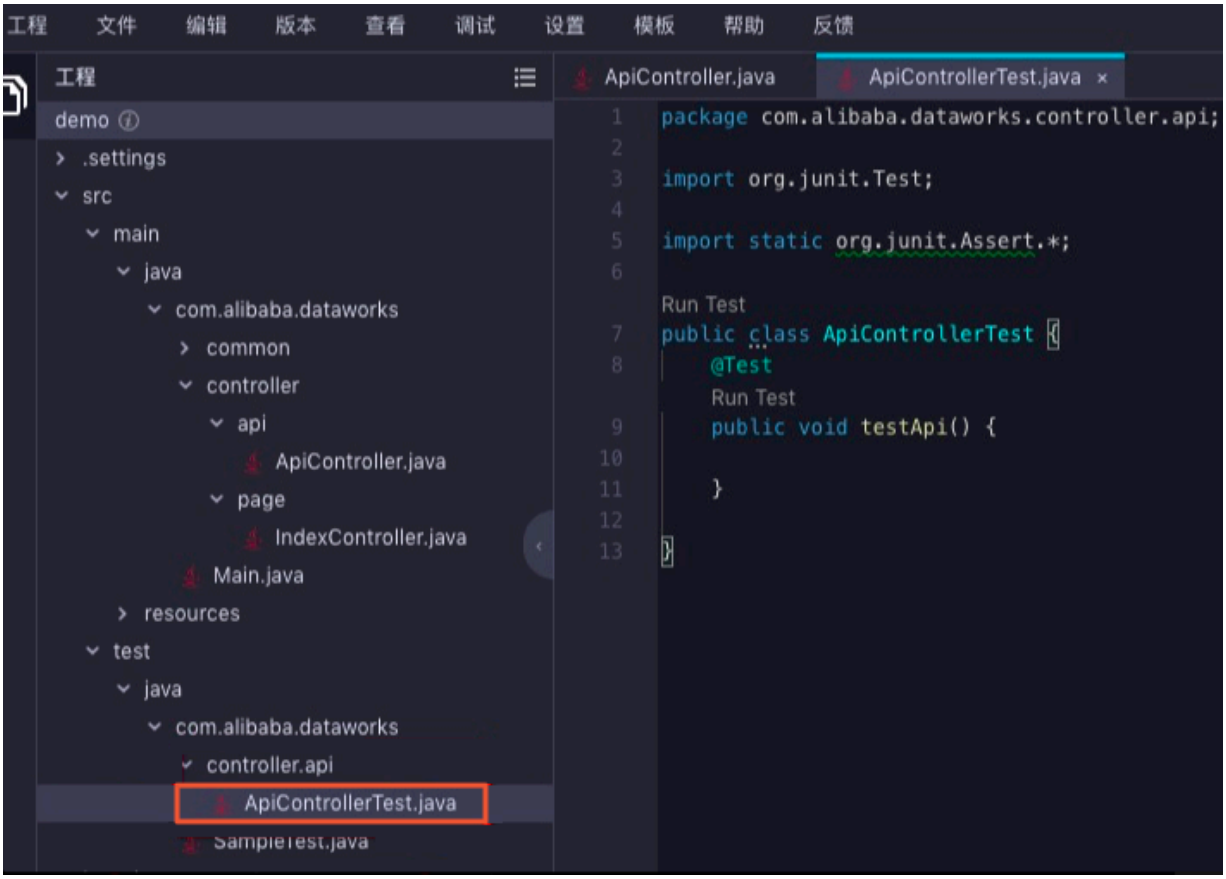
App Studio目前支持自动生成UT代码、检测UT测试入口、运行UT代码和展示运行结果等功能。

自动生成UT代码

打开相应文件后，右键单击代码编辑区，选择Generate > Create Test，即可在Test目录下自动生成测试类和测试代码。





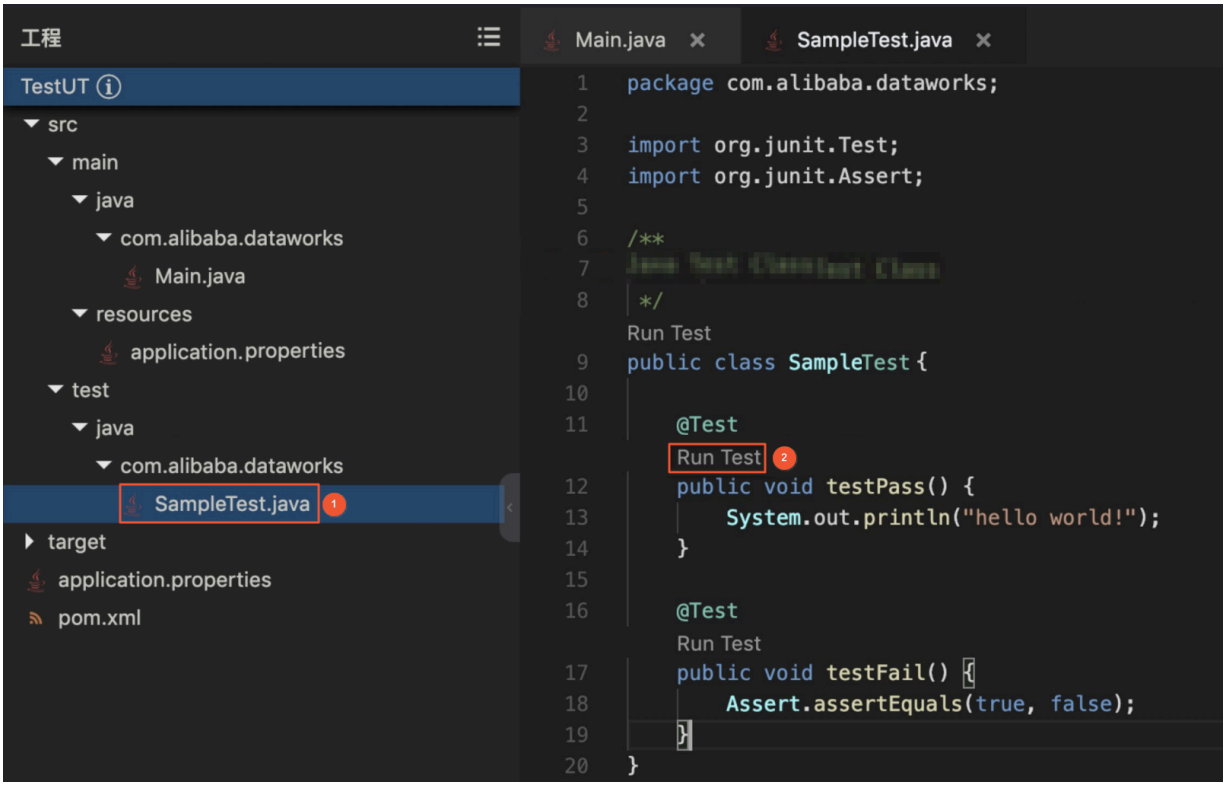


检测UT测试入口

说明:

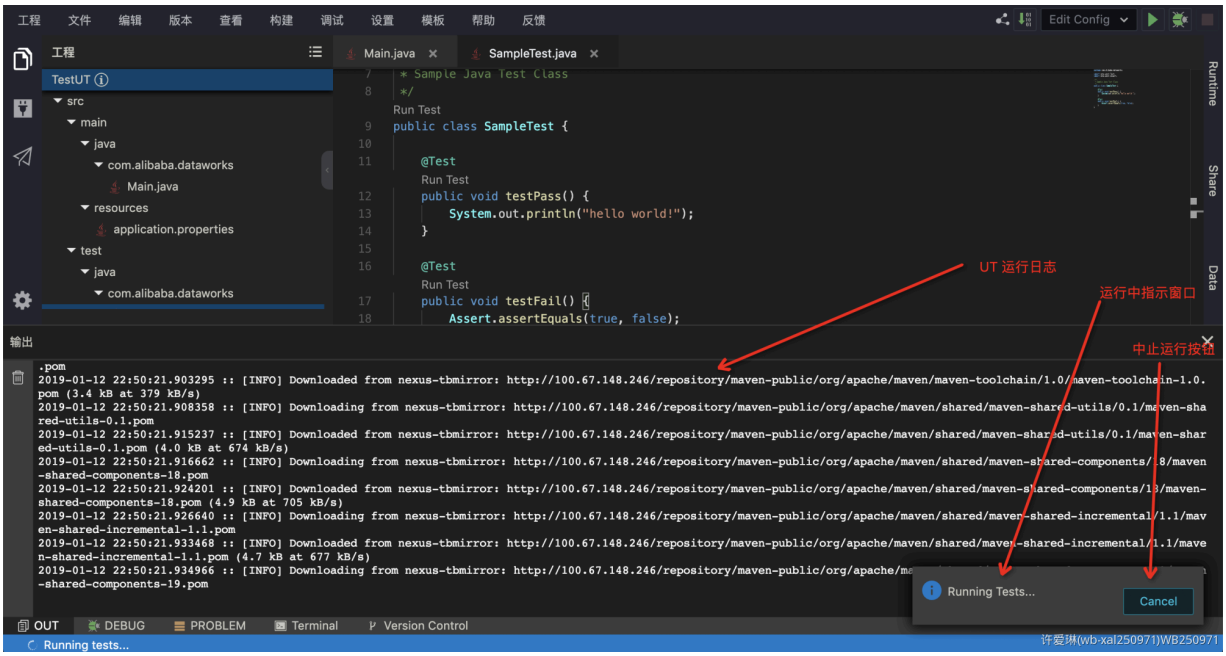
- **UT类需要写在*src/test/java*目录下。如果Java UT类文件不在该目录下，将无法被正常识别成Java UT类。**
- **@Test注解下的方法会出现Run Test的UT运行入口。**

完成Java类的创建后，在对应的测试用例方法上添加`org.junit.Test`的@Test注解即可。

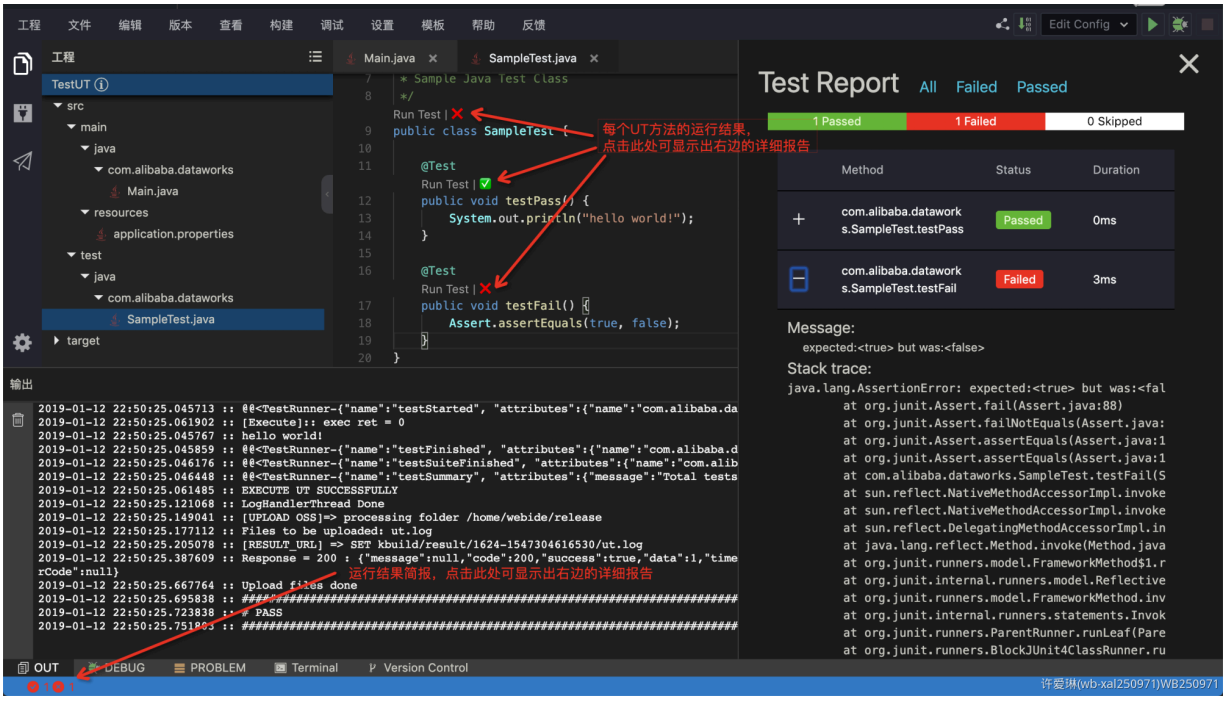


运行UT代码

单击右上角的运行按钮，即可运行UT的测试用例。



展示UT运行结果



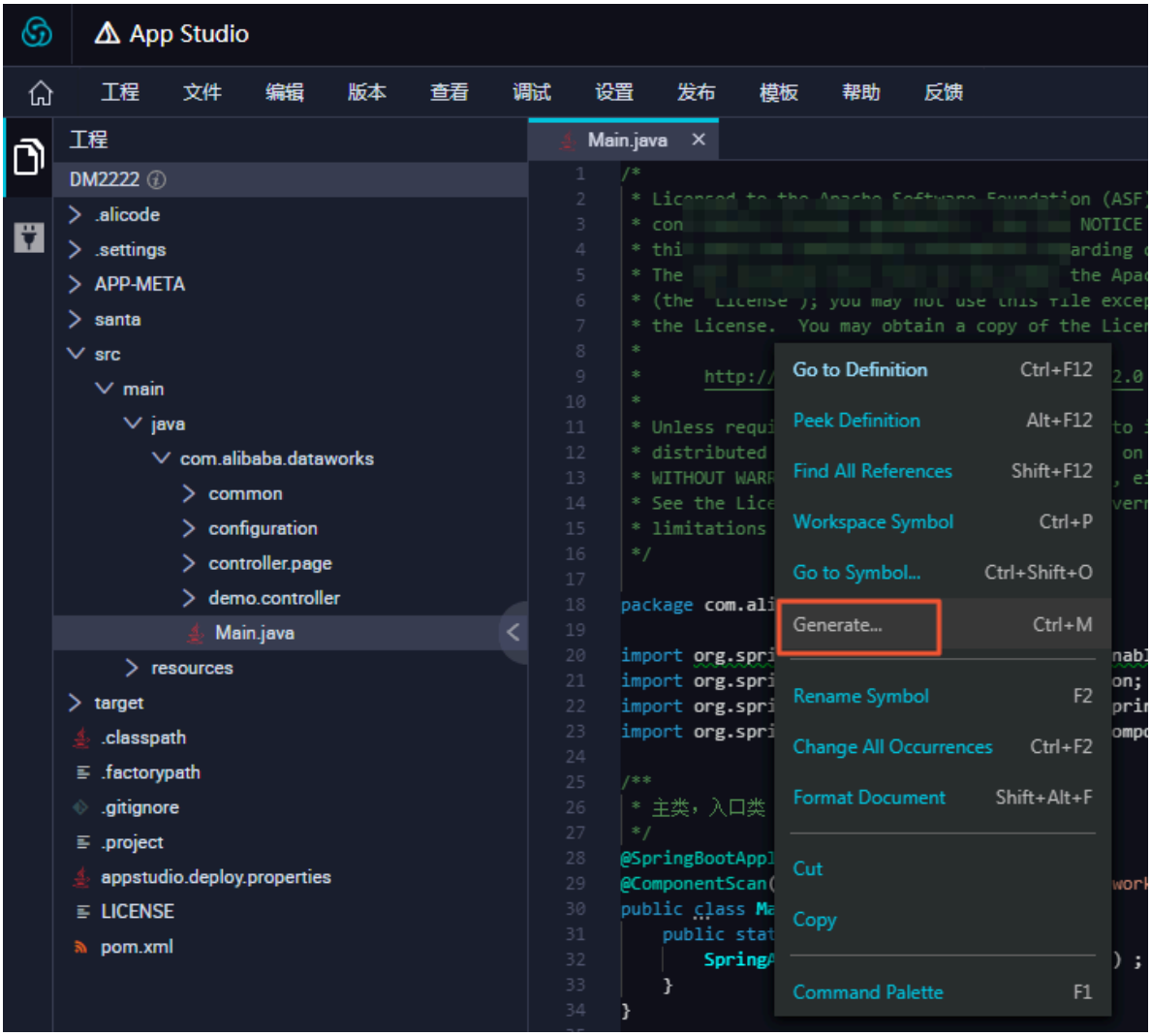
1.4.4.3 生成代码片段

目前App Studio在Java语言中，支持生成Java类的构造函数（Constructor）、Getter函数、Setter函数，也支持生成该类所继承父类的Override方法、所要实现的接口方法等。

功能入口

目前Java的代码生成入口有以下两种：

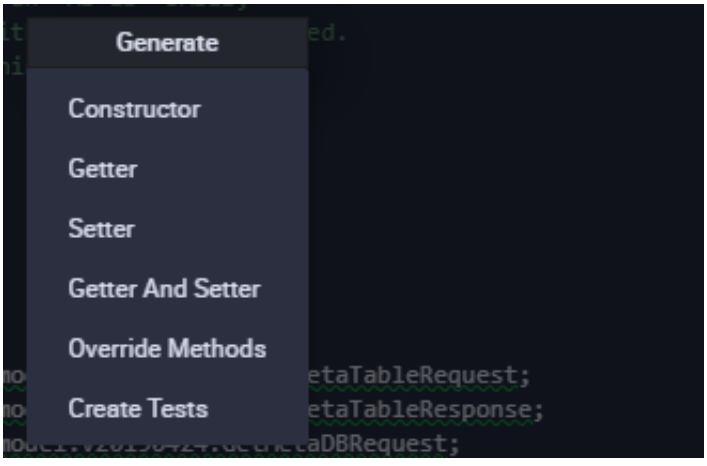
- 鼠标右键单击代码区域，选择Generate。



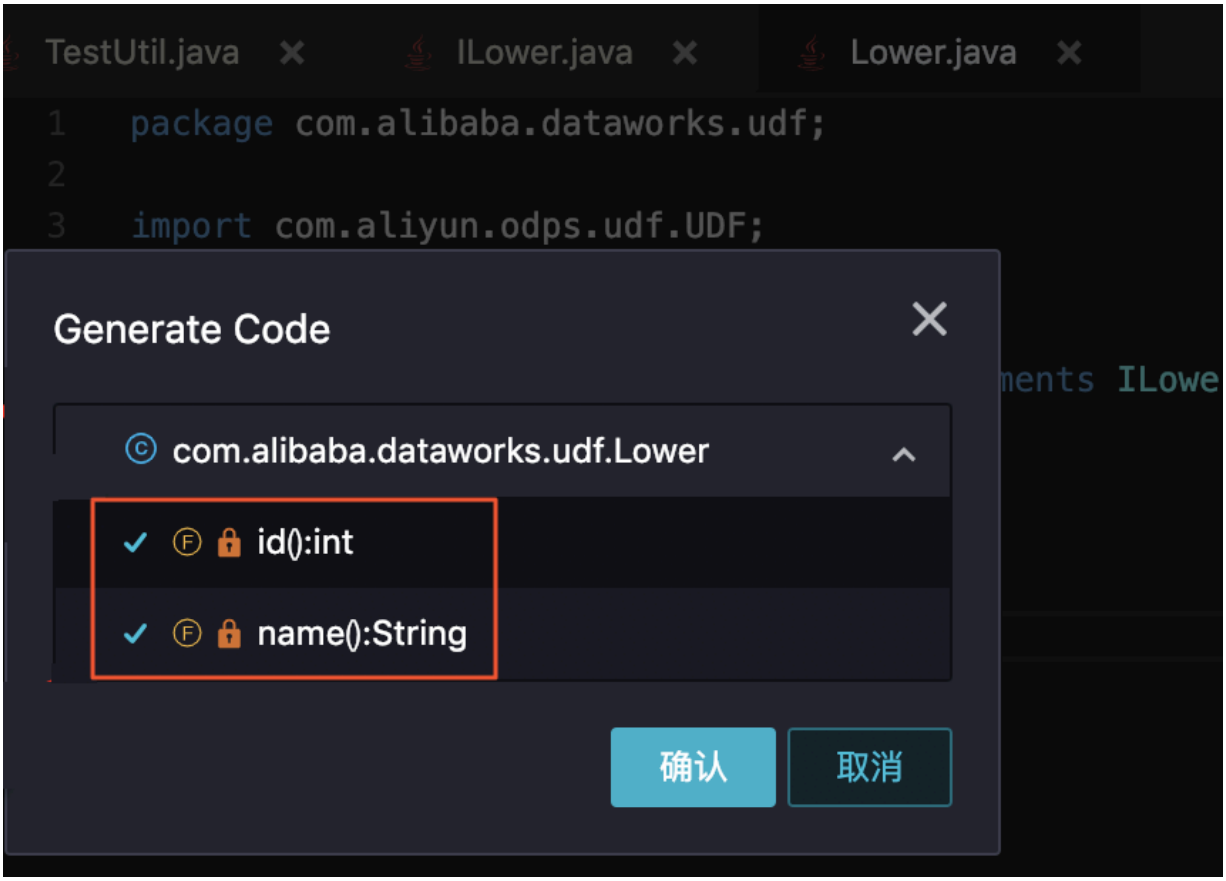
- 通过快捷键Ctrl+M进入。

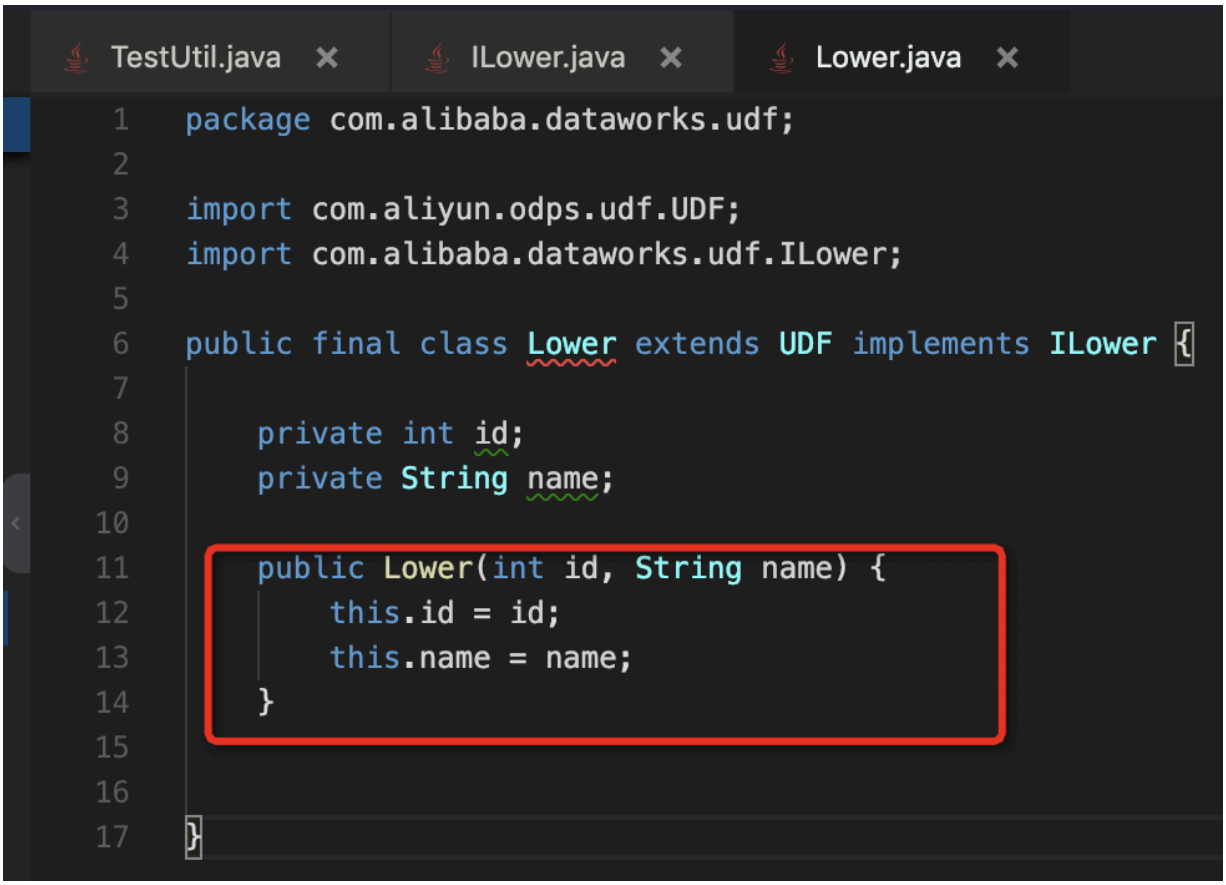
Constructor

进入Generate Code面板后，选择Constructor。



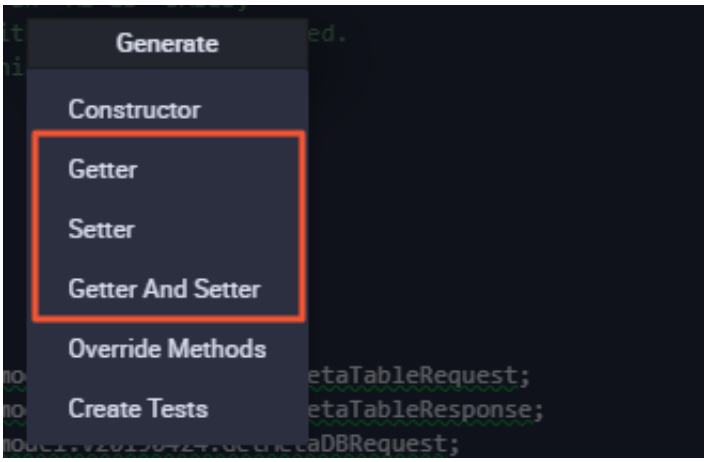
选择构造函数中要包含的字段，即可生成包含相应字段初始化语句的构造函数。


TestUtil.java
ILower.java
Lower.java
1 package com.alibaba.dataworks.udf;
2
3 import com.aliyun.odps.udf.UDF;
Generate Code
com.alibaba.dataworks.udf.Lower
id():int
name():String
确认
取消


TestUtil.java
ILower.java
Lower.java
1 package com.alibaba.dataworks.udf;
2
3 import com.aliyun.odps.udf.UDF;
4 import com.alibaba.dataworks.udf.ILower;
5
6 public final class Lower extends UDF implements ILower {
7
8 private int id;
9 private String name;
10
11 public Lower(int id, String name) {
12 this.id = id;
13 this.name = name;
14 }
15
16
17 }

Getter Add Setter

您可以参见Constructor的生成方式，来生成Getter和Setter函数。

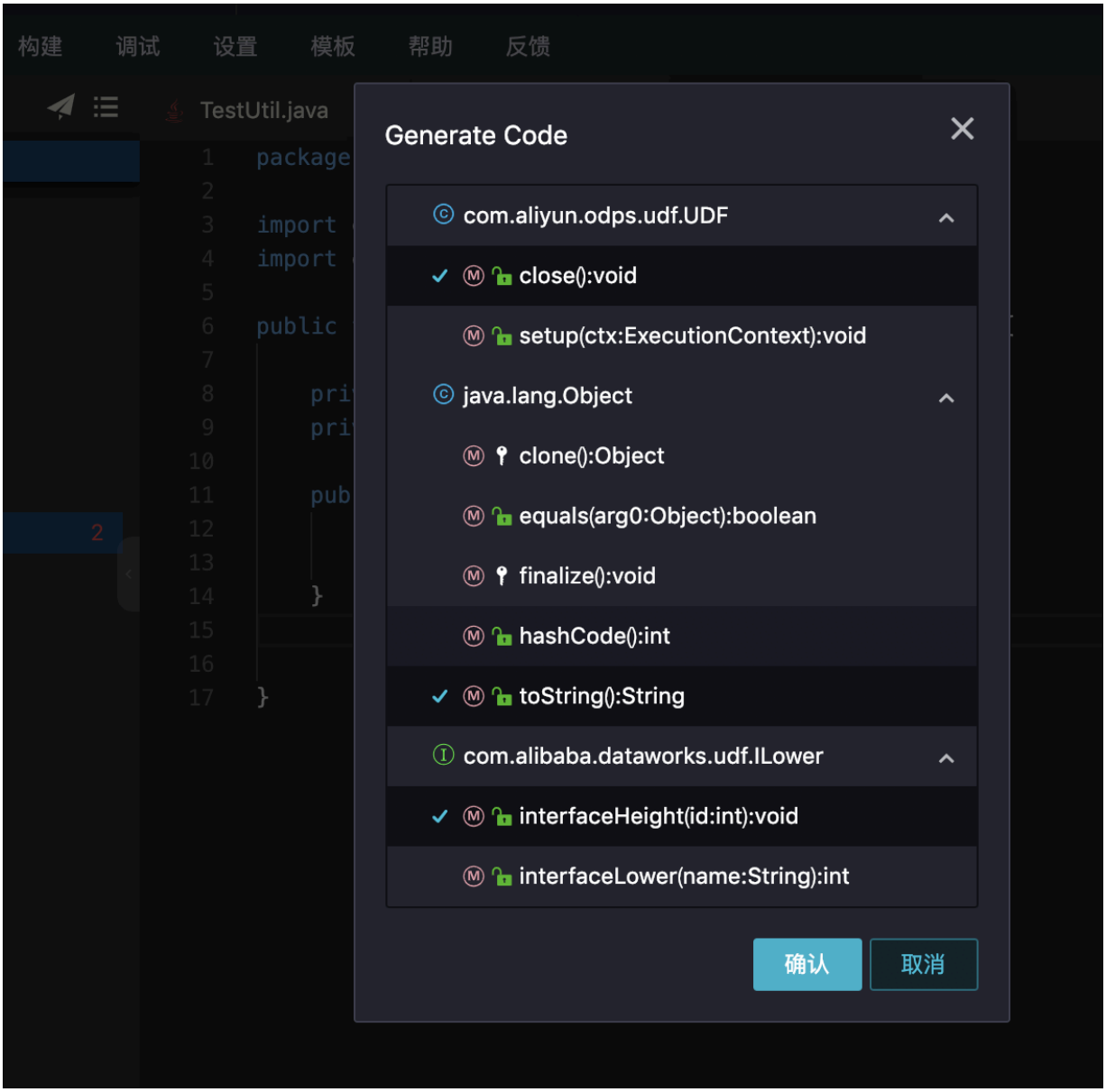


说明:

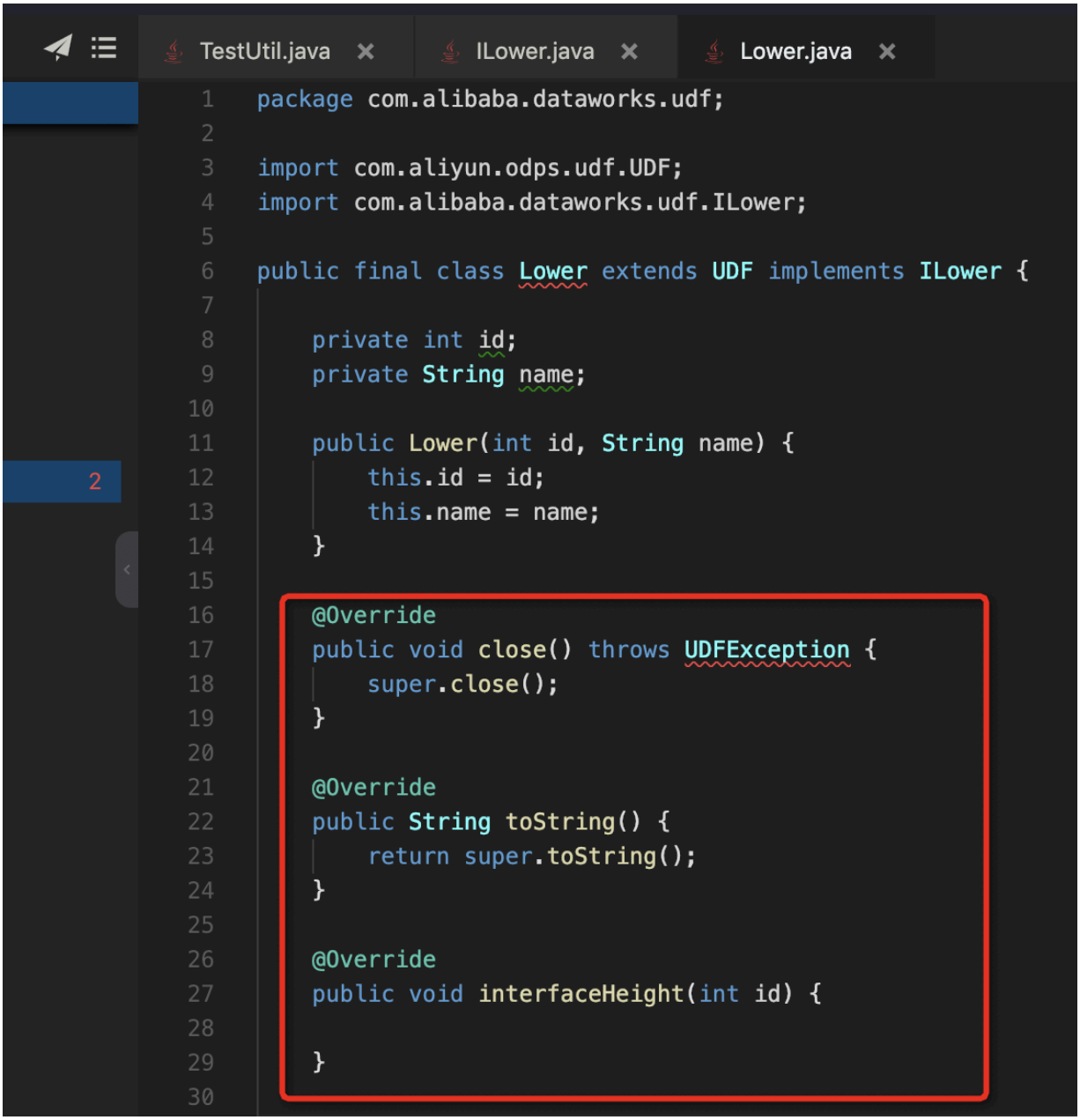
如果该Java类没有任何字段，或者该Java类已经被lombok的@data注解覆盖，则没有图中的三个选项，因为此时该类不需要生成Getter或Setter函数。

Override Methods

选择生成Override Methods的一级菜单后，在二级菜单中会罗列所有可以Override的方法。



选择后即可生成相应的方法。



Create Test

进入Generate Code面板后，选择Create Test，即可在Test目录下自动生成测试类和测试代码。详情请参见*UT*测试。

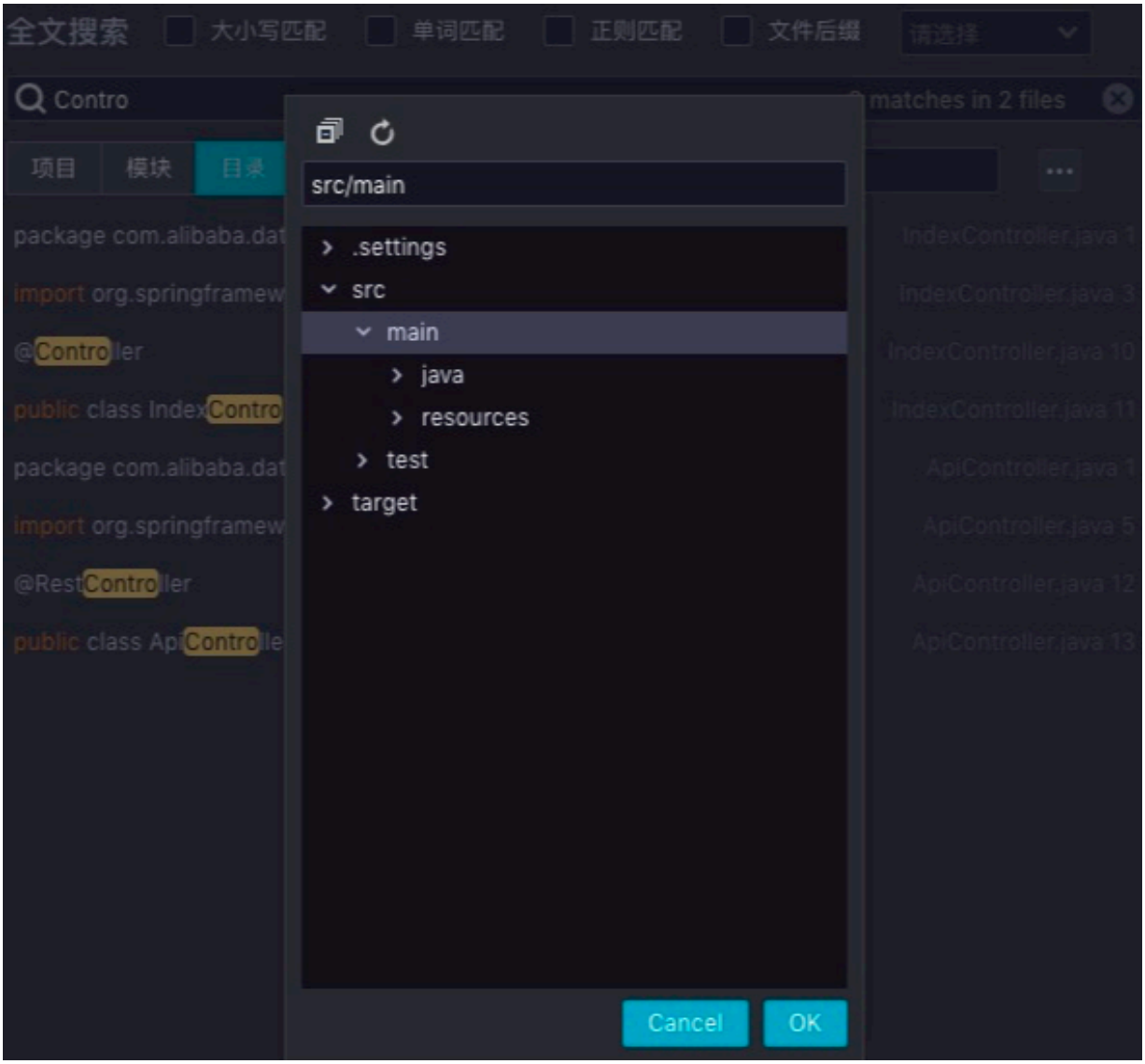
1.4.4.4 全文内容搜索

App Studio支持全文内容搜索功能。

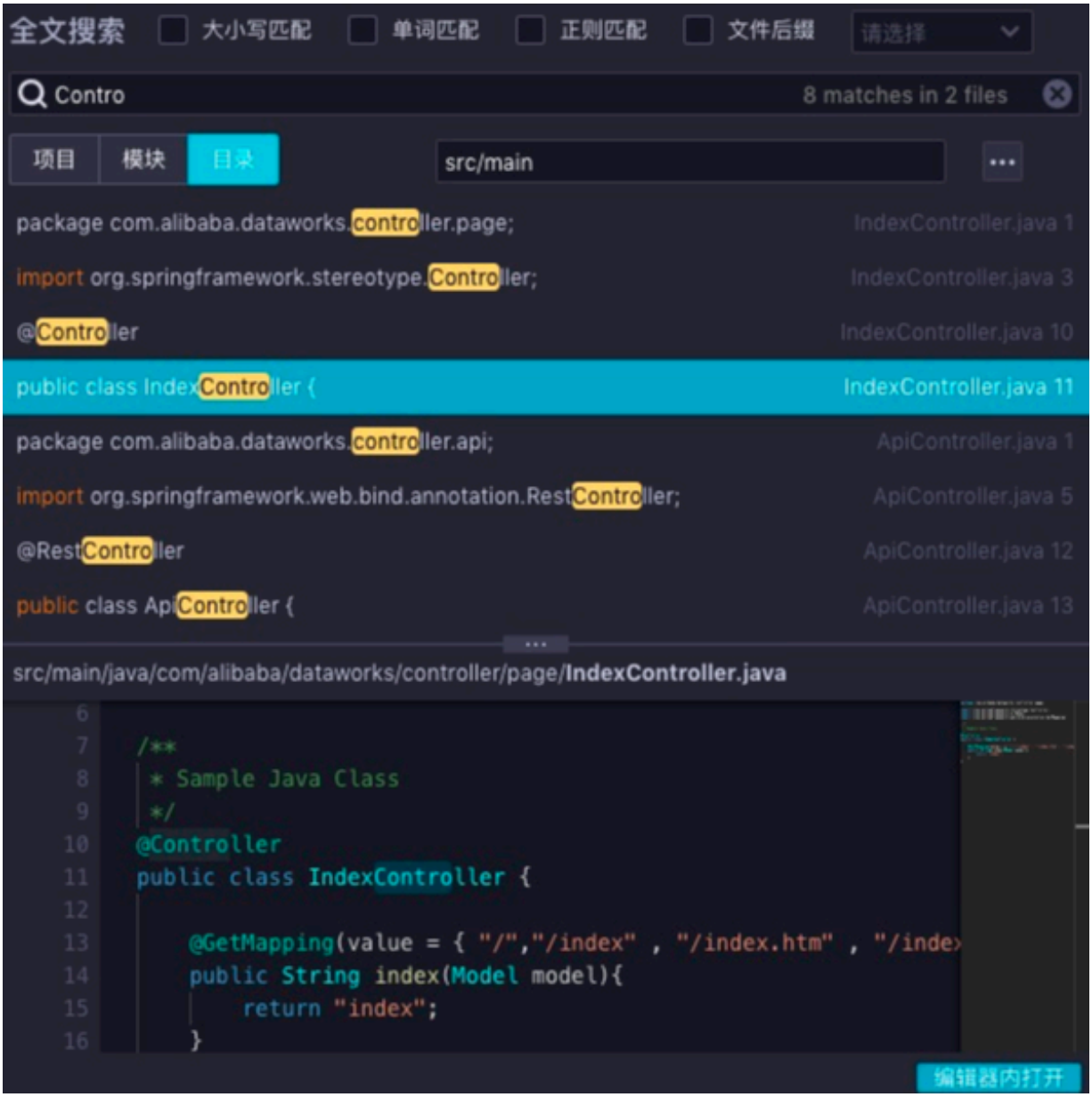
选择菜单栏中的编辑 > 全文搜索。

支持输入小写进行精确匹配、单词精确匹配、正则匹配，支持查找指定的文件类型。

支持根据模块、目录进行查找。



选中文件后，可以定位到文件中的搜索内容，并在编辑器内打开该文件。

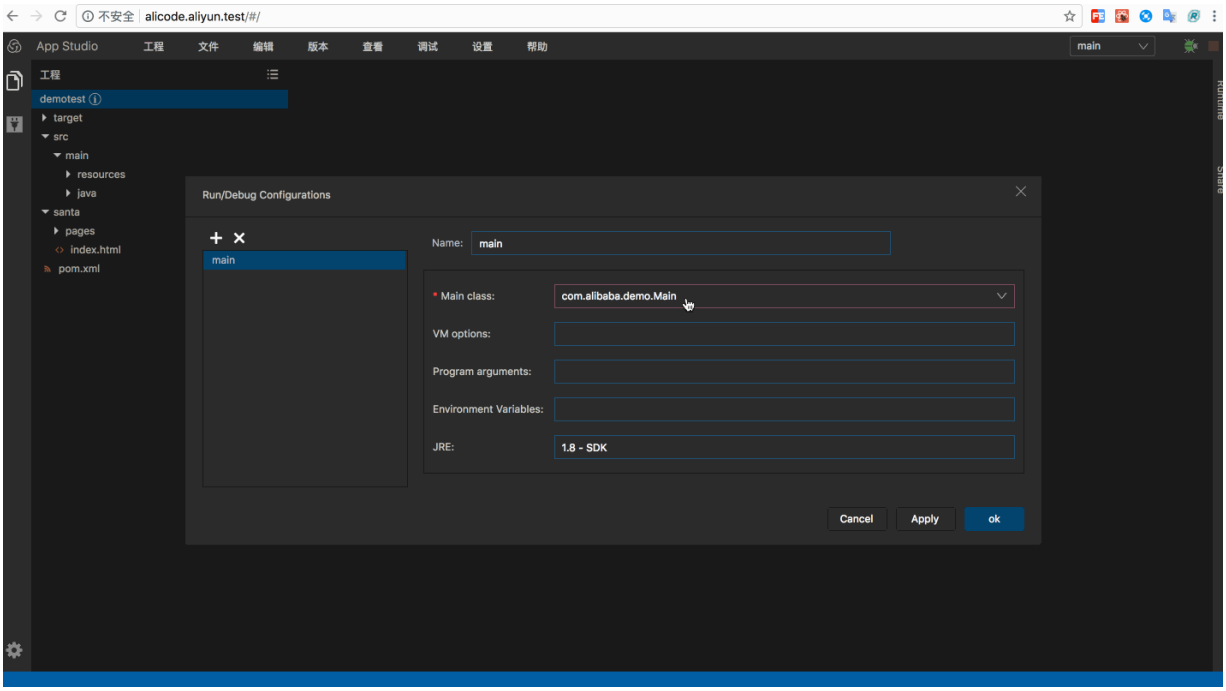

全文搜索
大小写匹配
单词匹配
正则匹配
文件后缀
请选择
Contro
8 matches in 2 files
项目
模块
目录
src/main
package com.alibaba.dataworks.controller.page;
IndexController.java 1
import org.springframework.stereotype.Controller;
IndexController.java 3
@Controller
IndexController.java 10
public class IndexController {
IndexController.java 11
package com.alibaba.dataworks.controller.api;
ApiController.java 1
import org.springframework.web.bind.annotation.RestController;
ApiController.java 5
@RestController
ApiController.java 12
public class ApiController {
ApiController.java 13
src/main/java/com/alibaba/dataworks/controller/page/IndexController.java
/**
* Sample Java Class
*/
@Controller
public class IndexController {
@GetMapping(value = { "/","/index" , "/index.htm" , "/inde
public String index(Model model){
return "index";
}
编辑器内打开

1.4.5 调试

1.4.5.1 Config配置及启动

您可通过配置入口函数，单击调试、断点等步骤，进行程序的调试。

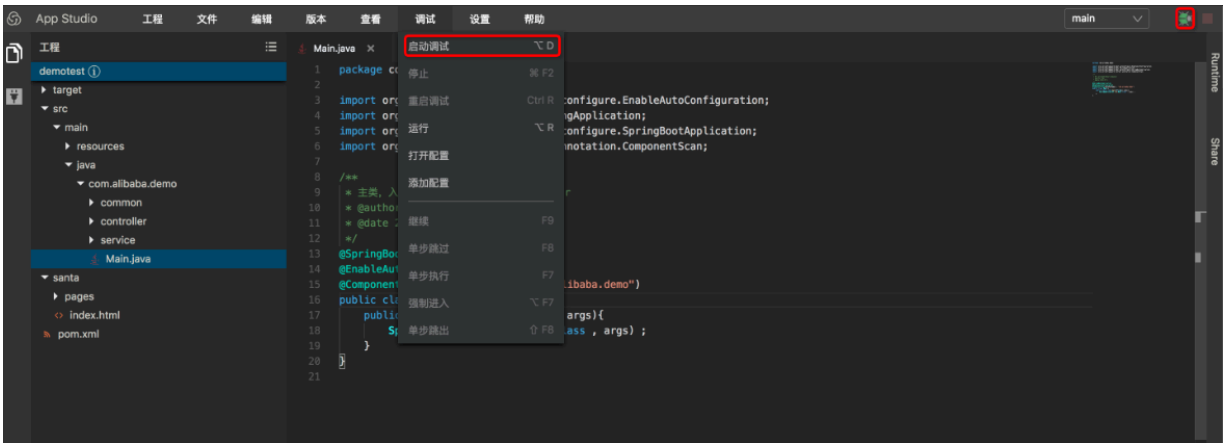
配置入口函数



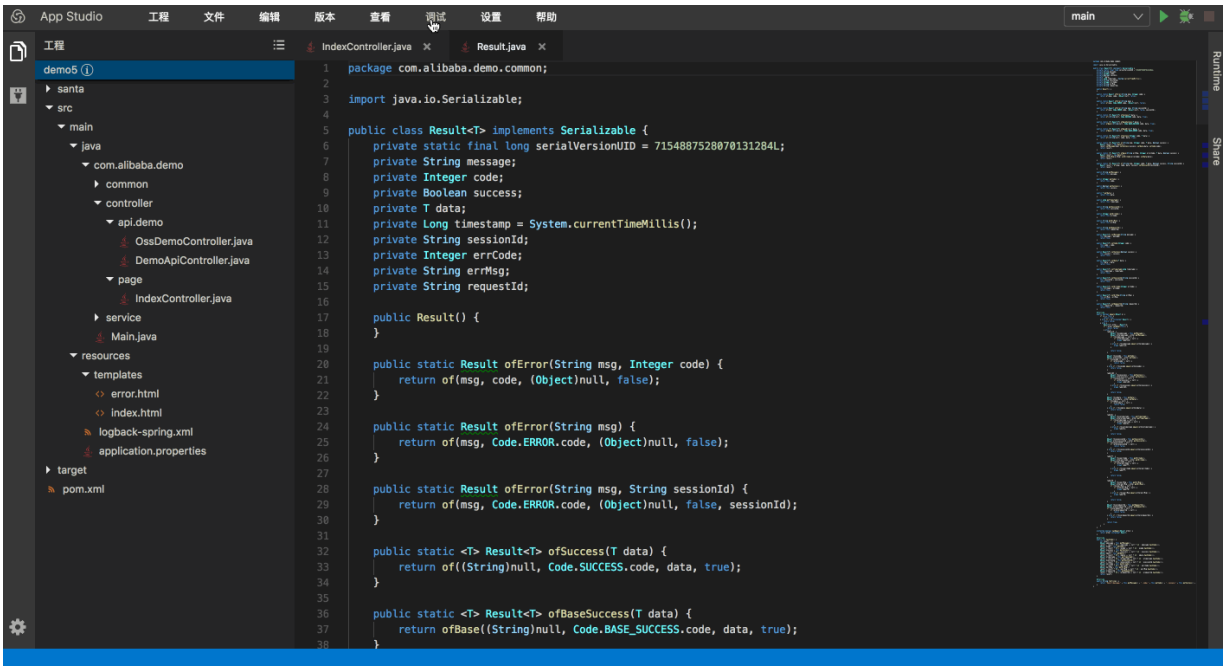
配置	说明
MainClass	您可以从多个配置中选择需要启动的main函数。
VM options	您可以配置在JVM启动时，例如-D -Xms -Xmx等配置。
Program arguments	您可以添加启动参数，此参数会被main函数的args参数接收。
Environment	环境变量参数。
PORT	端口，表示本程序需要暴露的端口信息，例如springboot经典的7001、8080等端口。
机器	您可以选择需要的机器配置进行调试。 2vCPU，4G内存 2vCPU，2G内存 ✓ 2vCPU，4G内存 4vCPU，8G内存
HotCode	此配置仅在run模式下生效，默认使用公司的HotCode2插件进行启动。

启动调试

选择菜单栏中的调试 > 启动调试。



因为需要为您准备运行环境和下载mvn依赖，第一次启动时速度较慢。重启调试会跳过此步骤，启动速度逐渐接近本地编辑器的体验。



1.4.5.2 在线调试

在线调试支持Java Application和基于SpringBoot的Web工程。

进行在线调试前，首先要配置入口函数和启动调试，完成上述步骤后，进行后续操作。

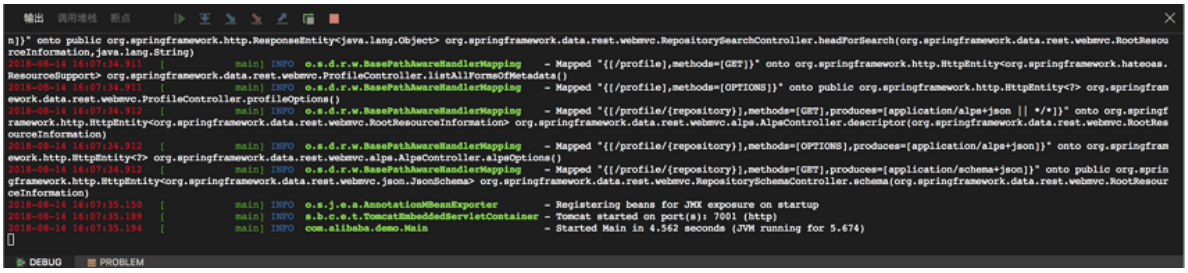
透出服务

程序成功启动后，会提供两个基本服务，您可以单击后端链接，对后端Java代码进行调试。

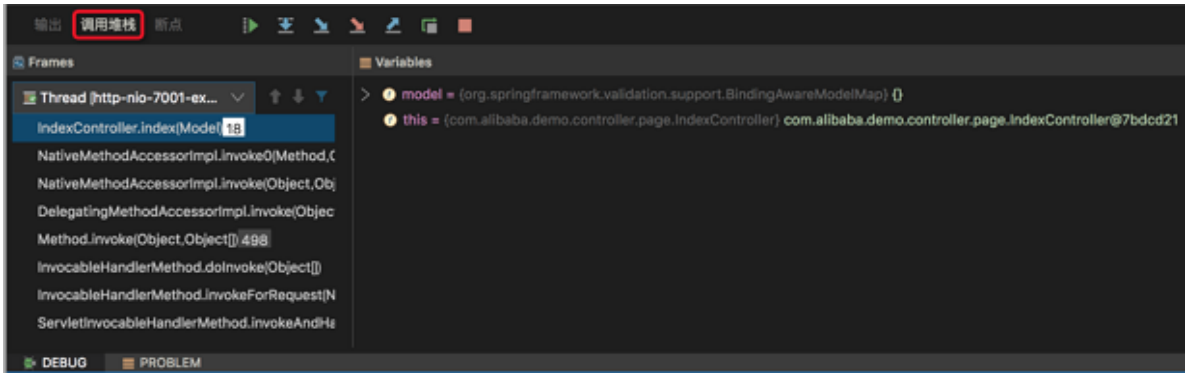
面板介绍

- **输出面板**

  输出面板会显示所有程序的标准输出（暂不支持System.in），支持ansi颜色，体验与本地终端基本一致。

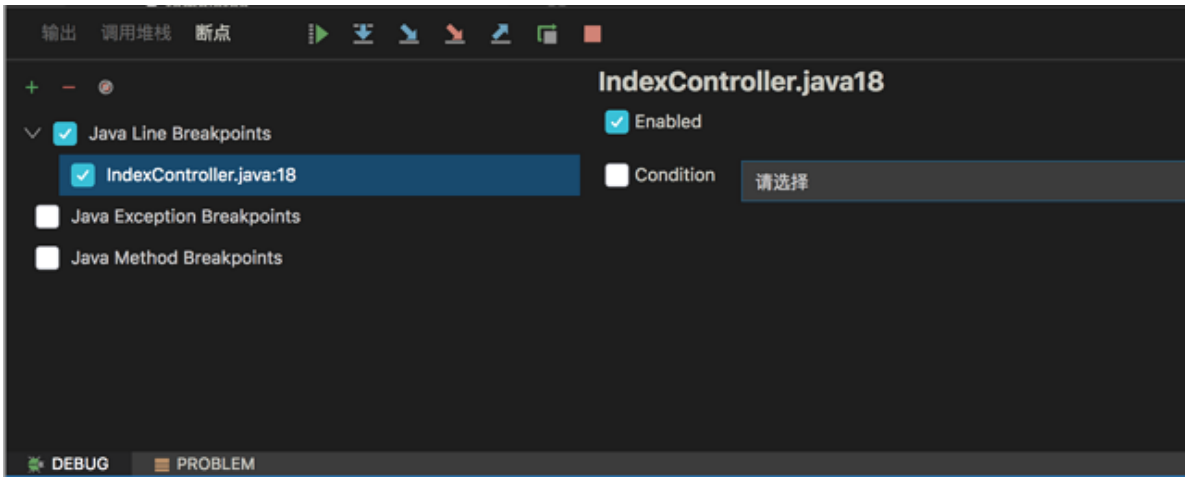


- **调用堆栈**



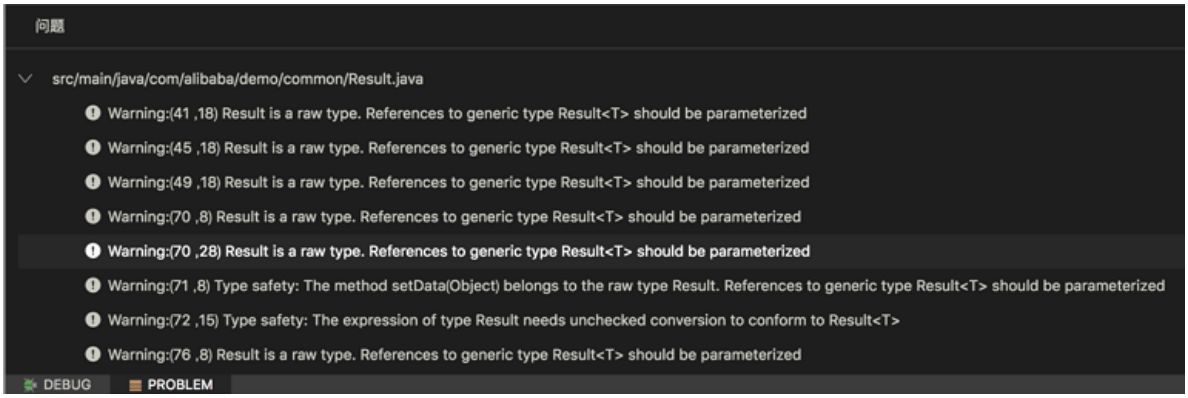
- **断点**

  断点面板为您展示当前设置的所有断点，后续将为您介绍断点类型及使用。



- **PROBLEM**

  如果程序遇到编译问题，会展示在PROBLEM面板上，您可通过单击跳转至对应的文件行。

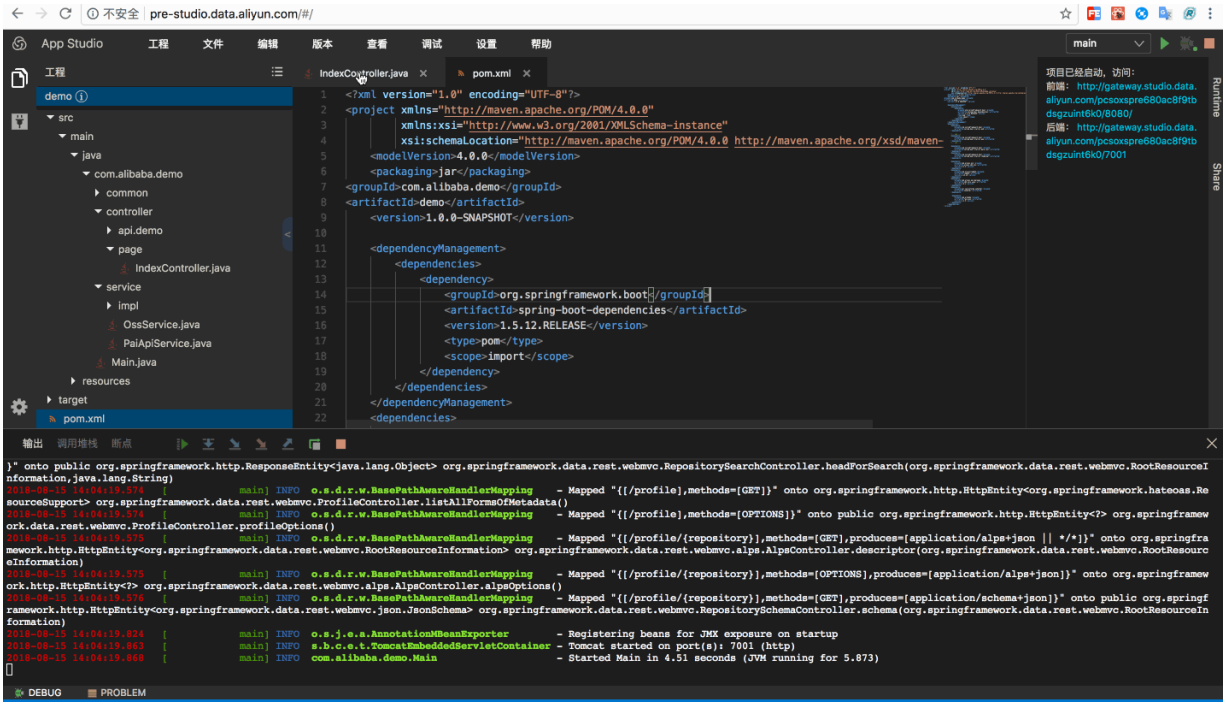


断点介绍

App Studio支持普通行断点、函数断点和异常断点，详情请参见断点类型。

调试按钮

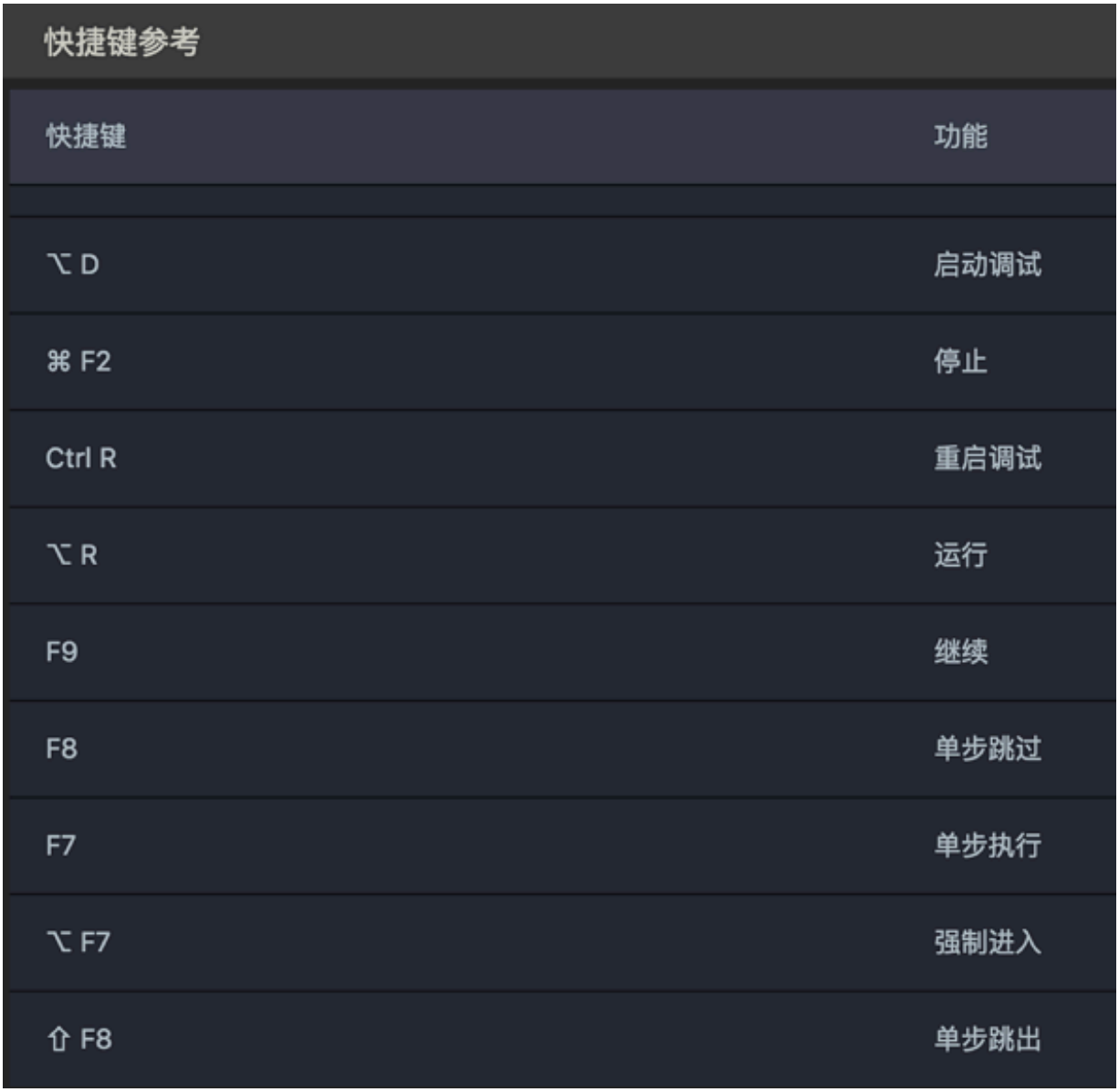
调试界面如下所示：



上图中从左到右的每个按钮所代表的含义如下：

功能	说明
continue	恢复当前断点时，当前线程继续运行。
step over	执行到下一行。
step in	进入函数。
force step in	强制进入函数，与step in的区别在于，它可以引导断点执行到java自带的类库中。
step out	从当前函数跳出。
restart	目前的restart的实现方式较为简单（可能无法完成程序清理等工作），正在优化中。
stop	停止。

操作的快捷键如下：

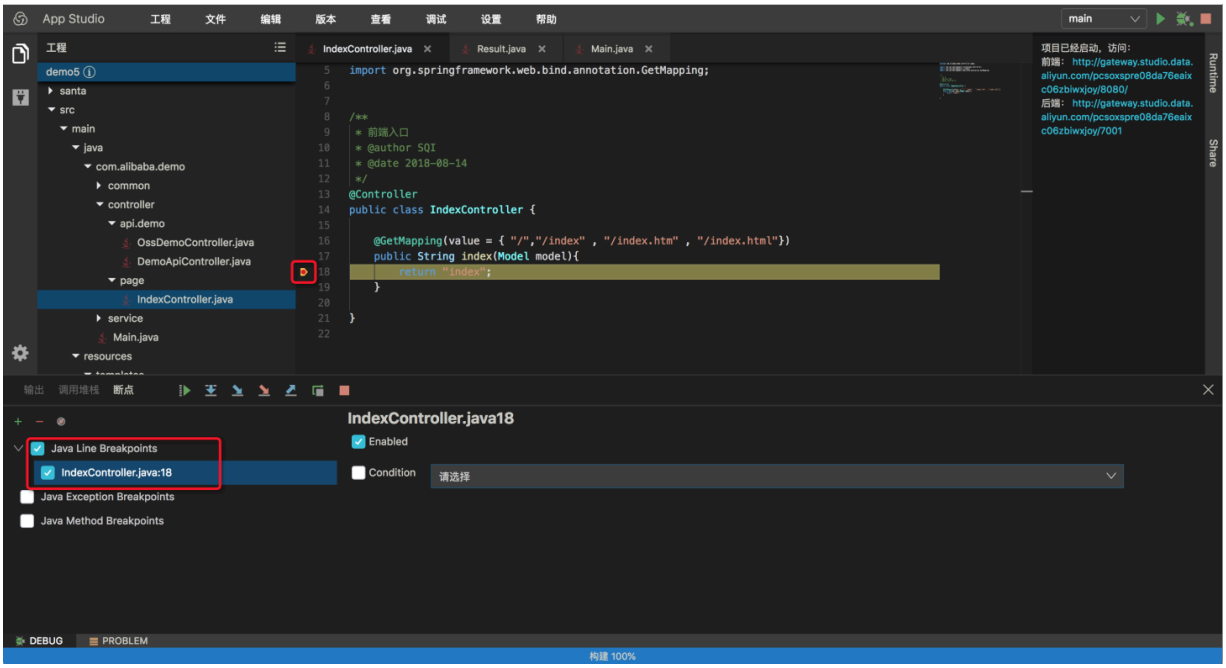


1.4.5.3 断点类型

App Studio支持普通行断点、函数断点和异常断点三种断点类型。

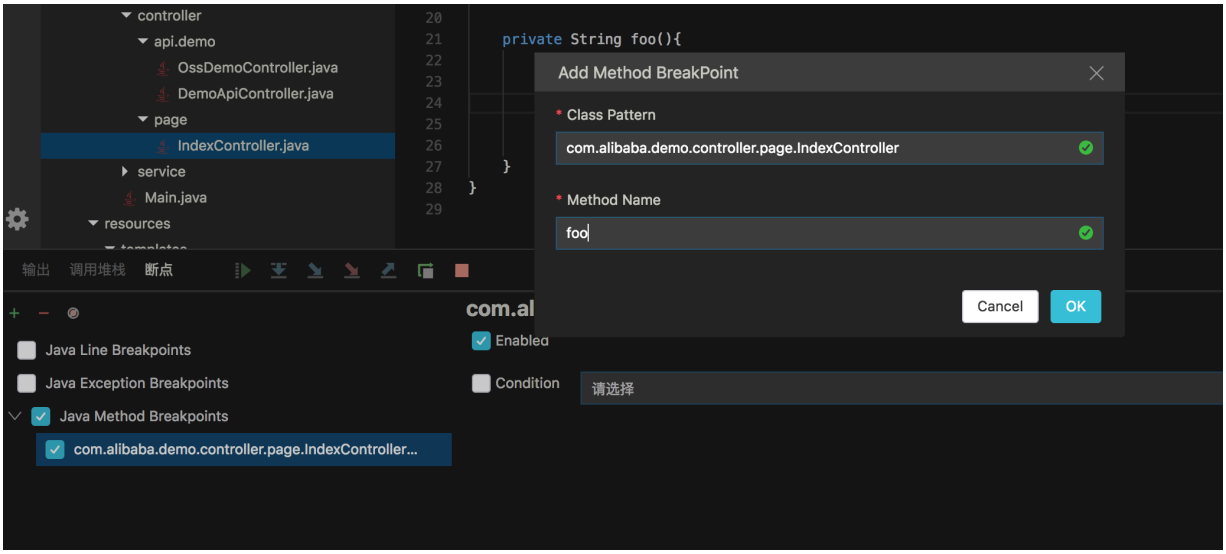
普通行断点

通过单击文件行号前的空白区域，可以生成针对该行的断点，同时断点面板中会显示该断点。

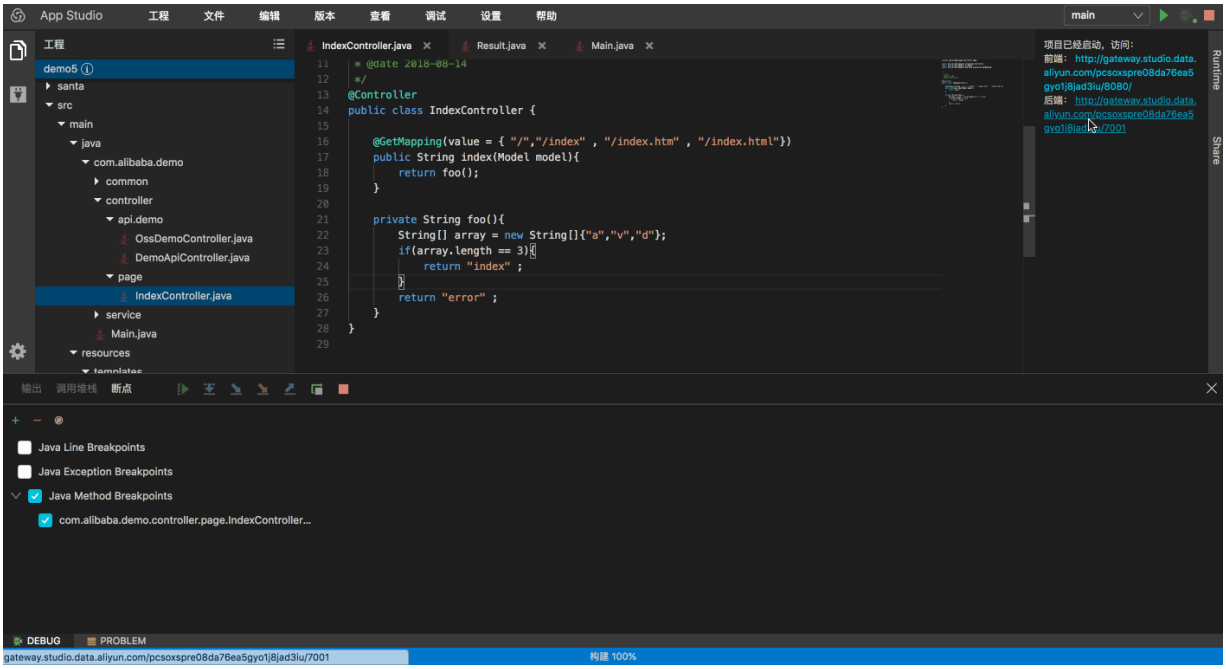


函数断点

函数断点相比异常断点与行断点的不同点为：函数断点会触发两次事件，即entry/exit。您可以手动添加一个函数断点，也可在函数被定义的地方打断点，同样会产生一个函数断点。

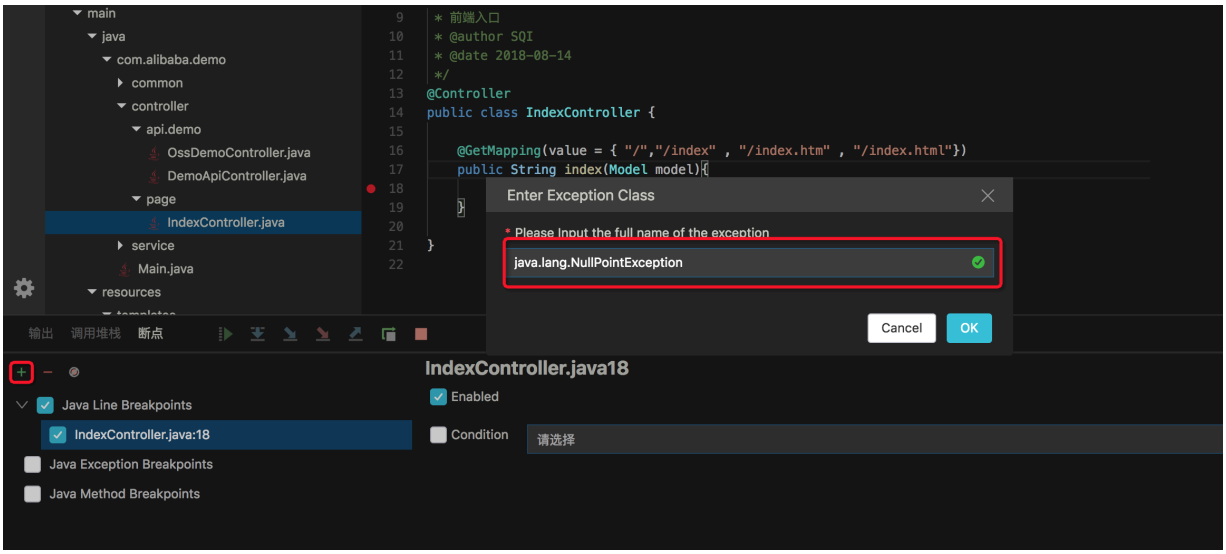


触发它后可以看到，进入该函数时会暂停，即将跳出程序时也会暂停。

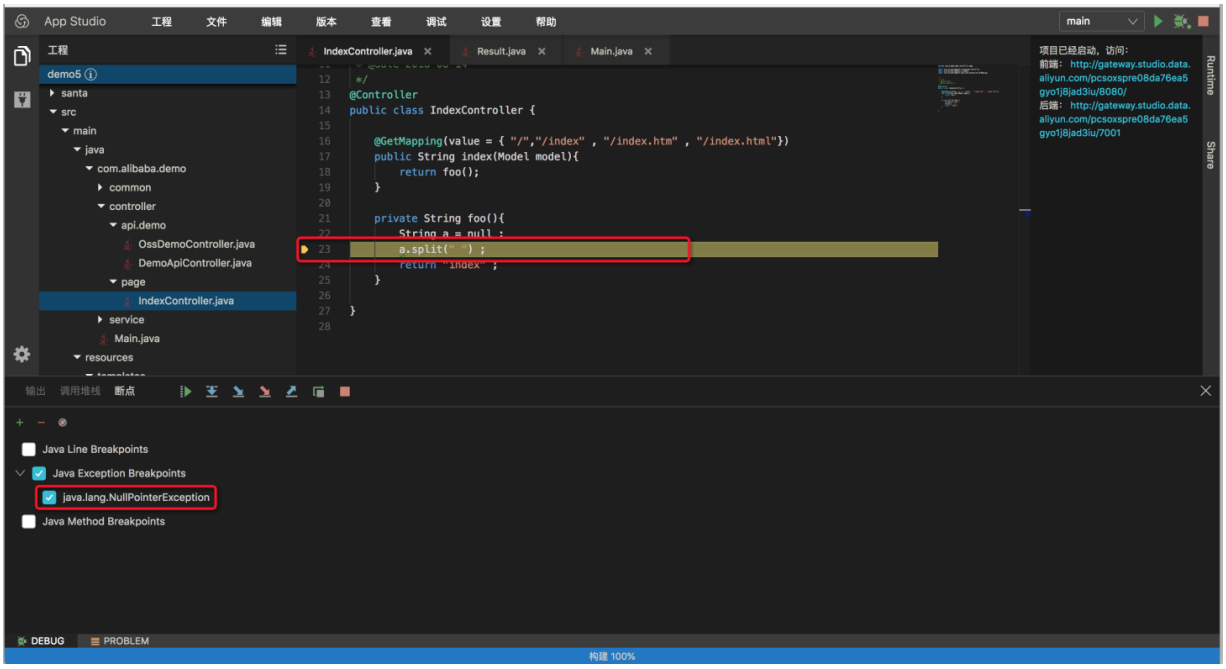


异常断点

如果配置了异常断点，当程序在遇到异常时，会在出现异常的地方进行断点。



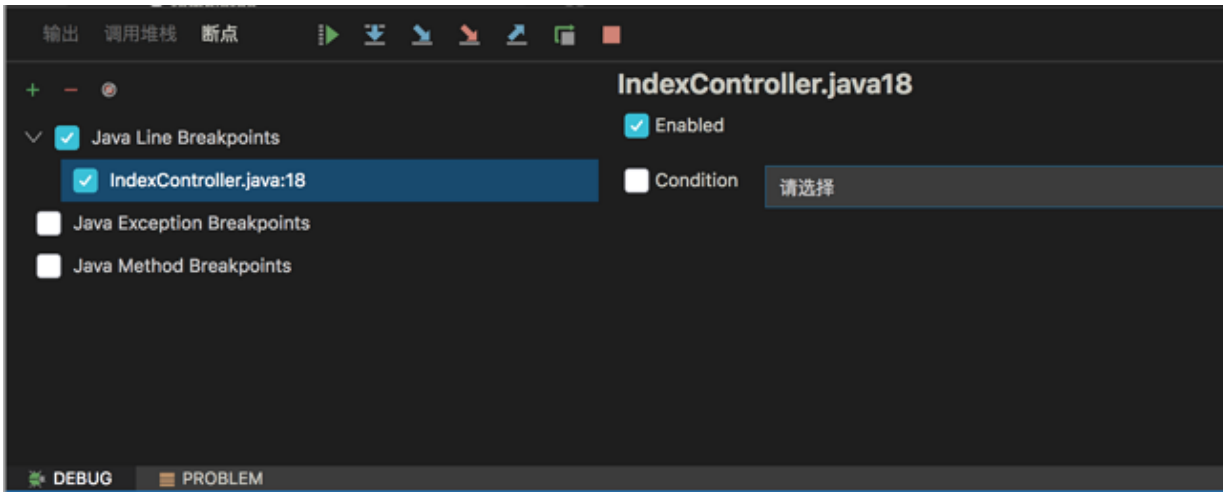
触发index，由于出现了NullPointerException，所以断点在23行。



1.4.5.4 断点及操作

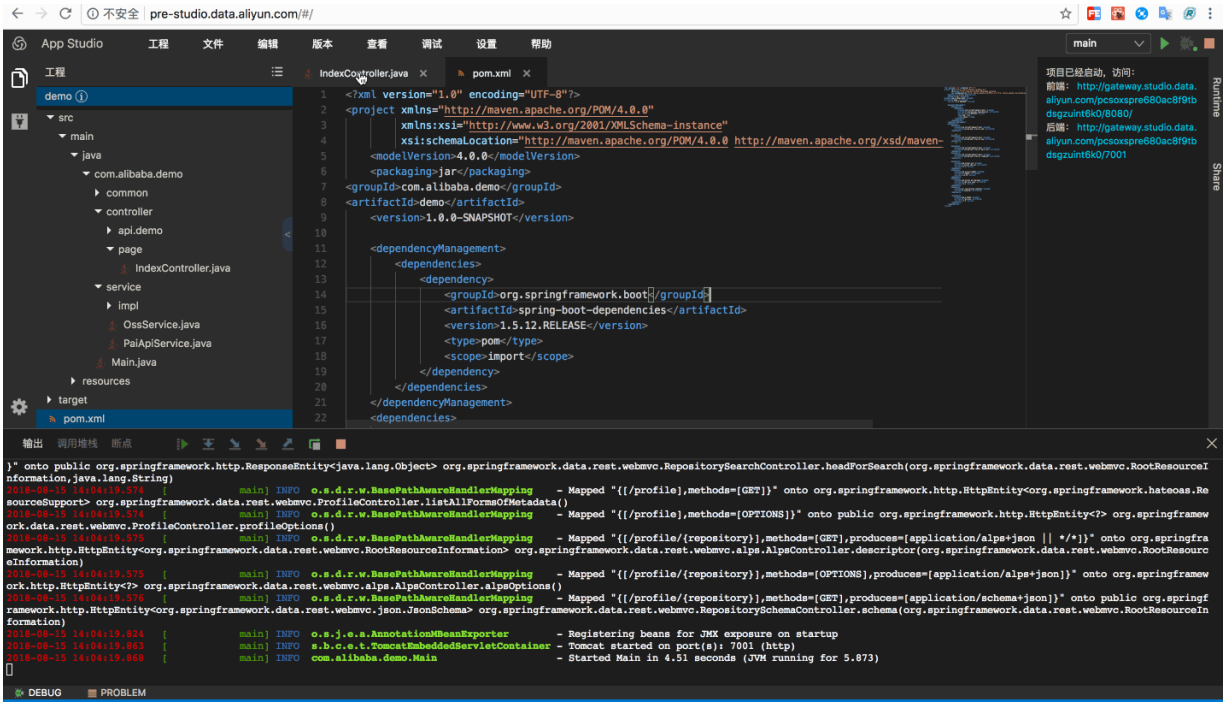
断点面板为您展示当前设置的所有断点，本文将为您介绍断点的操作。

断点包括普通行断点、函数断点和异常断点三种断点类型，详情请参见断点类型。



调试操作

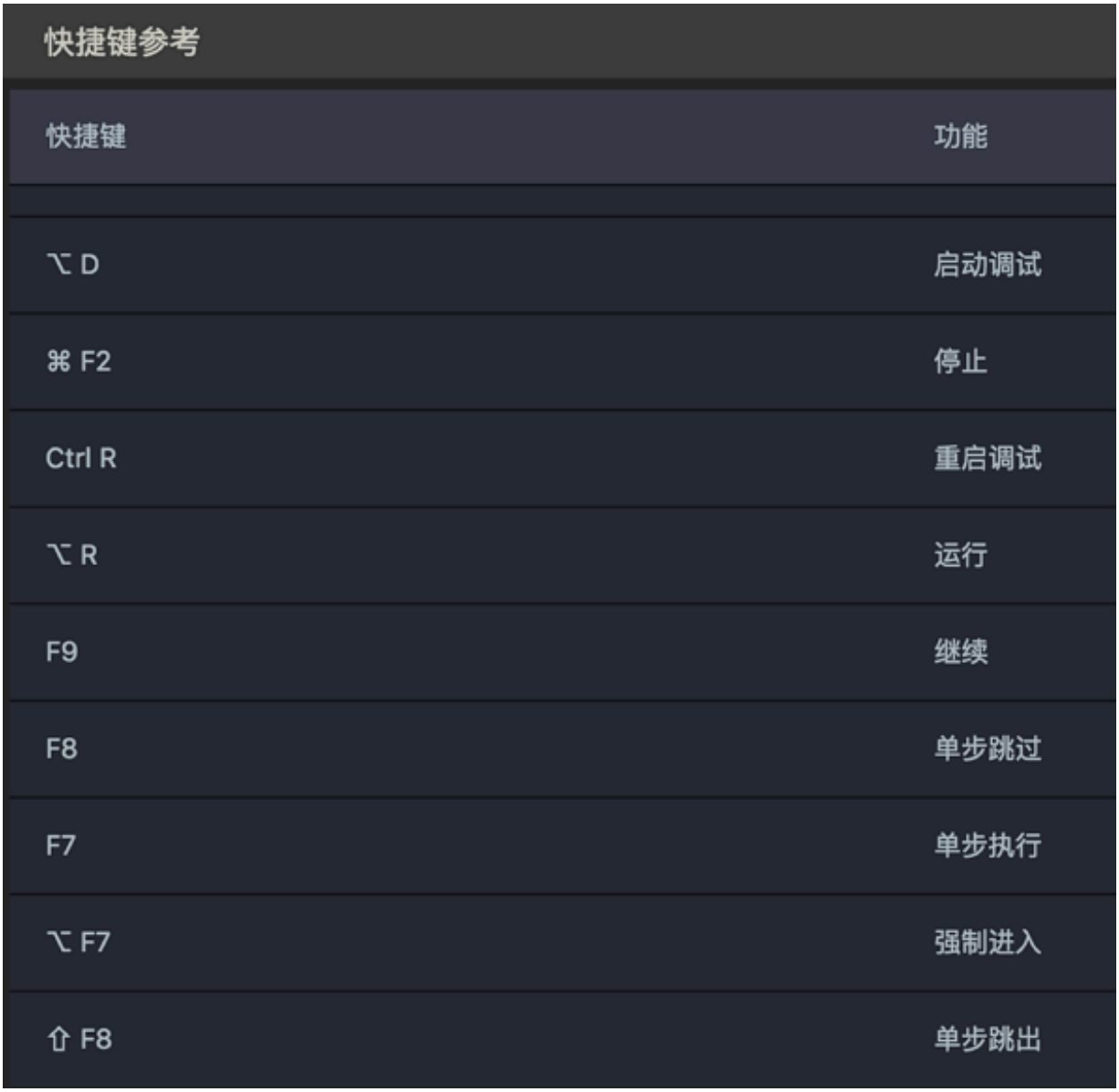
调试界面如下所示。



上图中从左到右的每个按钮所代表的含义如下。

功能	说明
continue	恢复当前断点时，当前线程继续运行。
step over	执行到下一行。
step in	进入函数。
force step in	强制进入函数，与step in的区别在于，它可以引导断点执行到Java自带的类库中。
step out	从当前函数跳出。
restart	目前的restart实现方式可能无法完成程序清理等工作，正在优化中。
stop	停止调试。
Drop Frame	删除当前栈，回退到上一个函数。
Run to Cusor	执行到当前行，可以在某一行打一个临时断点。
计算表达式	可以任意执行一个表达式进行计算。

操作的快捷键如下所示。

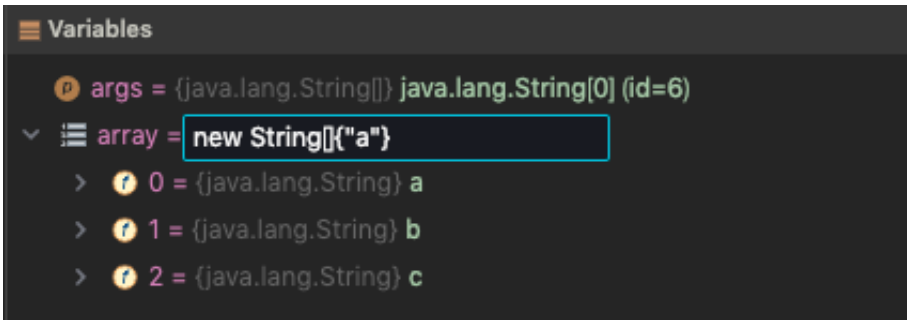


快捷键参考

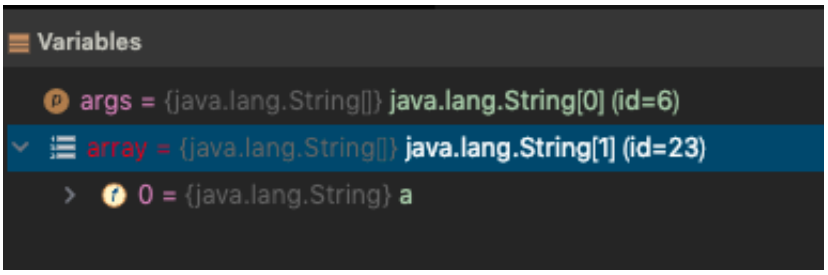
快捷键	功能
⌥ D	启动调试
⌘ F2	停止
Ctrl R	重启调试
⌥ R	运行
F9	继续
F8	单步跳过
F7	单步执行
⌥ F7	强制进入
⇧ F8	单步跳出

- **对变量进行赋值**

  您可以直接在断点上对一个变量进行赋值。



  双击某个字段，然后构造一个表达式对当前值进行赋值，按enter键生效。

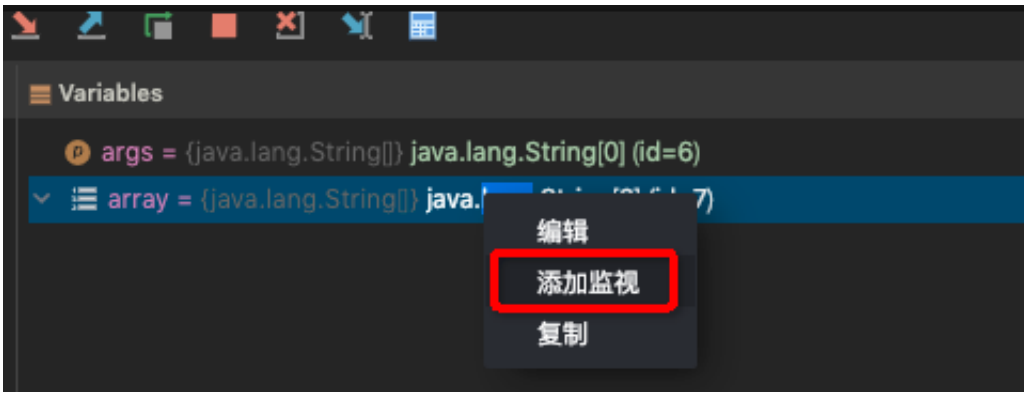


- **表达式计算**

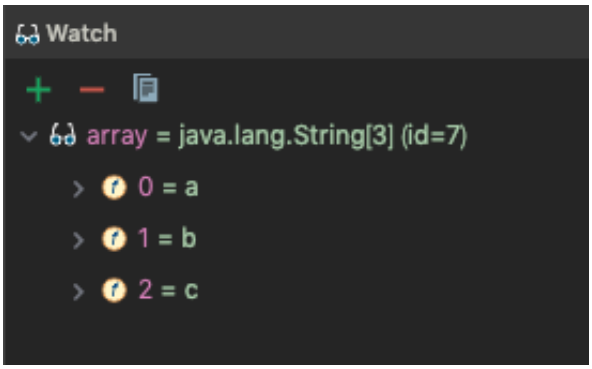
  打开计算表达式面板，输入可执行表达式。

- **变量监视**

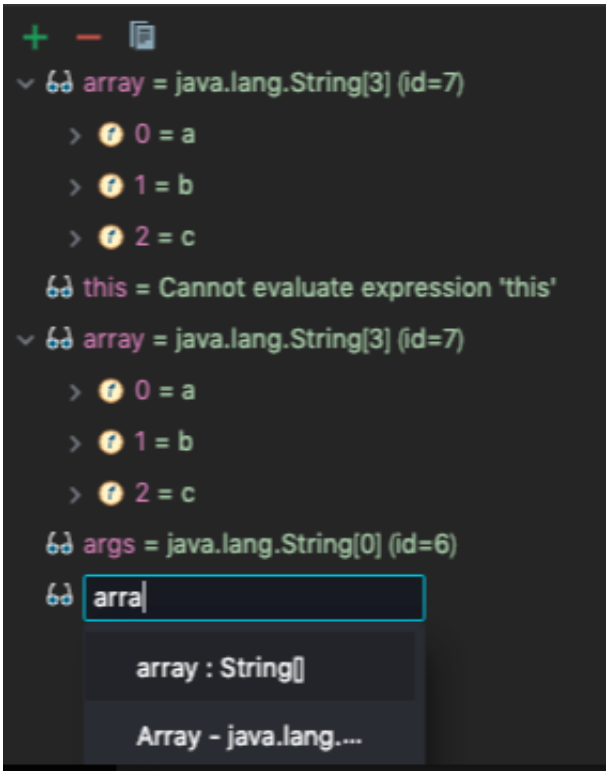
  您可以右键单击变量，选择添加监视。



  添加后，即可在右边面板看到相应的变量监视。

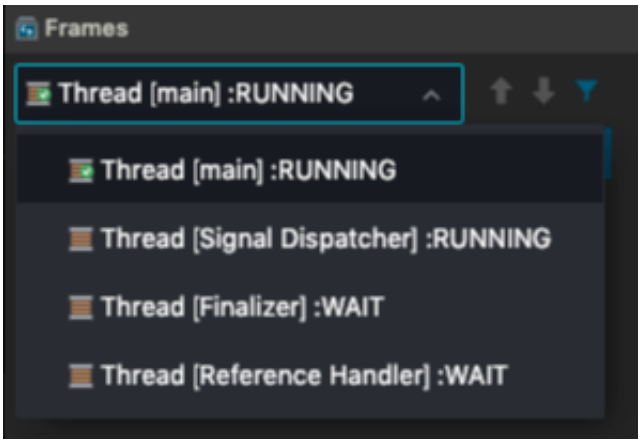


  您也可以在watch上手动新增变量。



- 线程操作

  您可以在调试面板查看线程操作。

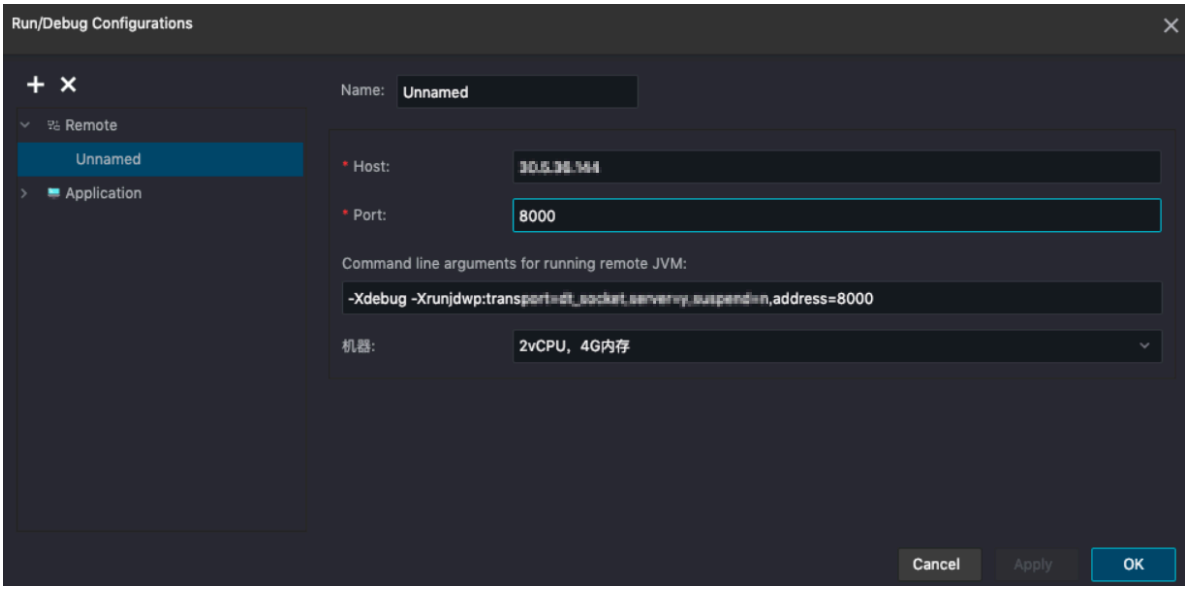


  根据当前线程的运行进度，下拉框中会显示RUNNING或WAIT等不同的信息。当您选中另外的线程时，变量的面板信息也会随之改变。

1.4.5.5 远程调试

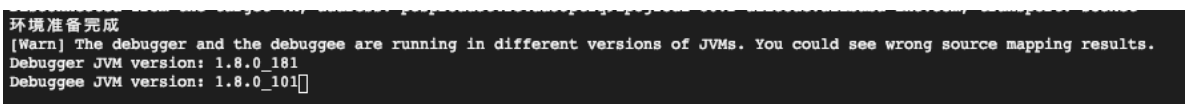
由于调试机器在日常环境，因此只能调试日常环境上部署的应用。

1. 配置调试信息。



> **说明:**
>
> 您需要填写Host和Port信息，告知JVMTI需要连接哪个远端服务。

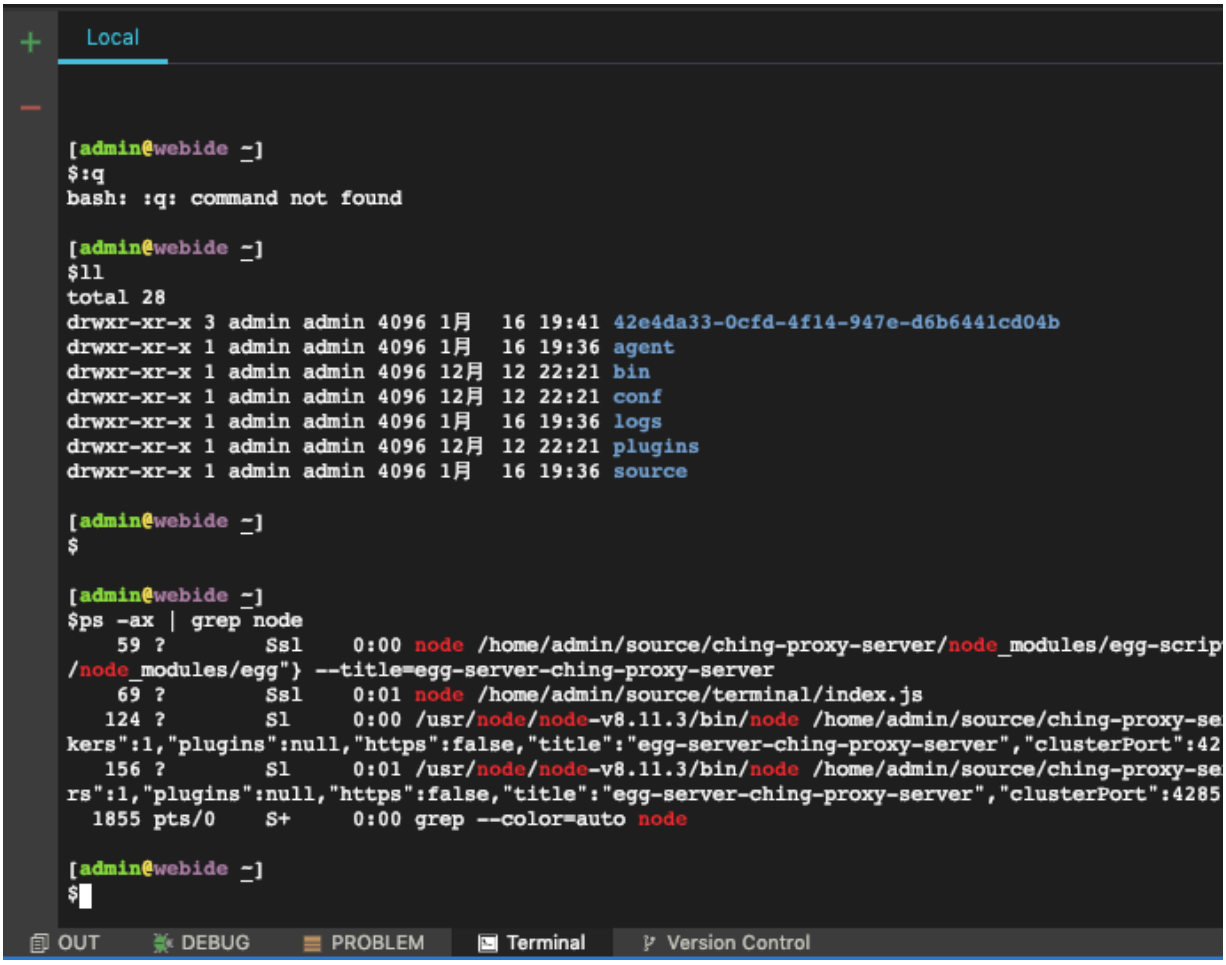
2. 单击调试，遇到Debugger信息时，代表连接成功，即可开始调试。



远程调试使用JVMTI进行Socket连接，本质上Debugger和Debuggee之间仅传输JVM运行信息，不会传输标准输出和错误输出。

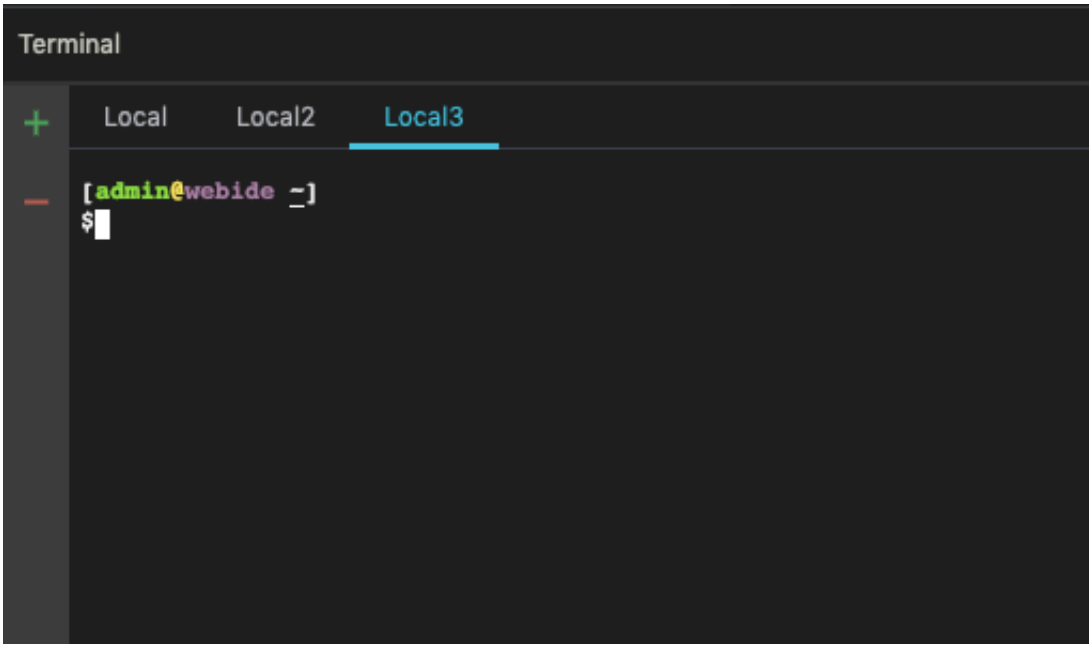
1.4.5.6 终端

Terminal按钮显示在页面底部。



App Studio支持常规的`ls`、`cat`等Shell命令和`vi`、`top`等带有交互的命令。

您可以开启多个终端。

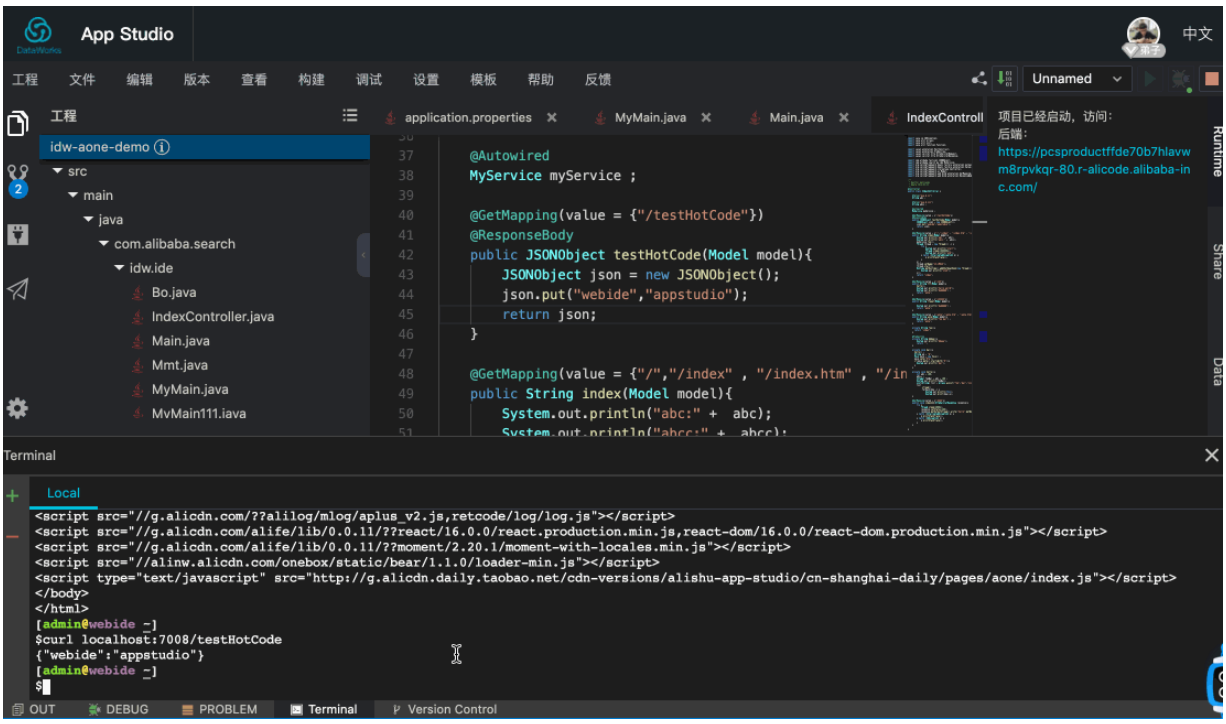


1.4.5.7 热部署

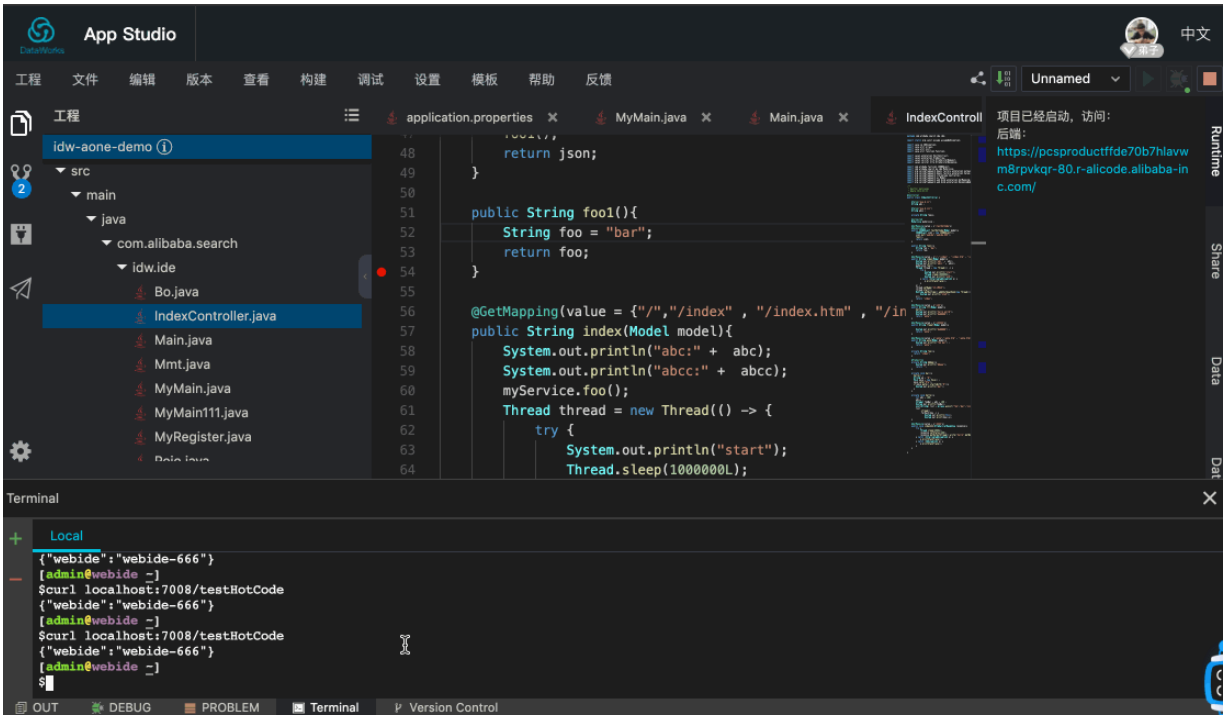
热部署是指代码运行过程中，您手动修改的代码可以在不重启服务的情况下生效。

例如SpringBoot在运行/调试过程中，修改完代码后无需重启，保存即可生效，App Studio已经默认包含此功能。

除调试模式外，运行模式下也支持这项功能。触发热部署无需安装插件和手动编译文件，您只需保存文件即可。

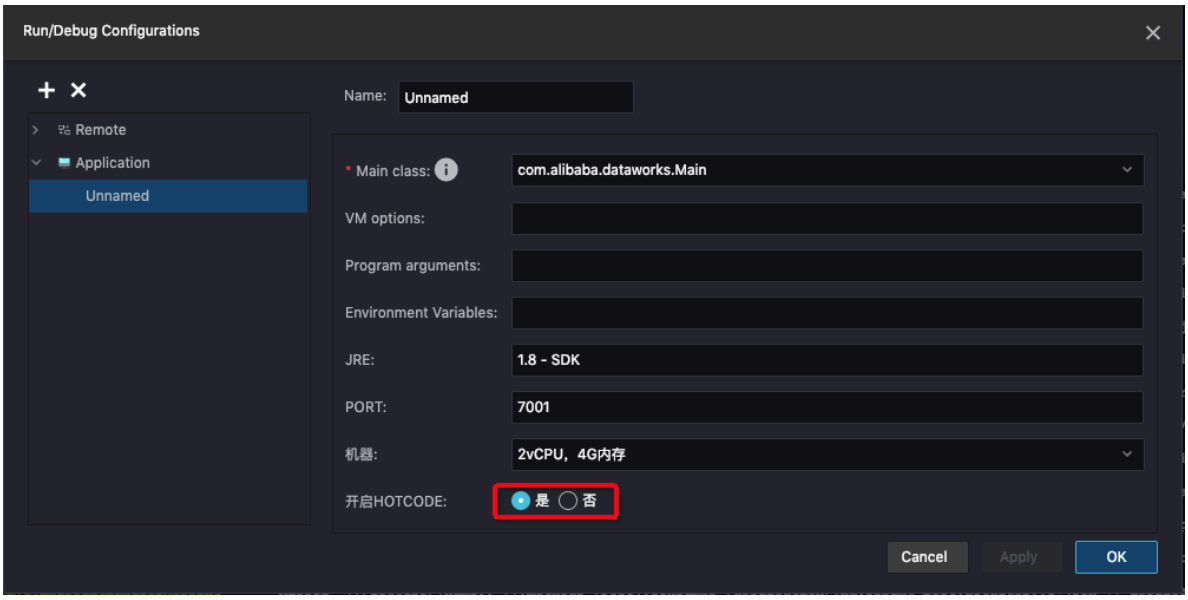


如果您正在Debug中进行代码变动，会自动删除当前运行栈，回退到函数入口。



运行模式下的热部署配置

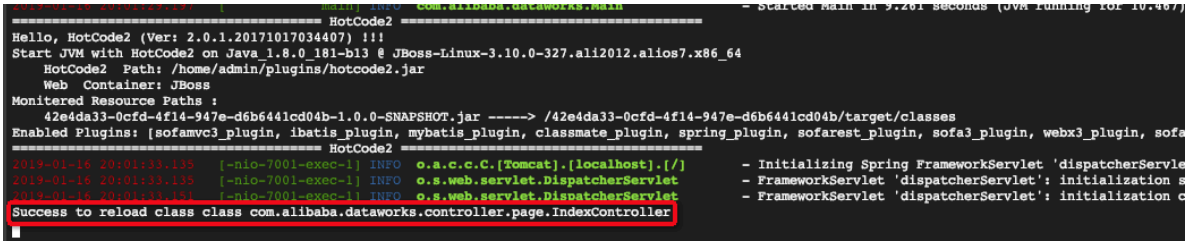
1. **在配置面板上主动开启热部署。**



启动后，即可在输出中看到HotCode2的输出信息。

2. 触发热部署。

当您对文件进行修改时，需要手动触发文件保存。



3. 当代码增量同步完成后，控制台显示Reload某个类的输出，则代表热部署生效。代码示例如下：

```
public class IndexController {
    @RequestMapping("/")
    @ResponseBody
    public String index(){
        return "cccc";
    }
}
```

您可以将Return字符串内容改为其它字符串，让其立即生效。

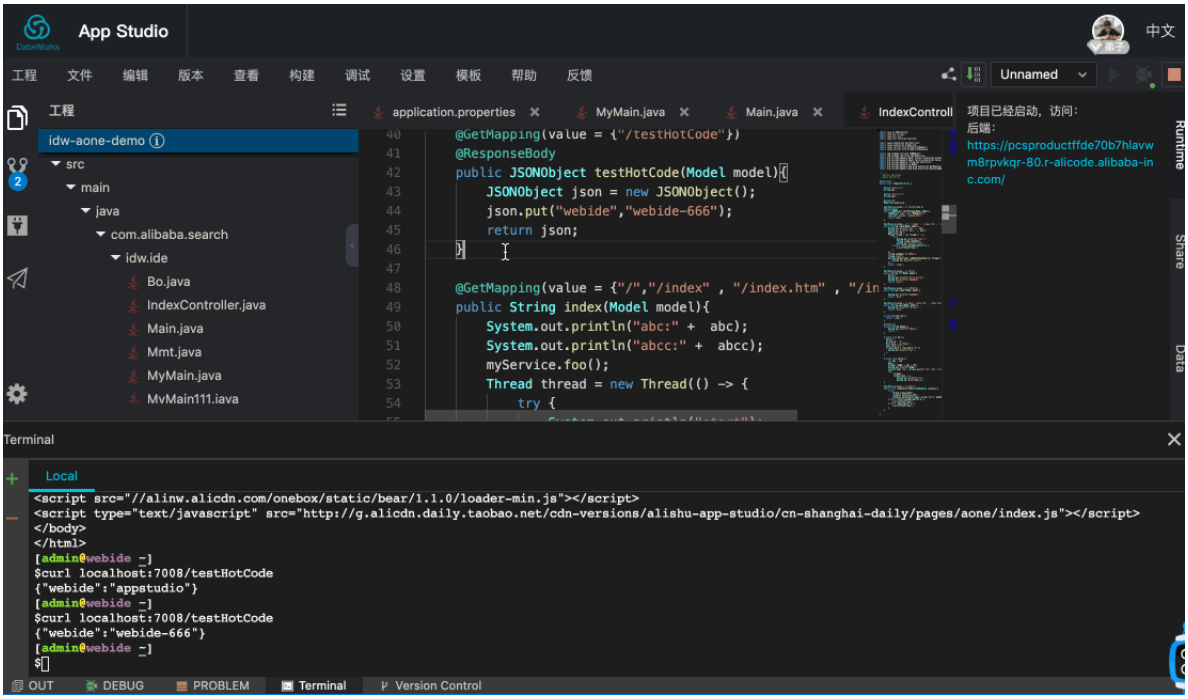
Debug模式下的热部署

您可以通过JDI原生方法实现Debug模式下的热部署，但由于JVM的限制，在给某个类增加或删除方法时，无法进行热部署。您同样只需保存文件即可触发热部署。

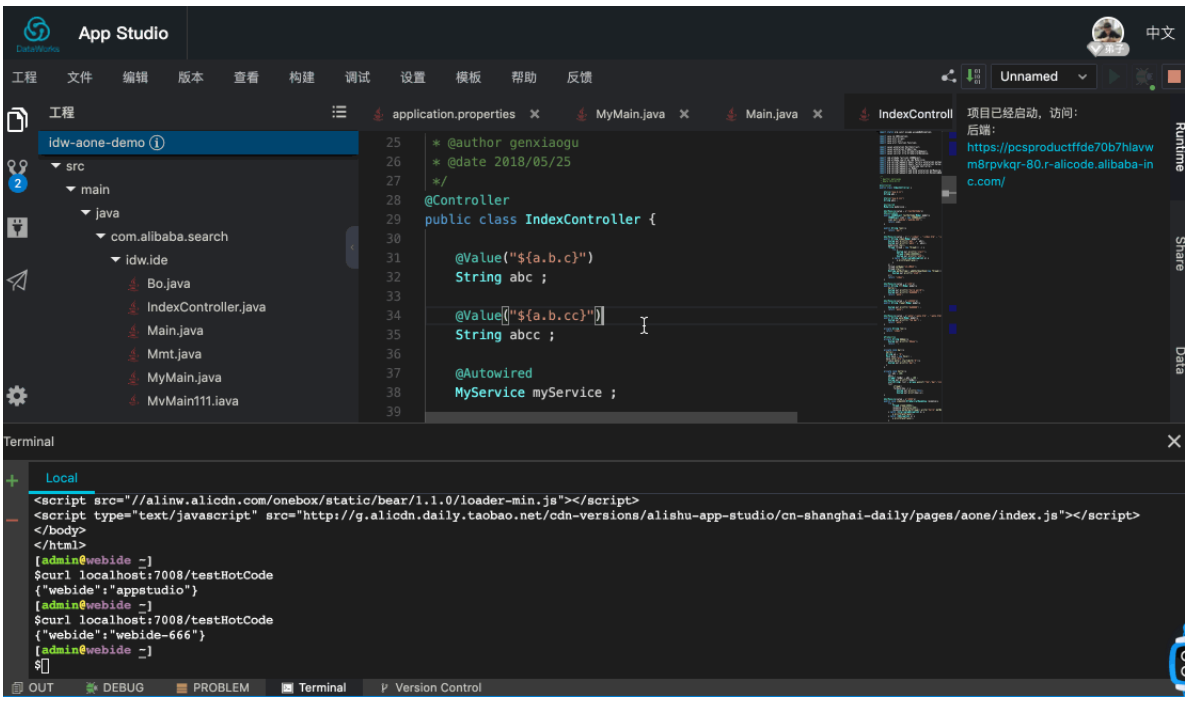
说明：

JVM原生不支持对类结构进行变动后的热部署，新增或删除类等其他操作都可以支持热部署。

- 新增方法或删除方法。



- 新增字段。



1.4.6 协同编程

本文将从实时协同编辑、邀请协作者、加入写作项目、协作者面板和权限等方面为您介绍协同编辑。

App Studio支持实时协同编辑功能，团队中多个成员可以同时在同一个项目中开发、编写代码，并实时查看其它成员的改动。能够避免同步代码、合并分支的繁琐，显著提升开发效率。

邀请协作者

项目的所有者可以邀请其他开发者加入项目进行协作。

1. **打开要分享的项目。**
2. **单击右侧的Share展开协作者面板。**
3. **单击右上角的邀请，进入邀请流程。**

4. **填写邀请协作者对话框中的各配置项。**

配置	说明
用户名	填写邀请的写作者的用户名。
权限	根据自身需求选择只读或读写权限。

5. 单击确认，即可成功邀请。

加入协作项目

当您被邀请加入其他开发者的项目后，可以在打开的工程面板下，选择我参与的，查看您加入的协作项目。单击即可加入项目，开始实时协同编辑。

协作者面板

实时协同编辑时，协作者们可以互相查看当前的状态。

1. 单击页面右侧的Share展开协作者面板。

2. 查看相应协作者的在线状态、正在编辑的文件和拥有的权限。

> **说明:**
> 项目所有者可以移除协作者。

权限说明

在协同编辑的过程中，参与的协作者权限分为以下三种：

- 所有者：所有者是项目的创建者，无法变更。所有者可以邀请其他开发者加入项目，也可移除其他的协作者。
- 读写权限：拥有读写权限的协作者可以查看项目中的所有文件，也可以对这些文件进行编辑。
- 只读权限：拥有只读权限的协作者只能查看项目中的文件，但是无法进行编辑。

1.4.7 应用部署

本文将为您介绍如何在App Studio上新建一个应用并部署到生产环境，获得一个可以通过公网访问的应用。

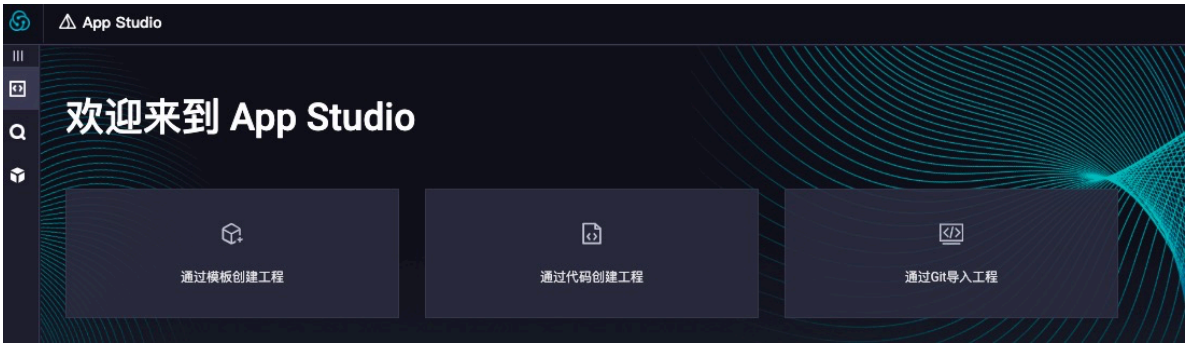
进入App Studio

新建工程

1. 登录DataWorks控制台，单击相应工作空间后的进入数据开发。

2. 单击左上角的DataWorks图标，选择全部产品 > App Studio。

3. 进入App Studio页面后，您可以通过模板、代码和Git导入三种方式创建工程。

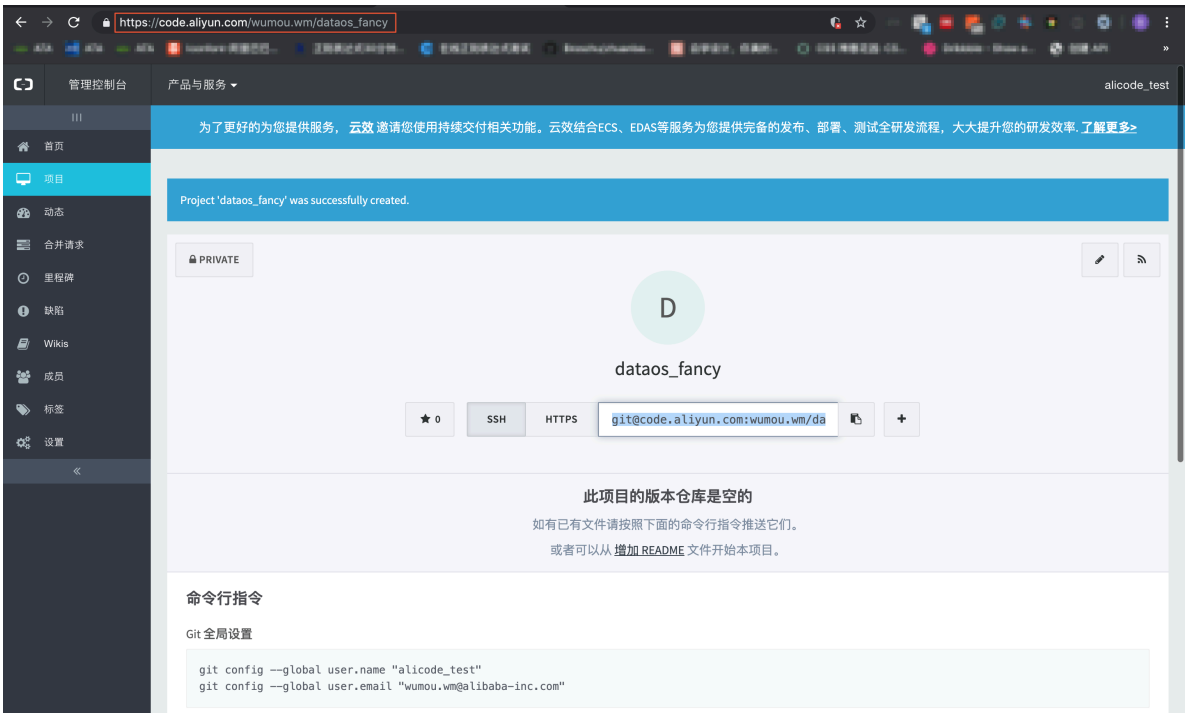


4. 根据自身需求选择相应的新建方式，并填写配置后，单击提交，即可新建工程。详细操作请参见工程管理。

关联Git

发布应用前，需要初始化Git。

1. 在Code页面新建一个repo，并保存仓库的SSH地址。



2. 进入App Studio页面，打开新建的工程，单击版本，选择初始化&关联远程仓库。

3. 填写关联远程仓库对话框中的配置，单击提交。

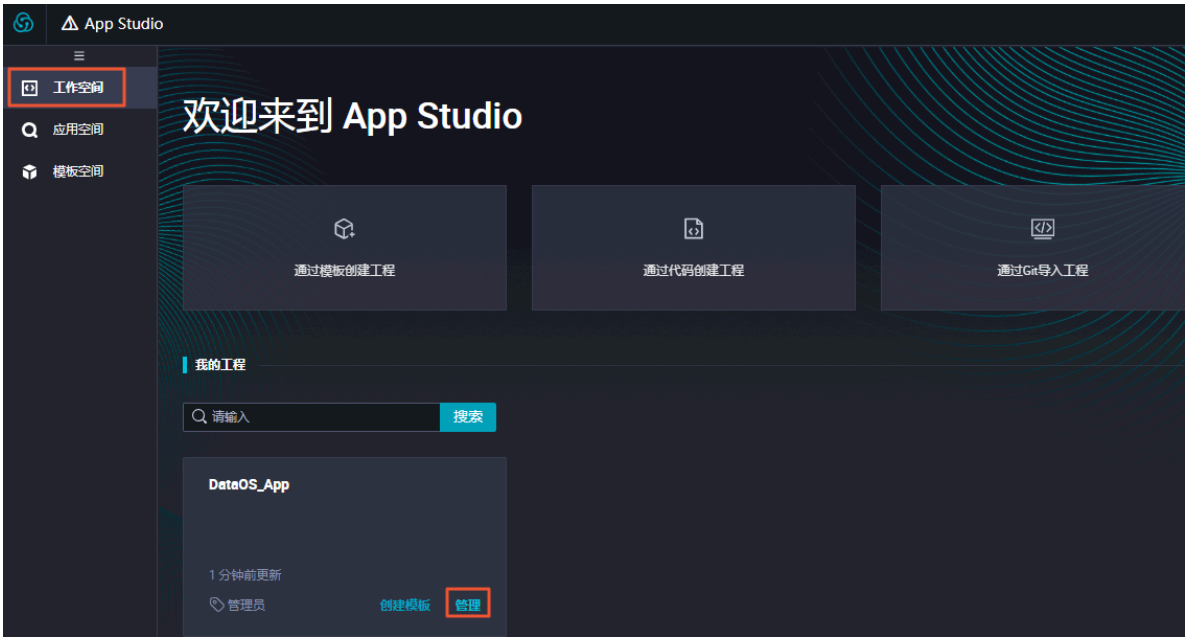
说明:

如果您未绑定SSH Key或Git用户名邮箱，可以根据页面引导进行操作。

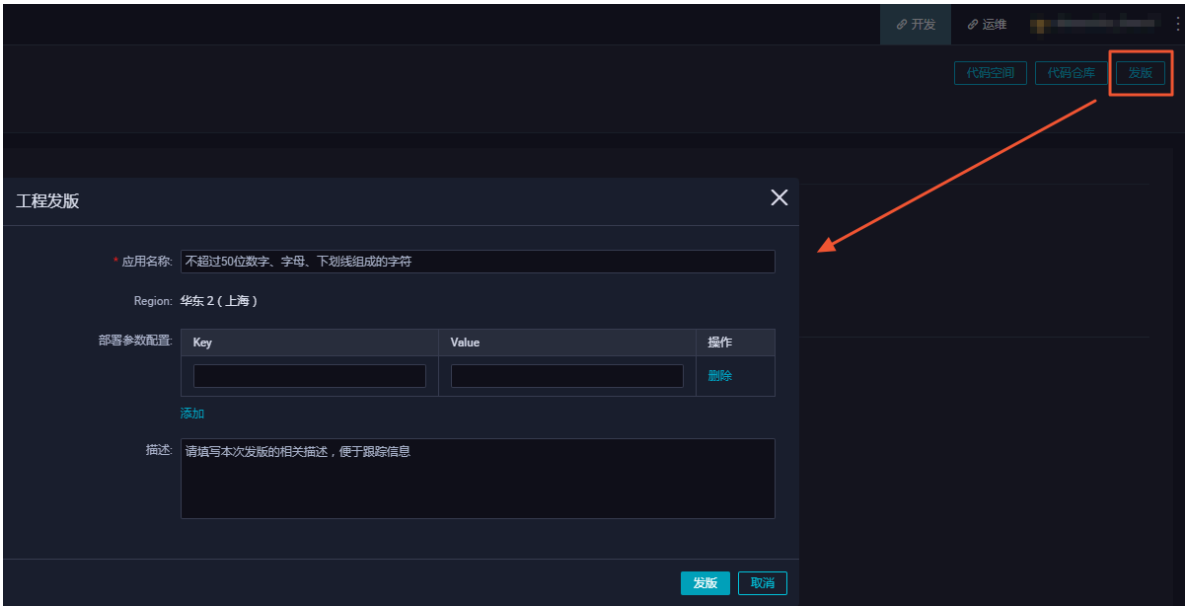
发版

关联Git完成后，即可通过发版创建应用。

1. 返回工作空间页面，单击相应工程下的管理。



2. 单击右上角的发版，填写工程发版对话框中的配置。



3. 配置完成后，单击发版。

部署应用

1. 单击左侧菜单栏中的应用空间，进入应用空间页面。
2. 单击已发版应用下的发布。

3. 单击应用部署提示框中的购买链接，根据指引在相应的区域购买AppStudio运行空间独享资源。

4. 购买成功后，单击部署控制台，进入运维页面。

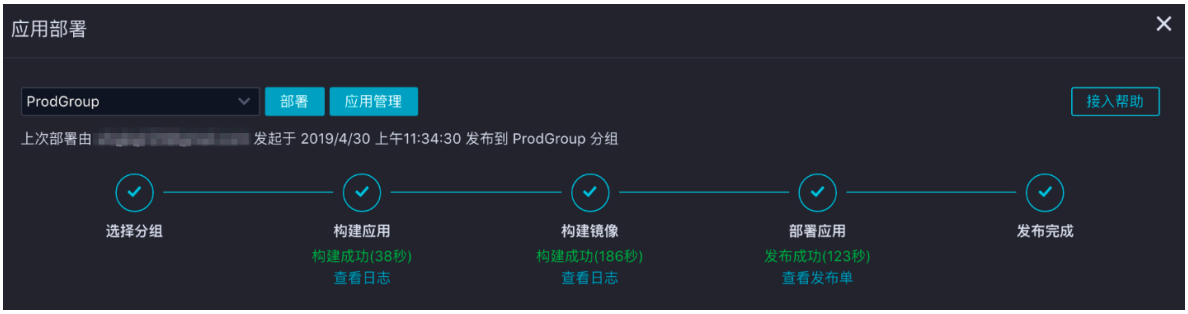
> **说明:**
>
> 此时需要解绑之前绑定的Host。

5. 单击分组列表下的创建分组，完成分组的创建。

6. 选择操作 > 应用扩容，将刚刚购买的机器加入创建的分组中。

7. 完成后刷新应用空间，单击部署，将应用发布到默认的分组即可。

出现下图中的状态，代表发布完成。此时应用已经部署到您的ECS，并启动服务。



VPC下沉

VPC下沉是指将VPC加入到用户购买机器的网段。该操作需要在阿里云和App Studio应用运维平台实现，且每个项目仅需执行一次，之后的版本迭代只需执行上面的部署应用即可。

VPC接入授权

App Studio用于发布的ECS通过弹性网卡和用户VPC连通，需要用户给App Studio的服务账号添加网卡权限， 提交给运维平台。

1. 进入角色管理页面，单击新建RAM角色。

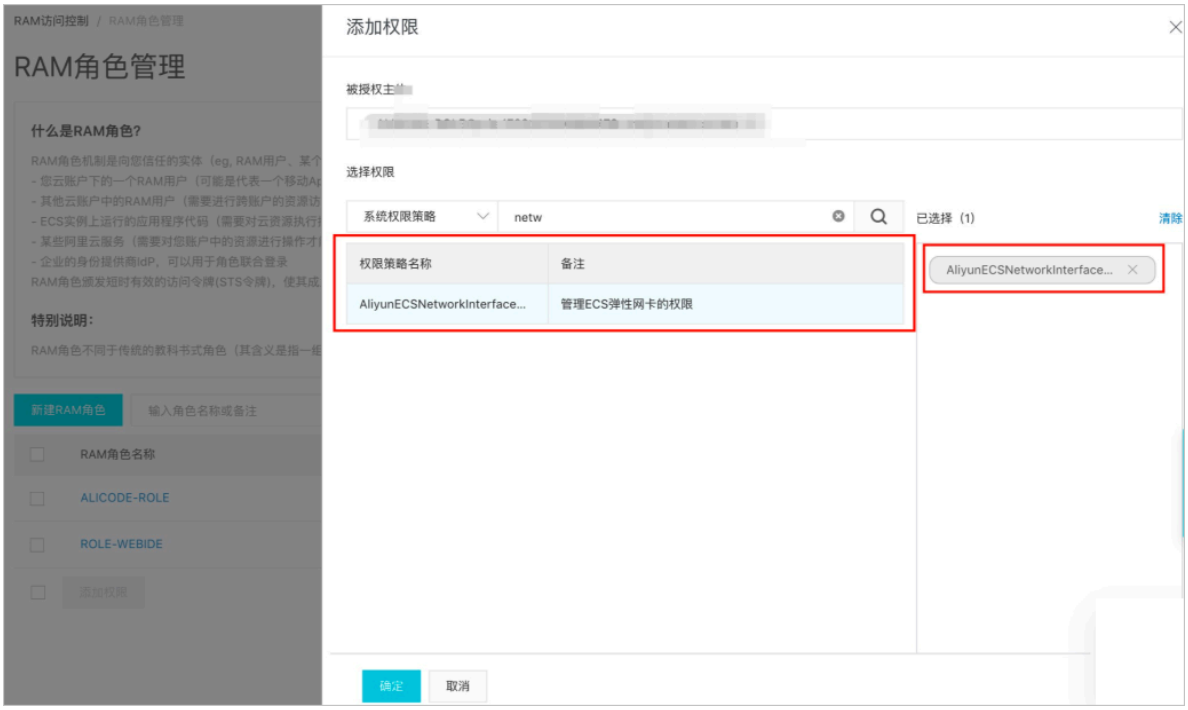
2. 在新建RAM角色对话框中，选择类型为阿里云账号，单击下一步。

3. 填写角色名称，并选择云账号为其他云账号（此处固定选择为1591568227964362）。

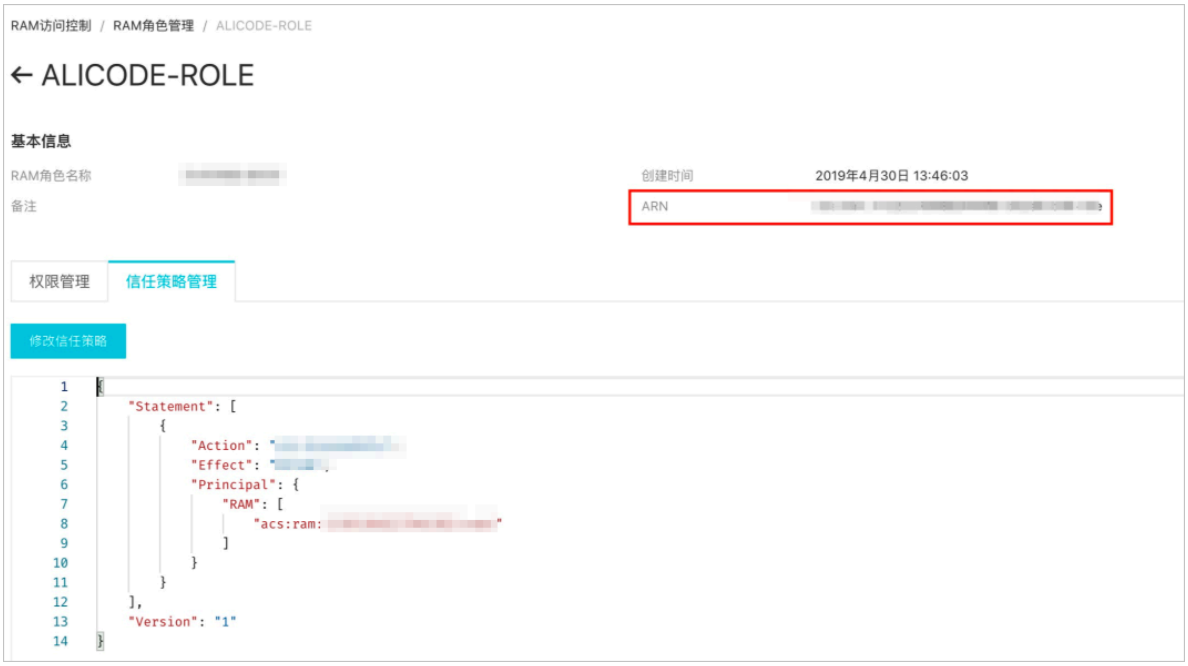
4. 单击完成，跳转至RAM角色管理页面。

5. 单击新建RAM角色后的添加权限，为其添加管理ECS弹性网卡的权限。

6. 选择完成后，单击确定。



7. 进入相应的RAM角色，查看ARN。



创建专有网络和交换机

创建专有网络和交换机需要在App Studio相同的区域进行，此处以上海区域为例。

登录*VPC*控制台创建专有网络，具体操作请参见*#unique_31*。

说明:

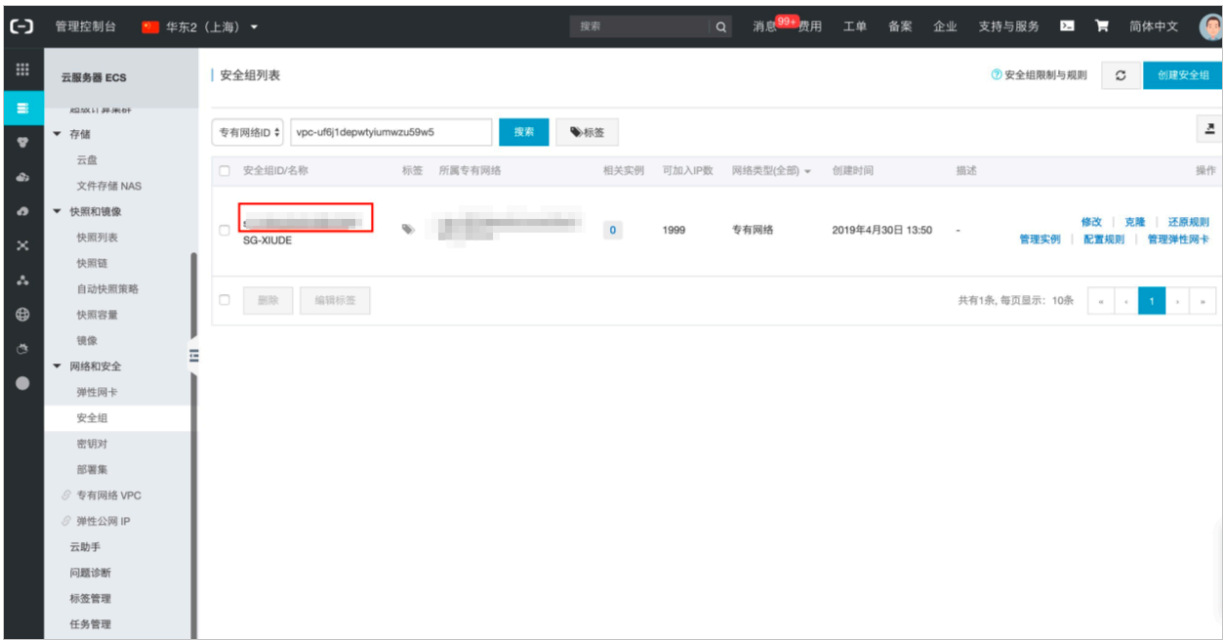
专有网络的IPv4网段需要选择与部署应用前选择的网段不同的网段。

创建完成后，在交换机页面记录下交换机的ID进行备用。

创建安全组

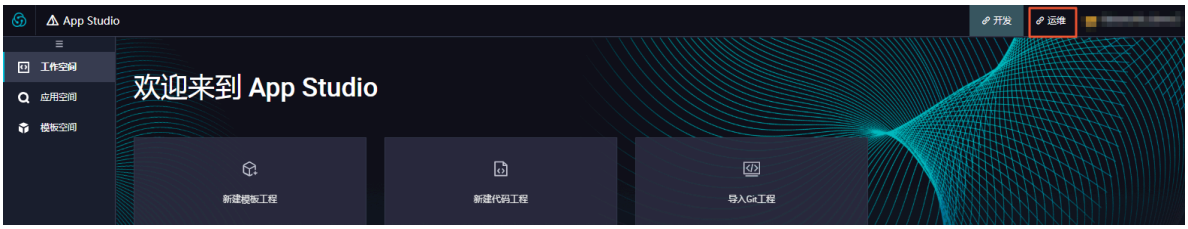
登录*ECS控制台*创建安全组，详细操作请参见*#unique_32*。

安全组创建完成后，请记录安全组的ID进行备用。



在运维平台添加用户VPC

1. **登录DataWorks控制台，进入App Studio页面。**
2. **单击页面右上角的运维。**



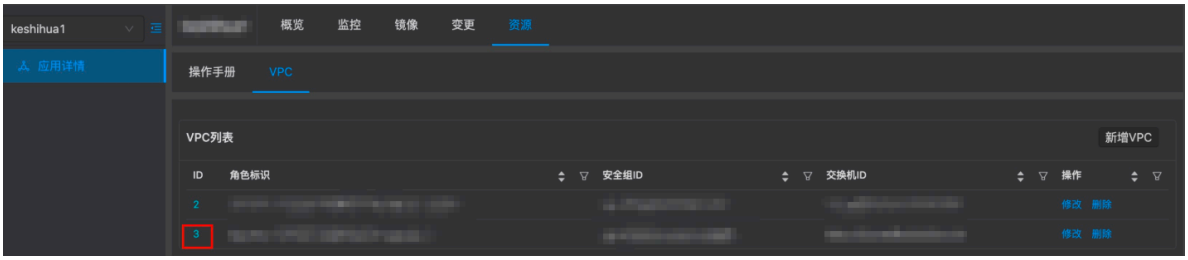
3. 进入资源 > VPC页面，单击新增VPC。

4. 在新增vpc对话框中，填写之前保存的角色标识（即ARN）、安全组ID和交换机ID，并进行相应的描述。

5. 配置完成后，单击执行。

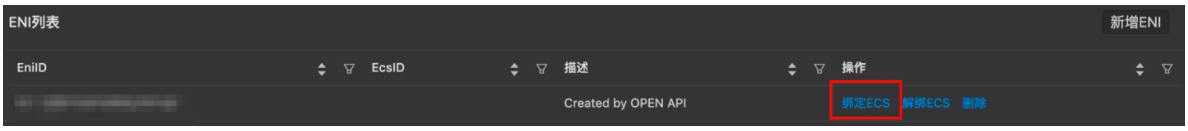
创建弹性网卡并绑定ECS

1. 单击相应VPC的ID，进入ENI管理页面。



2. 单击新增ENI。

3. 新增完成后，单击绑定ECS。



4. 在绑定ecs对话框中，选择相应的VpcID、EniID、分组和机器。

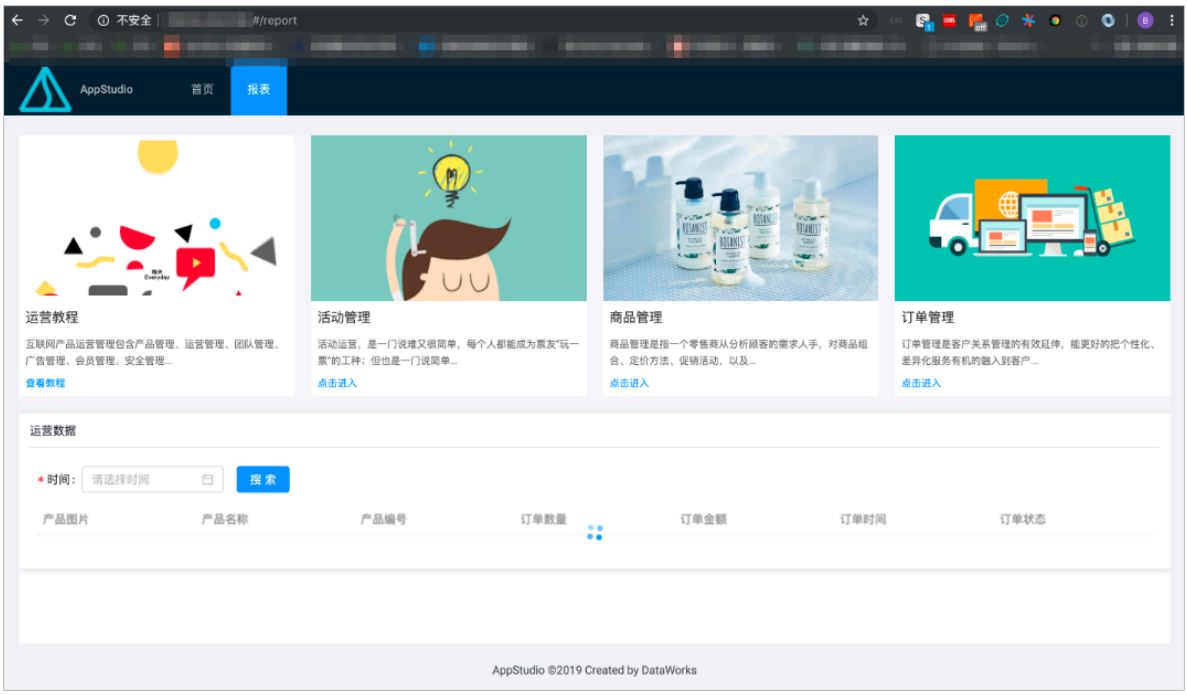
完成上述操作后，App Studio会为您创建弹性网卡，并绑定到机器实例。

公网访问

接下来，您可以通过将弹性网卡绑定至弹性公网IP的方式，将应用透出至公网。您也可以在其中加入负载均衡的服务。

通过弹性公网IP将应用透出至公网的操作，如下所示。

1. 登录VPC控制台购买弹性公网IP，具体操作请参见#unique_33。
2. 绑定弹性网卡，具体操作请参见绑定弹性网卡。
3. 完成上述操作后，即可通过公网IP访问您的服务。



1.4.8 第三方服务接入

1.4.8.1 数据服务

本文将为您介绍如何在App Studio中查看用户有权限调用的数据服务，并通过App Studio生成快速访问数据服务API的代码片。

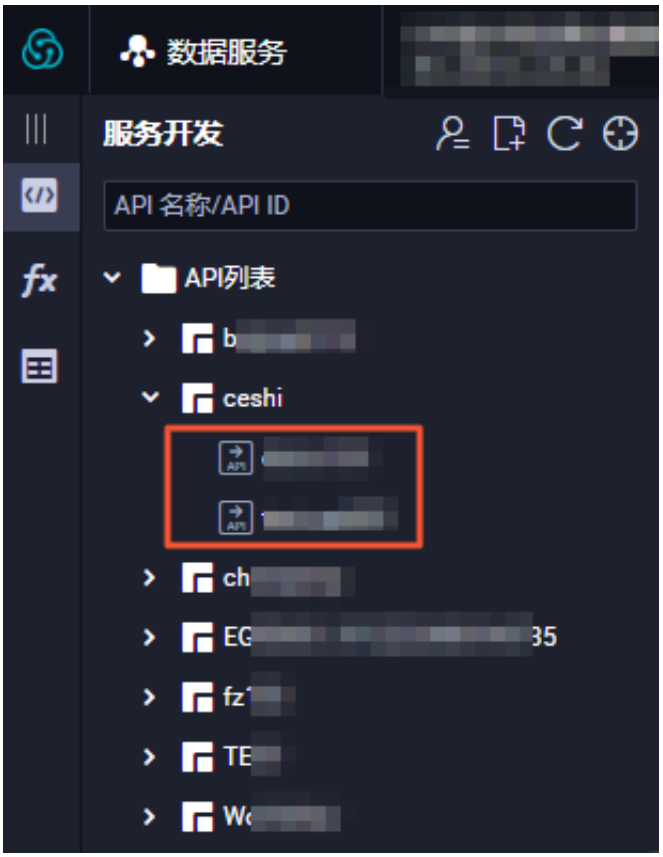
如果您想要获取更多数据服务API的申请、SDK以及调用方法，请参见数据服务。

准备工作

在开始操作前，您需要首先准备以下内容：

- 确认在数据服务中有相关工作空间的权限和API。

  由于本文内容适用于有权限的数据服务API，所以请首先进入数据服务页面，查看是否有DataWorks工作空间，并查看相应工作空间下是否存在有权限的API。

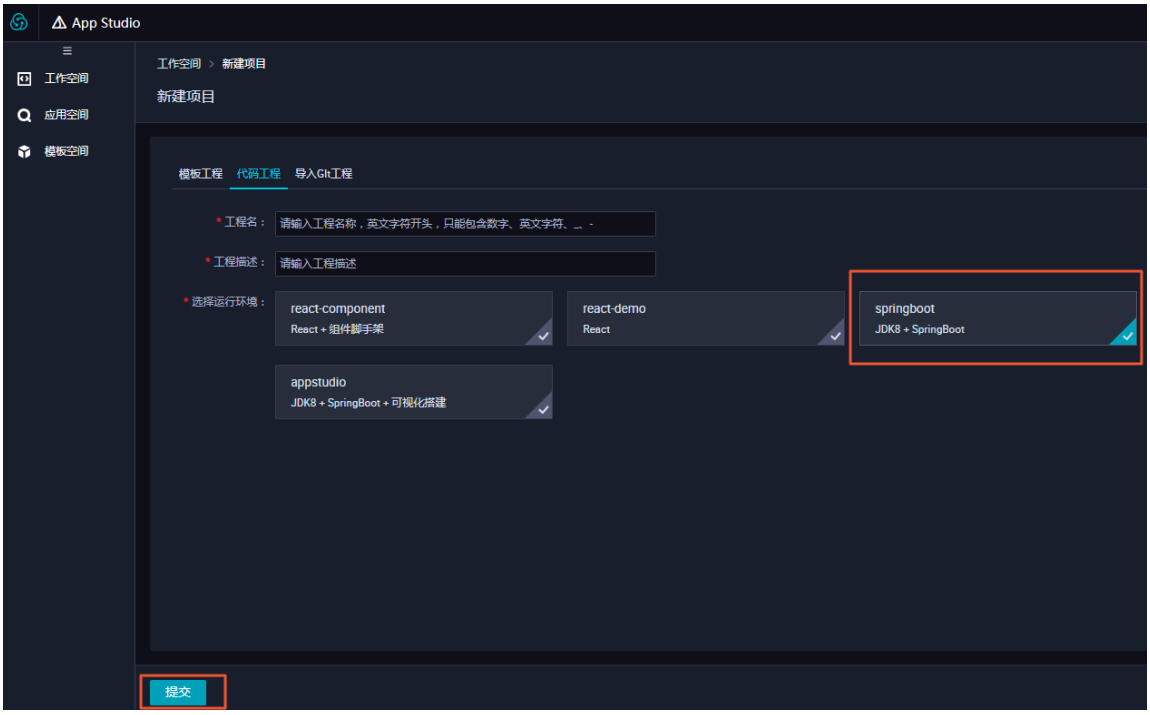


- **在App Studio页面准备一个Java项目。**

  以springboot类型的项目为例，为您介绍代码片生成的功能。

  1. 进入App Studio页面，单击工作空间页面的新建代码工程。

  2. 填写新建项目对话框中的工程名和工程描述，选择运行环境为springboot。



  3. 配置完成后，单击提交。

  项目创建完成后，请确保pom.xml中有数据服务的依赖，maven坐标为*Nexus Repository Manager*。

```
<dependency>
  <groupId>com.aliyun.dataworks</groupId>
  <artifactId>aliyun-dataworks-dataservice-java-sdk</artifactId>
  <version>0.0.1-aliyun</version>
</dependency>
```

在App Studio中使用数据服务

您可以直接在代码中使用数据服务，也可以在可视化搭建中使用数据服务。

- 直接在代码中使用数据服务。

  此步骤将为您介绍如何在App Studio中方便地根据关键字、项目和业务分组查看可用的数据服务，同时利用生成代码片的功能快速生成，并调用某个数据服务API的代码。

  1. 查看数据服务API列表。

     单击App Studio页面右侧的Data，为您展示数据服务API列表。支持根据API名称、工作空间和服务分组进行筛选。

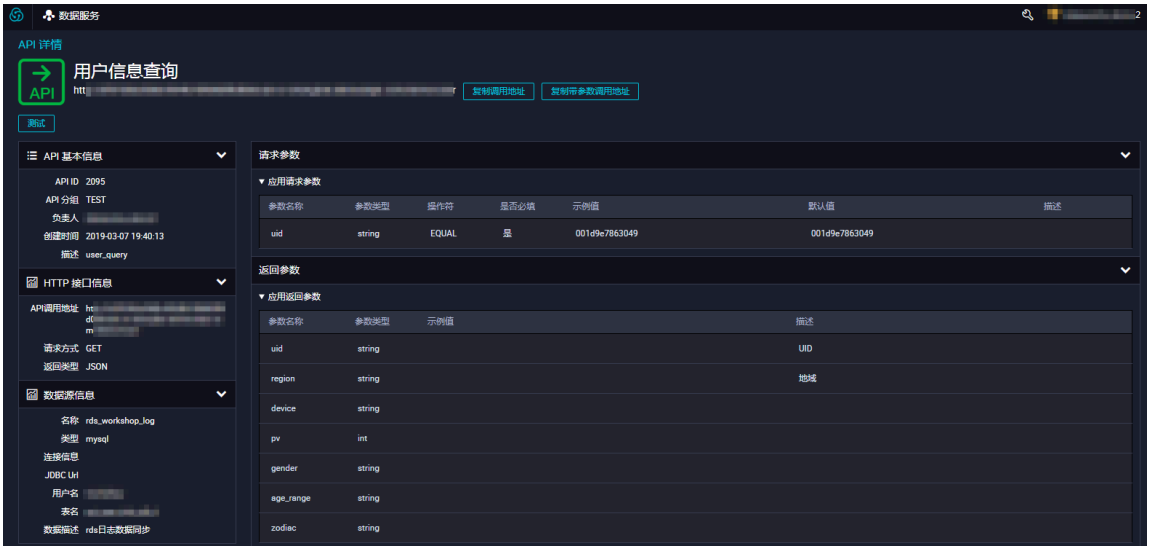
     

  2. 在数据服务页面新增API。

     单击右上角的前往DataService新增API，跳转至数据服务页面新增API，以满足调用API的需求。

  3. 查看数据服务API详情。

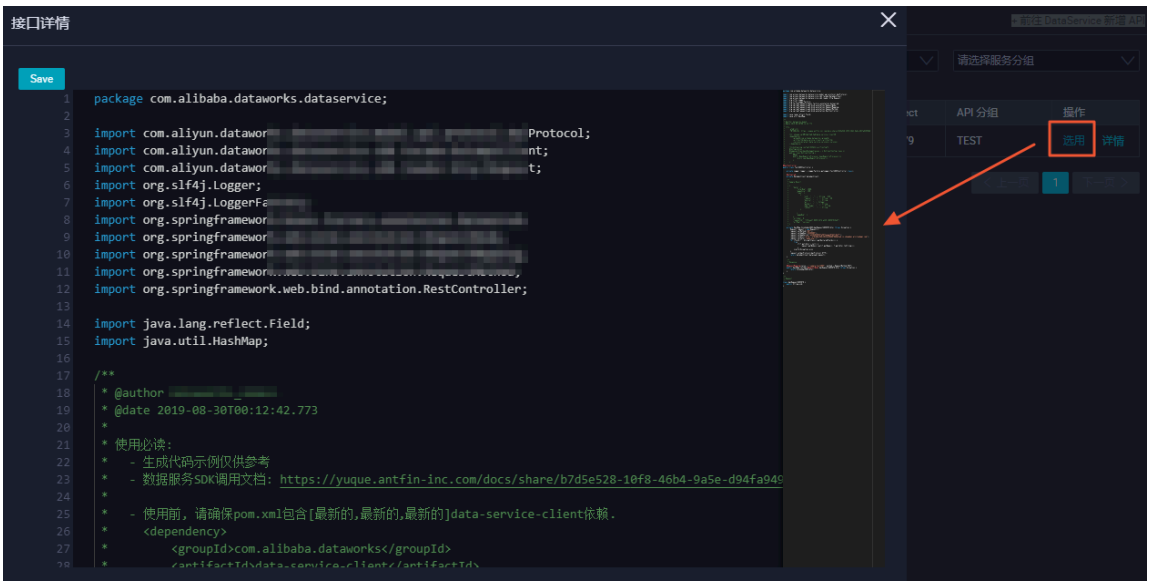
     单击相应数据服务API右侧的详情，即可跳转至数据服务页面查看API详情。



4. 快速生成访问代码。

App Studio支持一键生成访问代码的方式， 自动填充appkey、appsecret，生成样例代码，方便您直接插入项目。

单击相应数据服务API右侧的选用，即可打开包含样例访问代码的详情页。



完整的controller示例如下所示，仅供参考。在生成的InvokeApi2252方法中，您访问这个数据服务需要的path、host、key和secret都会被自动填充，ApiRequest2252DTO则包含了访问该服务的所有参数。

```
package com.alibaba.dataworks.dataservice;
import com.aliyun.dataworks.dataservice.model.api.protocol.
ApiProtocol;
import com.aliyun.dataworks.dataservice.sdk.facade.DataApiClient;
import com.aliyun.dataworks.dataservice.sdk.loader.http.Request;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import java.lang.reflect.Field;
import java.util.HashMap;
/**
 * @author ****
 * @date 2019-03-21T17:23:17.040
 * - 使用前，请确保pom.xml包含最新的data-service-client依赖。
 *     <dependency>
 *         <groupId>com.alibaba.dataworks</groupId>
 *         <artifactId>data-service-client</artifactId>
 *         <version>${latest-data-service-version}</version>
 *     </dependency>
 *  - 使用前，确保配置spring config类，需要单独配置，不可与其他config合
并。
 *     @Configuration
 *     @ComponentScan(basePackageClasses = { DsClientConfig.class
 })
 *     public class DsClientConfig {
 *         @Bean
 *         public BeanRegistryProcessor beanRegistryProcessor(){
 *             return new BeanRegistryProcessor();
 *         }
 *     }
 */
@RestController
public class Test2252Controller {
    private Logger logger = LoggerFactory.getLogger(Test2252Co
ntroller.class);
    @Autowired
    private DataApiClient dataApiClient;
    /**
     * Sample Result:
     * {
     *     "data": {
     *         "totalNum": 1000,
     *         "pageSize": 100,
     *         "rows": [
     *             {
     *                 "pageNum": "...", // 分页默认参数：页编号，
Integer类型。
     *                 "pageSize": "...", // 分页默认参数：页大小，
Integer类型。
     *                 "totalNum": "...", // 分页默认参数：总记录数，
Integer类型。
     *                 "id": "...", // Integer类型。
     *                 "name": "...", // String类型。
     *                 "sex": "...", // String类型。
     *                 "age": "...", // Integer类型。
     *             }
     *             ......
     *         ],
     *         "pageNum": 1
     *     },
     *     "errCode": 0,
     *     "requestId": "478cae2f-0***-42fb-a439-c0***e6f",
     *     "errMsg": "success"
     * }
     */
    private HashMap InvokeApi2252(ApiRequest2252DTO dto) throws
Exception {
        Request request = new Request();
        request.setMethod("GET");
```

```
        request.setAppKey("15810204");
        request.setAppSecret("******************************");
        request.setHost("http://0e5e6cd70******5e64****hai.a***pi.
com");
        request.setPath("/test");
        for (Field f : dto.getClass().getDeclaredFields()) {
            try{
                if(f.get(dto)!= null) {
                    request.getBodys().put(f.getName(), f.get(dto
).toString());
                }
            }catch(Exception e){}
        }
        request.setApiProtocol(ApiProtocol.HTTP);
        return dataApiClient.dataLoad(request);
    }
    /**
     * Response:
     */
    @RequestMapping(value = "/sample/test2252", method =
RequestMethod.POST)
    public HashMap testApi(@RequestBody ApiRequest2252DTO dto)
throws Exception {
        return InvokeApi2252(dto);
    }
}
/**
 * Request
 */
class ApiRequest2252DTO {
    public Integer pageNum;
    public Integer pageSize;
    public Integer id;
    public String name;
    public String sex;
    public Integer age;
}
```

说明:

您可以参考生成的代码样例，也可以直接单击保存，将代码添加到当前代码目录的dataservice包中。

- **在可视化搭建中使用数据服务。**

  可视化搭建的组件和数据服务接口进行了深度的融合，数据服务返回数据的默认格式，即为可视化组件接收数据的格式。可以实现即配即用，详情请参见可视化搭建。

1.4.8.2 DataOS API

本文将为您介绍DataOS API的功能、输入、输出等详情，以及如何进行配置使用。

CheckMetaTable

- **功能：判断table是否存在。**
- **输入：tableGuid（必选）。**
- **格式：** `odps.<project>.<table>`。

- **输出：true或false。**
- **示例如下：**
  - **输入：**

```
request.setTableGuid("odps.autotest.daily_test");
```

  - **输出：**

```
{"requestId":"0b85c9d915548770462378104e","errMsg":"success","
errCode":0,"data":true}
```

GetMetaDB

- **功能：获取MaxCompute项目的信息。**
- **输入：项目GUID（必选）。**
- **格式：** `odps.<project>`。
- **输出：项目详情。**

参数	描述
appGuid	**项目唯一标识**
project	**项目英文名称**
projectNameCn	**项目名称**
comment	**备注**
ownerId	**Owner的ID**
createTime	**创建时间**
modifyTime	**修改时间**

- **示例如下：**
  - **输入：**

```
request.setDbGuid("odps.autotest");
```

  - **输出：**

```
{
    "requestId": "0bfaefec61500****",
    "errMsg": "success",
    "errCode": 0,
    "data": {
        "appGuid": "odps.meta",
        "projectName": "meta",
        "projectNameCn": "ODPS元仓",
        "comment": "",
        "ownerId": "13101879118",
        "createTime": "2014-02-18",
```

```
        "modifyTime": "2018-04-16"
    }
}
```

GetMetaTable

- **功能：获取MaxCompute表的信息。**
- **输入：tableGuid（必选）。**
- **格式：**`odps.<project>.<table>`。
- **输出：表的详情。**

参数	描述
appGuid	**项目唯一标识**
tableGuid	**表唯一标识**
tableName	**表名称**
id	**数据库ID**
ownerId	**owner的ID**
hasPart	**是否为分区表**
dataSize	**表数据的大小**
createTime	**表的创建时间**
lastDdlTime	**表DDL最后的更新时间**
lastModifyTime	**表最后的修改时间**

- **示例如下：**
  - **输入：**

```
request.setTableGuid(tableGuid);
```

  - **输出：**

```
{
    "requestId": "0b8906da817****",
    "errMsg": "success",
    "errCode": 0,
    "data": {
        "appGuid": "odps.meta",
        "tableGuid": "odps.meta.m_table",
        "tableName": "m_table",
        "id": 64809,
        "OwnerId": "dp-base-odps@aliyun-test.com",
        "hasPart": 1,
        "dataSize": 49397610904693,
        "createTime": "2014-12-10 21:20:23",
        "lastDdlTime": "2017-04-18 10:10:06",
        "lastModifyTime": "2019-04-09 20:24:08"
```

```
  }
}
```

ListMetaTableColumn

- **功能：获取MaxCompute的列信息。**
- **输入：tableGuid（必选）。**
- **格式：**`odps.<project>.<table>`。
- **输出：列详情。**

参数	描述
appGuid	**项目唯一标识**
tableGuid	**表唯一标识**
tableName	**表名称**
columnGuid	**列唯一标识，格式为**`odps.<project>.<table>.<col>`
columnName	**列名**
columnType	**列类型**
seqNumber	**列编号（从1开始）**
isPartitionCol	**是否为分区列**
comment	**备注**
safeLevel	**安全等级**

- **示例如下：**
  - **输入：**

```
request.setTableGuid(tableGuid);
```

  - **输出：**

```
{
    "requestId": "0b8906d9796*****",
    "errCode": 0,
    "errMsg": "success",
    "columnList": [{
        "appGuid": "odps.meta",
        "tableGuid": "odps.meta.m_table",
        "tableName": "m_table",
        "columnGuid": "odps.meta.m_table.project_name",
        "columnName": "project_name",
        "columnType": "string",
        "seqNumber": 1,
        "isPartitionCol": 0,
        "comment": "Project名称",
        "safeLevel": "C2"
```

```
  },
{
    "appGuid": "odps.meta",
    "tableGuid": "odps.meta.m_table",
    "tableName": "m_table",
    "columnGuid": "odps.meta.m_table.name",
    "columnName": "name",
    "columnType": "string",
    "seqNumber": 2,
    "isPartitionCol": 0,
    "isPrimaryKey": 0,
    "isNullable": 0,
    "comment": "表名",
    "safeLevel": "C2"
  } ... ]
}
```

ListMetaTablePartition

- **功能：获取MaxCompute的分区信息。**
- **输入：**

参数	说明
tableGuid	格式为odps.<project>.<table>
pageNum	页码
pageSize	每页最多显示记录数

- **输出：表分区的详情。**

参数	描述
appGuid	项目唯一标识
tableGuid	表唯一标识
tableName	表名称
partitionGuid	分区唯一标识，格式为odps.<project>.<table>.<partition>
partitionName	分区名称
createTime	分区的创建时间
modifyTime	分区的修改时间
dataSize	分区的数据大小
records	分区的记录数
pageNum	当前分页页码
pageSize	当前分页大小

参数	描述
totalNum	总记录数

- 返回示例如下：

```
{
    "requestId": "0baf3e0502****",
    "errCode": 0,
    "errMsg": "success",
    "pageNum": 1,
    "pageSize": 10,
    "totalNum": 1101,
    "partitionList": [{
        "appGuid": "odps.meta",
        "tableGuid": "odps.meta.m_table",
        "tableName": "m_table",
        "id": 168504514,
        "partitionGuid": "odps.meta.m_table.ds\u003d20190408",
        "partitionName": "ds\u003d20190408",
        "createTime": "2019-04-08 13:59:52",
        "modifyTime": "2019-04-08 19:54:51",
        "dataSize": 273248012568,
        "records": 720503170
  } ... ]
}
```

SearchMetaTables

- 功能：模糊查找表。
- 输入：

参数	说明
keyword	表名称的关键字
pageNum	页码
pageSize	每页最多显示的记录数

- 输出：

参数	描述
appGuid	项目唯一标识
tableGuid	表唯一标识
tableName	表名称
ownerId	Owner的ID
createTime	表的创建时间
lastDdlTime	表DDL最后的更新时间
lastModifyTime	表最后的修改时间

- **示例如下：**
  - **输入：**

```
request.setKeyword("test");
```

  - **输出：**

```
{
    "message": null,
    "code": 200,
    "success": true,
    "data": {
        "requestId": "0be41b2227759****",
        "errCode": 0,
        "errMsg": "success",
        "pageNum": 1,
        "pageSize": 2,
        "totalNum": 5000,
    "data": [{
        "appGuid": null,
        "tableGuid": "odps.ant_p13n.finance_newsrec_tab_dataset_ds
",
        "tableName": "finance_newsrec_tab_dataset_ds",
        "createTime": "2018-07-06 16:24:41",
        "lastModifyTime": "2019-04-26 10:49:23",
        "lastDdlTime": null,
        "lastAccessTime": null,
        "ownerId": "163585"
    },
    {
        "appGuid": null,
        "tableGuid": "odps.tbcdm.dws_tm_itm_cate_food_ftr_test_cm
",
        "tableName": "dws_tm_itm_cate_food_ftr_test_cm",
        "createTime": "2017-11-23 17:06:18",
        "lastModifyTime": "2019-04-26 20:34:12",
        "lastDdlTime": null,
        "lastAccessTime": null,
        "ownerId": "108292"
    }]
  },
    "timestamp": 1556452227875,
    "sessionId": null
}
```

GetDQCEntity

- **功能：获取分区表达式信息。**
- **请求参数：**

参数	类型	是否必选	示例值	描述
EnvType	STRING	否	MaxCompute	项目类型，包括MaxCompute、Hive和Streaming。

参数	类型	是否必选	示例值	描述
MatchExpression	STRING	否	dt=${yyyymmdd-1}	分区表达式，可以不填写。如果不填写，则返回表下的所有分区表达式。
ProjectName	STRING	否	autotest	MaxCompute项目名称。
TableName	STRING	否	test_dqc_decimal_1119_2	表名称。

- 返回参数：

参数	类型	示例值	描述
ReturnCode	STRING	0	返回码
ReturnValue	-	-	返回的分区表达式列表
Id	LONG	4003918	分区表达式ID
ProjectName	STRING	autotest	MaxCompute项目名称
TableName	STRING	test_dqc_decimal_1119_2	表名称
EnvType	STRING	MaxCompute	项目类型，包括MaxCompute、Hive和Streaming
MatchExpression	STRING	dt=${yyyymmdd-1}	分区表达式
EntityLevel	INTEGER	1	分区表达式级别 - 0：SQL级别 - 1：任务级别
OnDuty	STRING	50624	分区表达式责任人
ModifyUser	STRING	50624	最近修改人
GmtCreate	STRING	2018-11-26 23:18:34	创建时间
GmtModify	STRING	2018-11-26 23:18:34	修改时间

参数	类型	示例值	描述
Sql	INTEGER	0	是否支持修改为SQL级别 - 0：支持 - -1：不支持
Task	INTEGER	0	是否支持修改为任务级别 - 0：支持 - -1：不支持
Followers	STRING	050624	订阅人，逗号分隔
HasRelativeNode	BOOLEAN	false	是否关联调度

- **请求示例：**

```
/?Action=GetDQCEntity
&EnvType=odps
&MatchExpression=dt=${yyyymmdd-1}
&ProjectName=autotest
&TableName=test_dqc_decimal_1119_2
&<公共请求参数>
```

- **正常返回示例：**

```
{
  "ReturnCode":"0",
  "ReturnValue":{
    "Entity":[
      {
        "EntityLevel":0,
        "EnvType":"odps",
        "Followers":"050624",
        "GmtCreate":"2018-11-26 15:06:32",
        "GmtModify":"2018-11-26 15:06:32",
        "HasRelativeNode":false,
        "Id":4003918,
        "MatchExpression":"dt=${yyyymmdd-3}",
        "OnDuty":"050624",
        "ProjectName":"autotest",
        "Sql":0,
        "TableName":"test_dqc_decimal_1119_2",
        "Task":0
      },
      {
        "EntityLevel":0,
        "EnvType":"odps",
        "Followers":"050624",
        "GmtCreate":"2018-11-26 22:31:13",
        "GmtModify":"2018-11-26 22:31:13",
        "HasRelativeNode":false,
        "Id":4003922,
        "MatchExpression":"dt=${yyyymmdd-1}",
        "OnDuty":"050624",
        "ProjectName":"autotest",
```

```
        "Sql":0,
        "TableName":"test_dqc_decimal_1119_2",
        "Task":0
      },
      {
        "EntityLevel":0,
        "EnvType":"odps",
        "Followers":"050624",
        "GmtCreate":"2018-11-26 23:18:34",
        "GmtModify":"2018-11-26 23:18:34",
        "HasRelativeNode":false,
        "Id":4003923,
        "MatchExpression":"dt=${yyyymmdd-2}",
        "OnDuty":"050624",
        "ProjectName":"autotest",
        "Sql":0,
        "TableName":"test_dqc_decimal_1119_2",
        "Task":0
      }
    ]
  }
}
```

- **异常返回示例：**

```
{
  "ReturnCode":"500"
}
```

GetDQCFollower

- **功能：获取告警订阅信息。**
- **请求参数：**

参数	类型	是否必选	示例值	描述
EntityId	LONG	否	4003922	分区表达式ID
ProjectName	STRING	否	MaxCompute	MaxCompute项目名称

- **返回参数：**

参数	类型	示例值	描述
ReturnCode	STRING	0	返回码
ReturnValue	-	-	返回的订阅列表
Id	LONG	4003918	分区表达式ID
ProjectName	STRING	autotest	MaxCompute项目名称
TableName	STRING	test_dqc_decimal_1119_2	表名称

参数	类型	示例值	描述
EntityId	STRING	4003922	分区表达式ID
Followers	STRING	050624	订阅人，逗号分隔
AlarmMode	INTEGER	1	包括以下4种告警模式： - 1：邮件 - 2：邮件和短信 - 3：钉钉群机器人/hook - 4：钉钉群机器人@ALL

- 请求示例：

```
/?Action=GetDQCFollower
&EntityId=4003922
&ProjectName=odps
&<公共请求参数>
```

- 正常返回示例：

```
{
  "ReturnCode":"0",
  "ReturnValue":{
    "Follower":[{
      "AlarmMode":1,
      "EntityId":"4003922",
      "Follower":"050624",
      "Id":1726,
      "ProjectName":"autotest",
      "TableName":"test_dqc_decimal_1119_2"
    }]
  }
}
```

- 异常返回示例：

```
{
  "ReturnCode":"500"
}
```

GetDQCRule

- 功能：获取规则详情。
- 请求参数：

参数	类型	是否必选	示例值	描述
ProjectName	STRING	否	autotest	MaxCompute项目名称

参数	类型	是否必选	示例值	描述
EntityId	LONG	否	4003922	分区表达式ID

- 返回参数：

一级参数	二级参数	三级参数	类型	示例值	描述
ReturnCode	-	-	STRING	0	返回码
ReturnValue（返回的规则列表）	TemplateRules（模板规则列表）	Id	INTEGER	4003918	分区表达式ID
		ProjectName	STRING	autotest	MaxCompute项目名称
		TableName	STRING	test_dqc_decimal_1119_2	表名称
		EntityId	INTEGER	4003922	分区表达式ID
		Property	STRING	table_count	参数
		MethodId	INTEGER	8	采集方法ID
		MethodName	STRING	table_count	采集方法名称
		OnDuty	STRING	050624	规则配置人
		RuleType	INTEGER	0	规则类型
		BlockType	INTEGER	1	强弱性 - 0：弱规则 - 1：强规则
		TemplateId	INTEGER	7	模板ID
		TemplateName	STRING	SQL任务表行数，1、7、30天波动检测	模板名称
		RuleCheckerRelationId	INTEGER	1008007	规则内部关联ID
		CheckerId	INTEGER	7	校检器ID
		FixCheck	BOOLEAN	false	是否是固定值校验
		Trend	STRING	up	趋势

一级参数	二级参数	三级参数	类型	示例值	描述
		WarningThreshold	STRING	20	橙色阈值
		CriticalThreshold	STRING	90	红色阈值
		HistoryWarningThreshold	STRING	history max:40%, history min:10%	历史橙色告警阈值
		HistoryCriticalThreshold	STRING	history max:40%, history min:10%	历史红色告警阈值
		PropertyKey	STRING	table_count	用于DQC前端规则联动，可以忽略
		MatchExpression	STRING	dt=${yyyymmdd-1}	分区表达式
	SelfserviceRules（自定义规则列表）	ProjectName	STRING	autotest	MaxCompute项目名称
		TableName	STRING	test_dqc_decimal_1119_2	表名
		Id	INTEGER	279580664	规则ID
		EntityId	INTEGER	4003922	分区表达式ID
		Property	STRING	table_count	参数
		MethodId	INTEGER	21	采集方法ID
		MethodName	STRING	count或table_count	采集方法名称
		WhereCondition	STRING	id>10	过滤条件或自定义SQL
		OnDuty	STRING	050624	规则配置人
		RuleType	INTEGER	1	规则类型

一级参数	二级参数	三级参数	类型	示例值	描述
		BlockType	INTEGER	1	强弱性： - 0：弱规则 - 1：强规则
		TemplateId	INTEGER	7	模板ID
		TemplateName	STRING	SQL任务表行数，1、7、30天波动检测	模板名称
		RuleCheckerRelationId	INTEGER	1008006	数据质量监控的内部规则映射表
		CheckerId	INTEGER	6	校检器ID
		Checker	INTEGER	9	校验器，此参数对应于前端的ID标识，需要由pkId转换
		CheckerName	STRING	compared with a fixed value	校检器名称
		FixCheck	BOOLEAN	false	是否是固定值校验
		Trend	STRING	abs	趋势
		CheckResult	INTEGER	0	规则的校检结果，该接口不返回，可以忽略
		WarningThreshold	STRING	20	橙色阈值
		CriticalThreshold	STRING	60	红色阈值
		HistoryWarningThreshold	STRING	history max:40%, history min:10%	历史橙色告警阈值

一级参数	二级参数	三级参数	类型	示例值	描述
		HistoryCriticalThreshold	STRING	history max:40%, history min:10%	历史红色告警阈值
		HistoryActualThreshold	STRING	history max:10%, history min:10%	历史真实波动率
		PropertyKey	STRING	table_count	用于DQC前端规则联动进行映射，可以忽略
		MatchExpression	STRING	dt=${yyyymmdd-1}	分区表达式

GetQualityByEntity

- 功能：根据分区表达式获取质量汇总信息。
- 请求参数：

参数	类型	是否必选
Action	STRING	是
EndDate	STRING	是
EntityId	INTEGER	是
PageSize	INTEGER	是
PageStart	INTEGER	是
ProjectName	STRING	是
StartDate	STRING	是

- 返回参数：

参数	类型	描述
Count	INTEGER	搜索结果的总行数
ReturnCode	STRING	返回码
ReturnValue	-	分区下规则的校验详情

- **请求示例：**

```
http(s)://[Endpoint]/?Action=GetQualityByEntity
&EndDate=2019-08-25 00:00:00
&EntityId=1526081
&PageSize=15
&PageStart=1
&ProjectName=autotest
&StartDate=2019-08-20 00:00:00
&<公共请求参数>
```

- **返回示例：**

```
{
  "returnCode": "0",
  "returnMessage": null,
  "returnErrorSolution": null,
  "returnErrorOper": null,
  "requestId": "40911dfc-a936-4092-bfe9-6fcd*****",
  "returnValue": [{
    "id": 787700746,
    "taskId": "15668171746013d72950384444****80",
    "entityId": 1508198,
    "ruleId": 28410562,
    "property": "table_count",
    "bizdate": 1566662400000,
    "dateType": "YMD",
    "actualExpression": "ds=20190826",
    "matchExpression": "ds=${yyyymmdd}",
    "blockType": 1,
    "checkResult": 0,
    "checkResultStatus": 0,
    "methodName": "count",
    "comment": null,
    "whereCondition": "",
    "beginTime": 1566817174000,
    "endTime": 1566817355000,
    "timeConsuming": "181s",
    "externalType": "CWF2",
    "externalId": "100033671",
    "discrete": false,
    "fixedCheck": true,
    "referenceValue": [{
      "bizDate": 32535100800000,
      "discreteProperty": "table_count",
      "value": 0.0,
      "threshold": null,
      "singleCheckResult": 0
    }],
    "sampleValue": [{
      "bizDate": 1566662400000,
      "discreteProperty": null,
      "value": 1306012.0
    }],
    "checkResultDetail": {
      "checkerId": null,
      "checkerType": null,
      "isDiscrete": false,
      "warningThreshold": null,
      "criticalThreshold": null,
      "op": ">",
      "expectValue": 0,
```

```
      "trend": null,
      "externalId": "100033671",
      "status": 0,
      "qualityStatus": 0,
      "actualResult": null,
      "detail": [{
        "currentSample": {
          "bizdate": 1566662400000,
          "property": "table_count",
          "distinctProperty": null,
          "value": 1306012,
          "detail": "1306012"
        },
        "historicalSamples": [{
          "bizdate": null,
          "property": null,
          "distinctProperty": null,
          "value": null,
          "detail": "--",
          "isExisted": null,
          "fluctuatedValue": null,
          "fluctuatedValueType": null,
          "checkResult": null
        }]
      }]
    },
    "trend": "abs",
    "warningThreshold": null,
    "criticalThreshold": null,
    "expectValue": 0.0,
    "op": ">",
    "projectName": null,
    "tableName": null,
    "templateId": null,
    "templateName": null,
    "resultStr": null,
    "checkerId": null,
    "checkerType": 0,
    "ruleName": null,
    "isPrediction": false,
    "upperValue": null,
    "lowerValue": null,
    "checkerName": "与固定值比较"
  }]
}
```


GetQualityByRule

- **功能：根据规则查询校验详情。**
- **请求参数：**

参数	类型	是否必选
Action	**STRING**	**是**
EndDate	**STRING**	**是**
PageSize	**INTEGER**	**是**
PageStart	**INTEGER**	**是**

参数	类型	是否必选
ProjectName	STRING	是
RuleId	INTEGER	是
StartDate	STRING	是

- 返回参数：

参数	类型	描述
Count	INTEGER	数据总条数
ReturnCode	STRING	返回码
ReturnValue	-	返回的校验详情

- 请求示例：

```
http(s)://[Endpoint]/?Action=GetQualityByRule
&EndDate=2019-08-26 00:00:00
&PageSize=15
&PageStart=1
&ProjectName=cdo_meta
&RuleId=28791246
&StartDate=2019-08-22 00:00:00
&<公共请求参数>
```

- 返回示例：

```
{
  "returnCode": "0",
  "returnMessage": null,
  "returnErrorSolution": null,
  "returnErrorOper": null,
  "requestId": "f34d529b-d559-4699-b850-9****b58",
  "returnValue": {
    "ruleCheckDto": [{
      "id": 787723200,
      "taskId": "156681910181215a8b8f9cd154****34e7d",
      "entityId": 1346320,
      "ruleId": 28791246,
      "property": "-",
      "bizdate": 1566662400000,
      "dateType": "YMD",
      "actualExpression": "NOTAPARTITIONTABLE",
      "matchExpression": "NOTAPARTITIONTABLE",
      "blockType": 1,
      "checkResult": 0,
      "checkResultStatus": 0,
      "methodName": "table_count",
      "comment": "监控表行数波动",
      "whereCondition": null,
      "beginTime": 1566819101000,
      "endTime": 1566819103000,
      "timeConsuming": "2s",
      "externalType": "CWF2",
      "externalId": "1607534",
      "discrete": false,
```

```
      "fixedCheck": false,
      "referenceValue": [{
        "bizDate": 32535100800000,
        "discreteProperty": null,
        "value": 3778.0,
        "threshold": 0.1852832186,
        "singleCheckResult": 0
      }, {
        "bizDate": 32535100800000,
        "discreteProperty": null,
        "value": 3772.0,
        "threshold": 0.3446447508,
        "singleCheckResult": 0
      }, {
        "bizDate": 32535100800000,
        "discreteProperty": null,
        "value": 3691.0,
        "threshold": 2.5,
        "singleCheckResult": 0
      }],
      "sampleValue": [{
        "bizDate": 1566662400000,
        "discreteProperty": null,
        "value": 3785.0
      }],
      "checkResultDetail": {
        "checkerId": null,
        "checkerType": null,
        "isDiscrete": false,
        "warningThreshold": 10,
        "criticalThreshold": 50,
        "op": null,
        "expectValue": null,
        "trend": null,
        "externalId": "1607534",
        "status": 0,
        "qualityStatus": 0,
        "actualResult": null,
        "detail": [{
          "currentSample": {
            "bizdate": 1566662400000,
            "property": "-",
            "distinctProperty": null,
            "value": 3785,
            "detail": "3785"
          },
          "historicalSamples": [{
            "bizdate": 32535100800000,
            "property": null,
            "distinctProperty": null,
            "value": 3778,
            "detail": "3778,  0.1852832186%",
            "isExisted": null,
            "fluctuatedValue": 0.185283,
            "fluctuatedValueType": null,
            "checkResult": 0
          }, {
            "bizdate": 32535100800000,
            "property": null,
            "distinctProperty": null,
            "value": 3772,
            "detail": "3772,  0.3446447508%",
            "isExisted": null,
            "fluctuatedValue": 0.344645,
```

```
              "fluctuatedValueType": null,
              "checkResult": 0
            }, {
              "bizdate": 32535100800000,
              "property": null,
              "distinctProperty": null,
              "value": 3691,
              "detail": "3691,  2.5%",
              "isExisted": null,
              "fluctuatedValue": 2.5,
              "fluctuatedValueType": null,
              "checkResult": 0
            }]
          }]
        },
        "trend": "abs",
        "warningThreshold": 10.0,
        "criticalThreshold": 50.0,
        "expectValue": null,
        "op": "abs",
        "projectName": null,
        "tableName": null,
        "templateId": 7,
        "templateName": null,
        "resultStr": null,
        "checkerId": null,
        "checkerType": 1,
        "ruleName": "表行数监控",
        "isPrediction": false,
        "upperValue": null,
        "lowerValue": null,
        "checkerName": null
      }],
      "count": 4325,
      "checkerType": 0
    }
}
```


使用DataOS API

添加maven依赖如下所示。

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-dataworks-enterprise-ultimate</
artifactId>
    <version>0.0.4</version>
</dependency>

<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>

<!-- 要求gson 2.8.5以上版本 -->
    <dependency>
        <groupId>com.aliyun</groupId>
        <artifactId>aliyun-java-sdk-core</artifactId>
        <version>4.4.0</version>
    </dependency>
```

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

Java代码如下所示，其中创建IClientProfile时，需要指定云账号的AccessKeyID和AccessKeySecret，详情请参见下文的常见问题。

说明:

将测试代码直接放在src/test/java目录下，会有UT快速启动入口。

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
CheckMetaTableRequest;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
CheckMetaTableResponse;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
GetMetaDBRequest;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
GetMetaDBResponse;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
GetMetaTableRequest;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
GetMetaTableResponse;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
ListMetaTableColumnRequest;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
ListMetaTableColumnResponse;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
ListMetaTablePartitionRequest;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
ListMetaTablePartitionResponse;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
SearchMetaTablesRequest;
import com.aliyuncs.dataworks_enterprise_ultimate.model.v20190424.
SearchMetaTablesResponse;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import com.google.gson.Gson;

import org.junit.BeforeClass;
import org.junit.Test;

public class Sample {
    private static final String REGION_ID = "cn-shanghai";
    private static IAcsClient client = null;

    @BeforeClass
    public static void setup() throws ClientException {
        String endpoint = "dataworks-ee-ue-share." + REGION_ID + ".
aliyuncs.com";
        IClientProfile profile = DefaultProfile.getProfile(
            REGION_ID,
            "<!!!!your.accessId>",
            "<!!!!your.accessKey>");
```

```
        DefaultProfile.addEndpoint(REGION_ID, "dataworks-enterprise-
ultimate", endpoint);
        client = new DefaultAcsClient(profile);
    }

    @Test
    public void testCheckMetaTable() throws ServerException,
ClientException {
        String tableGuid = "odps.meta.m_table";

        CheckMetaTableRequest request = new CheckMetaTableRequest();
        request.setTableGuid(tableGuid);
        CheckMetaTableResponse response = client.getAcsResponse(
request);
        System.out.println(new Gson().toJson(response));
    }

    @Test
    public void testGetProject() throws ServerException, ClientExce
ption {
        String appGuid = "odps.meta";

        GetMetaDBRequest request = new GetMetaDBRequest();
        request.setDbGuid(appGuid);
        GetMetaDBResponse getMetaDBResponse = client.getAcsResponse(
request);
        System.out.println(new Gson().toJson(getMetaDBResponse));
    }

    @Test
    public void testGetPartitions() throws ServerException, ClientExce
ption {
        String tableGuid = "odps.meta.m_table";

        ListMetaTablePartitionRequest request = new ListMetaTa
blePartitionRequest();
        request.setTableGuid(tableGuid);
        request.setPageNum(1);
        request.setPageSize(10);
        ListMetaTablePartitionResponse response = client.getAcsResp
onse(request);
        System.out.println(new Gson().toJson(response));
    }

    @Test
    public void testSearchTables() throws ServerException, ClientExce
ption {
        SearchMetaTablesRequest request = new SearchMetaTablesRequest
();
        request.setKeyword("test");
        request.setPageNum(1);
        request.setPageSize(10);
        SearchMetaTablesResponse response = client.getAcsResponse(
request);
        System.out.println(new Gson().toJson(response));
    }

    @Test
    public void testGetColumns() throws ServerException, ClientExce
ption {
        String tableGuid = "odps.meta.m_table";

        ListMetaTableColumnRequest request = new ListMetaTableColumnR
equest();
```

```
        request.setTableGuid(tableGuid);
        ListMetaTableColumnResponse response = client.getAcsResponse(
request);
        System.out.println(new Gson().toJson(response));
    }

    @Test
    public void testGetTable() throws ServerException, ClientException
 {
        String tableGuid = "odps.meta.m_table";

        GetMetaTableRequest request = new GetMetaTableRequest();
        request.setTableGuid(tableGuid);
        GetMetaTableResponse response = client.getAcsResponse(request
);
        System.out.println(new Gson().toJson(response));
    }

}
```

常见问题

- **无法访问API，错误提示如下所示：**

  ```
  Exception in thread "main" com.aliyuncs.exceptions.ClientException:
  InvalidApi.NotFound : Specified api is not found, please check your
  url and method.
  RequestId : B081CCF1-9F19-473E-9B99-68F20****
  ```

  错误原因：没有获取API权限。

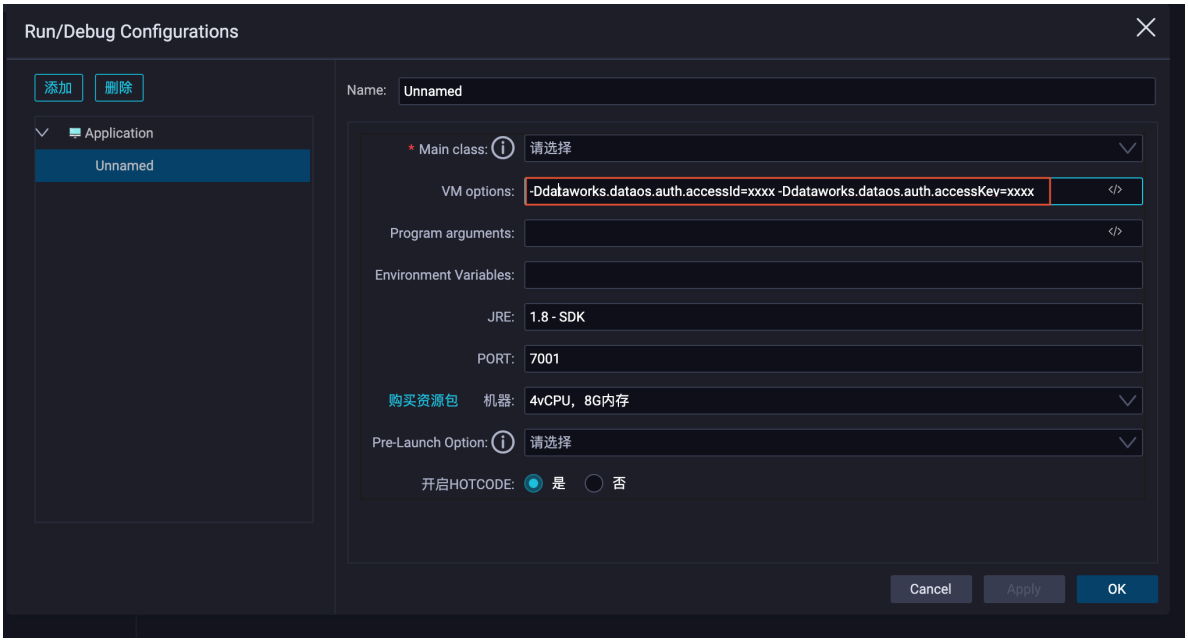
- **如何查询AccessKeyID和AccessKeySecret?**

  单击页面右上角账号下的accesskeys，即可进行查询。

- **调试时启动服务，配置项报错Circular Reference。**

```
java.lang.IllegalArgumentException: Circular placeholder
reference 'dataworks.dataos.auth.accessId' in property
definitions
```

启动时需要添加如下图所示的配置参数。



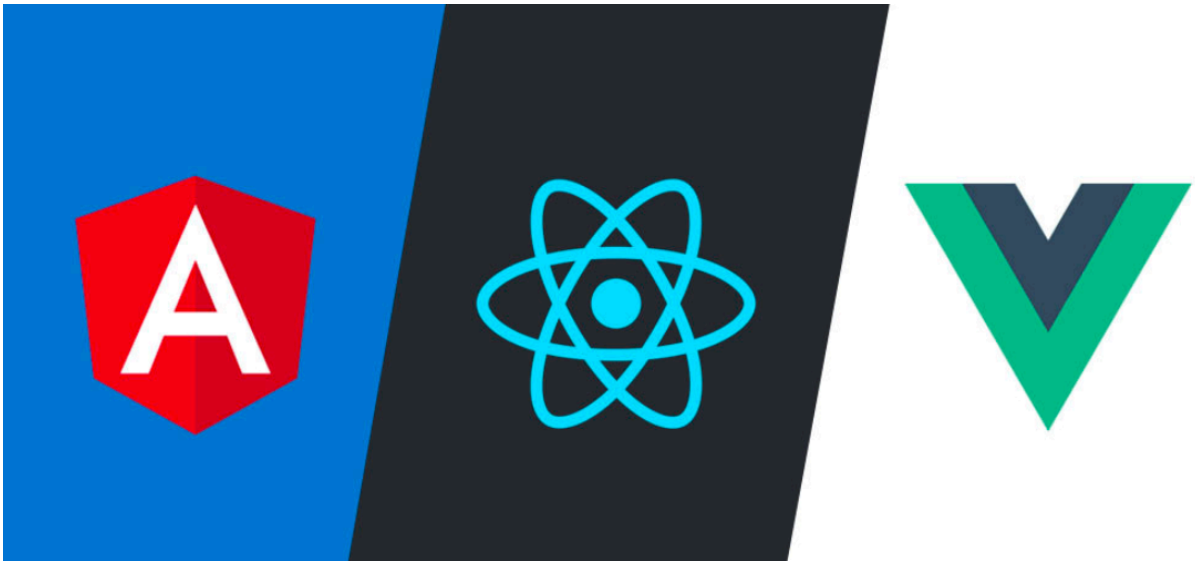
1.4.9 可视化搭建

1.4.9.1 可视化搭建概述

App Studio可视化搭建是辅助生成前端页面的工具，提供了一系列常见的网页组件，让开发者可以通过简单的拖拽，便可生成前端页面。本文将为您介绍App Studio可视化搭建的特点。

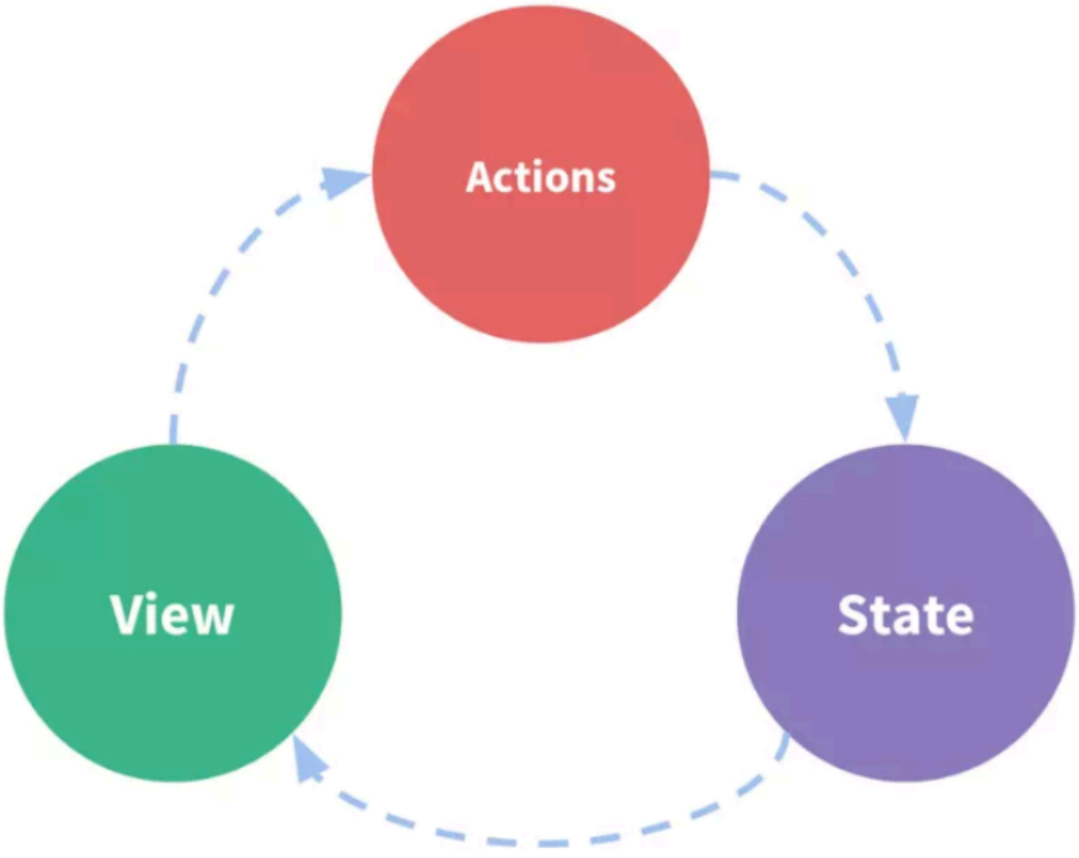
框架无感知

无论您使用React、Angular或Vue，App Studio可视化搭建系统都可以适配，因为底层使用了一种通用的描述语言来描述页面的结构、表现、行为等属性。



集成简单的数据处理来满足复杂交互需求

App Studio集成了一个全局的状态管理方案，来完成页面数据管理以及组件之间的交互。

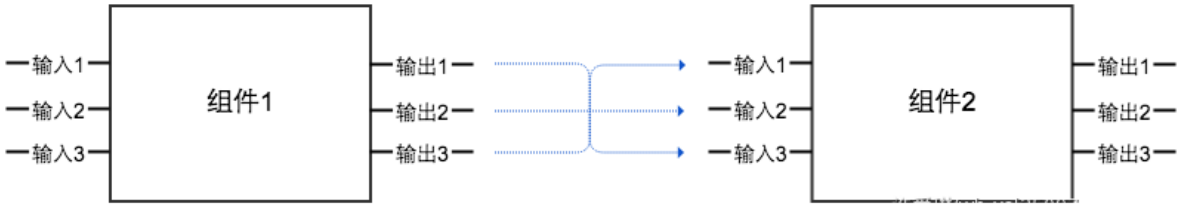


提供代码模式来满足复杂交互页面的搭建

App Studio可视化搭建的底层使用了通用的结构化可描述性语言（DSL）作为中间层。您可以直接基于DSL进行代码模式的修改，实现了代码模式与可视化拖拽模式的互转，对于高阶开发者来说这是一种进阶的使用体验。

可视化方式配置组件联动

App Studio可视化搭建提供了一种非常简单的可视化连线方式，来进行组件之间的交互联动。



无需构建即可直接发布运行

App Studio可以将中间层的DSL在线编译为一份可直接在浏览器中执行的代码，进行页面渲染。

对接DataWorks数据服务，快速集成数据接口

App Studio无缝对接DataWorks数据服务接口，可实时调试接口。

丰富的组件、模板市场

App Studio提供了丰富的组件，同时支持您自定义组件并上传到组件库。

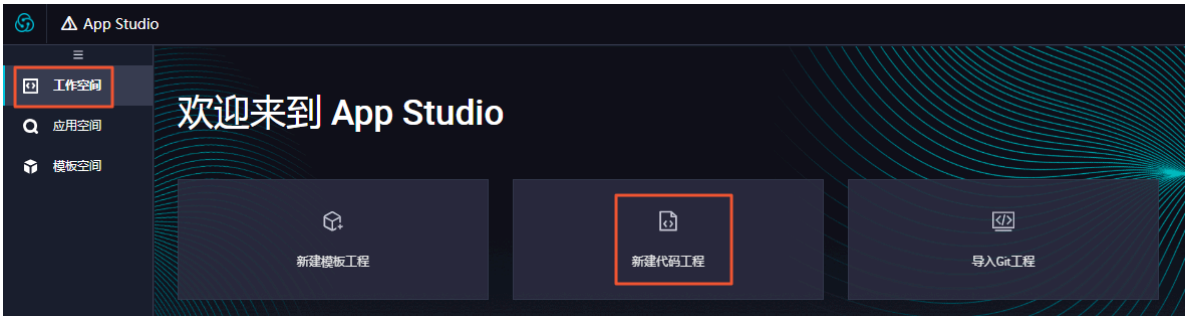
另外，App Studio也提供了丰富的模板，您可以直接基于某一个模板快速生成页面，也可以将页面保存为模板并发布到模板市场供他人使用。

1.4.9.2 基本使用

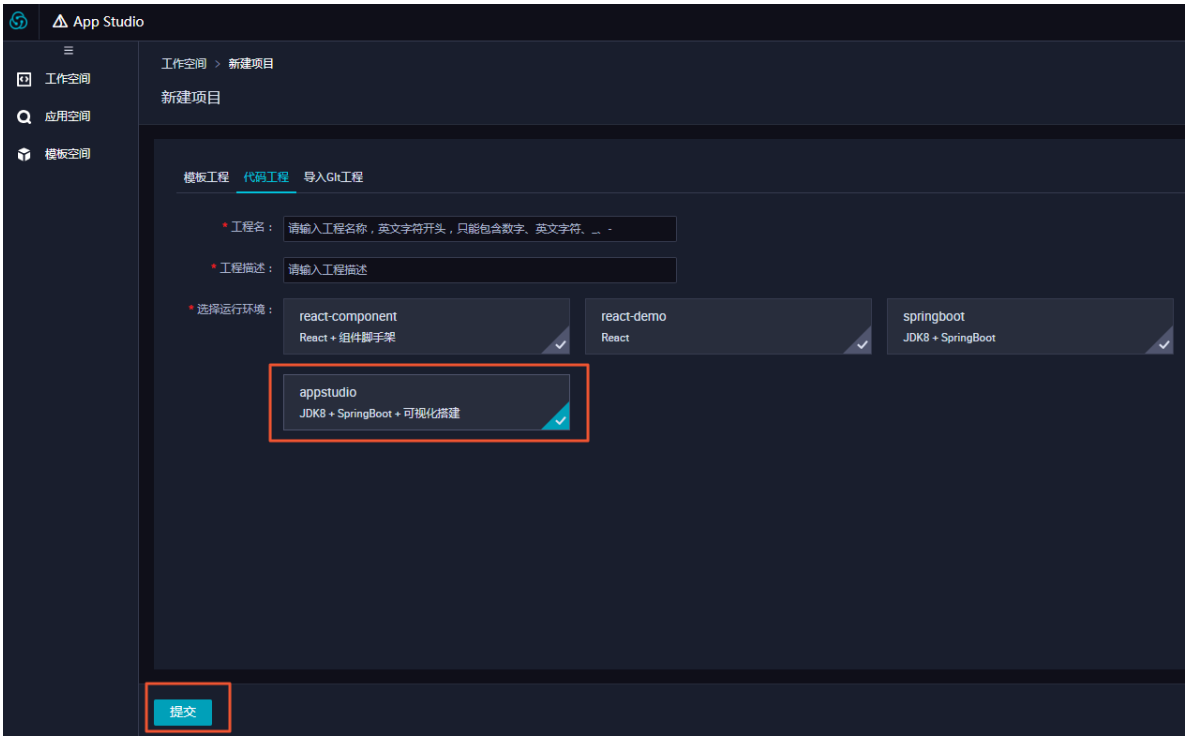
本文将为您介绍可视化搭建系统的新建工程、可视化搭建等基本操作。

新建工程

1. 进入App Studio页面，单击工作空间页面的新建代码工程。

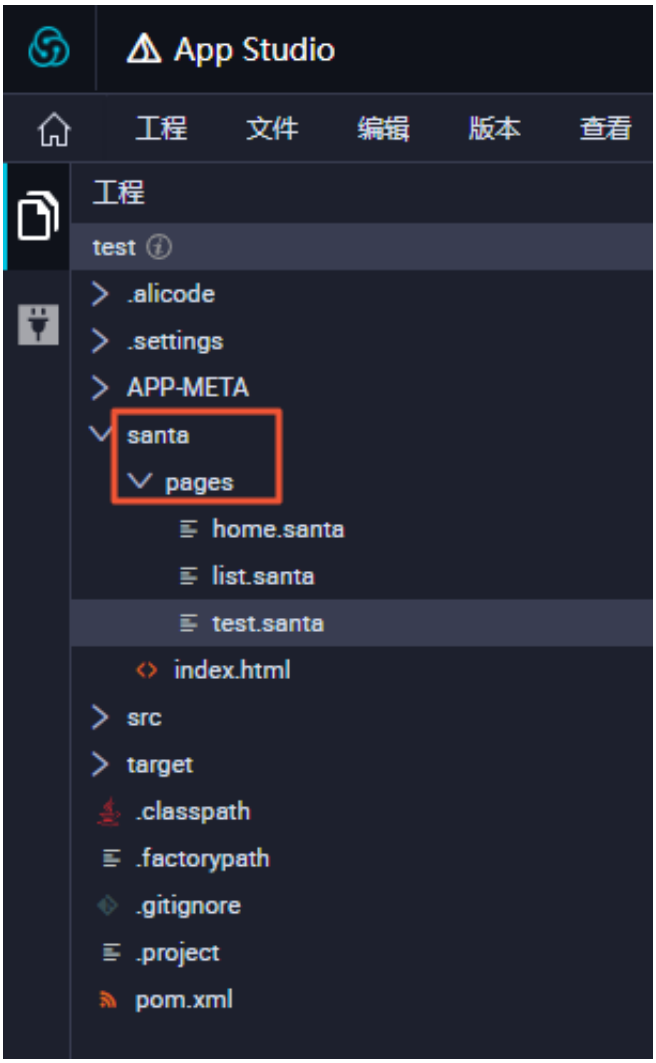


2. 填写新建项目对话框中的工程名和工程描述，选择运行环境为appstudio。



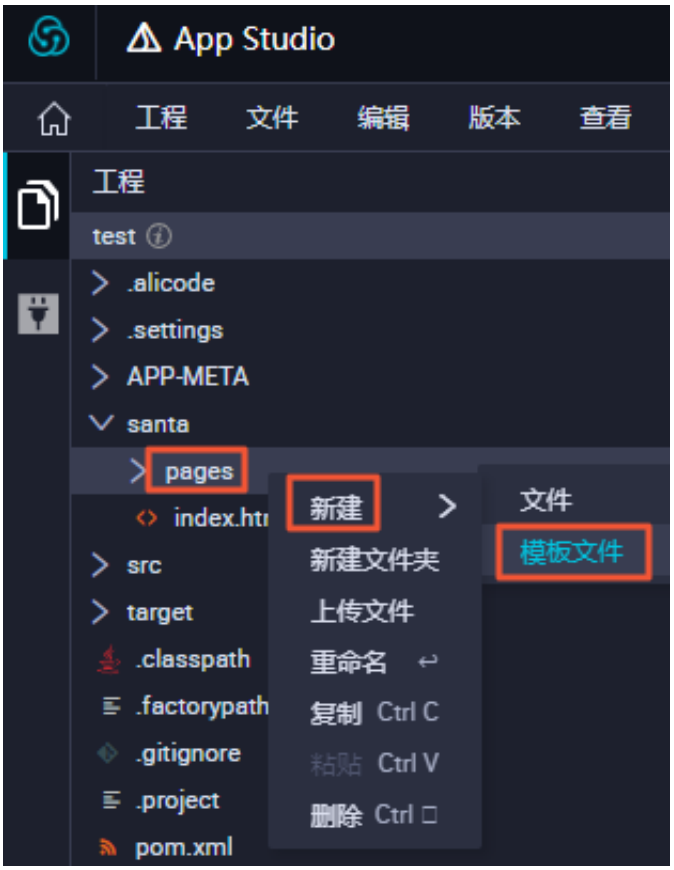
3. 配置完成后，单击提交。

4. **打开***santa/pages***目录。**



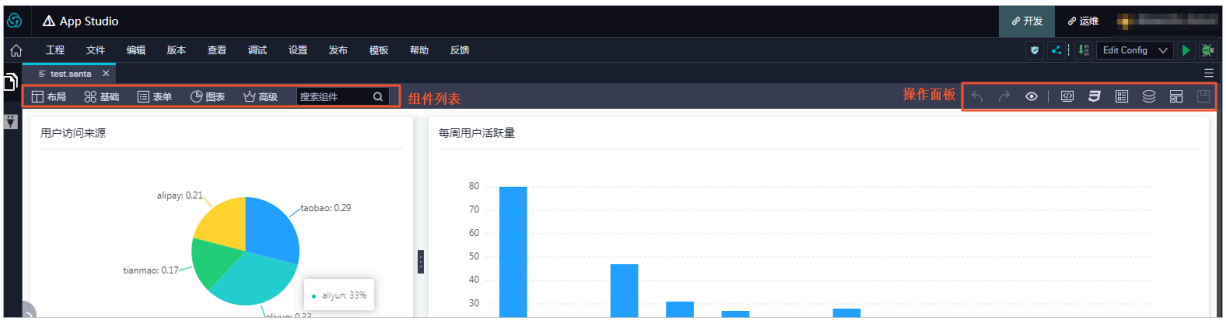
5. 单击任意一个.santa文件进入可视化搭建。

   您也可以右键单击pages，选择新建 > 模板文件，基于模板进行开发。



可视化搭建

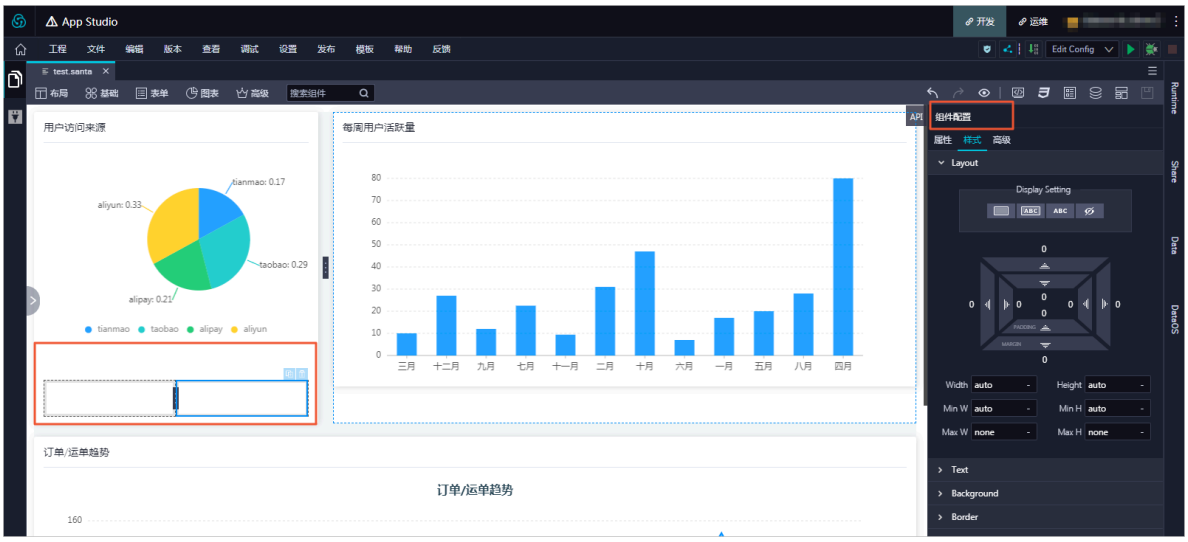
可视化搭建页面主要由组件列表和操作面板组成。



- 组件列表

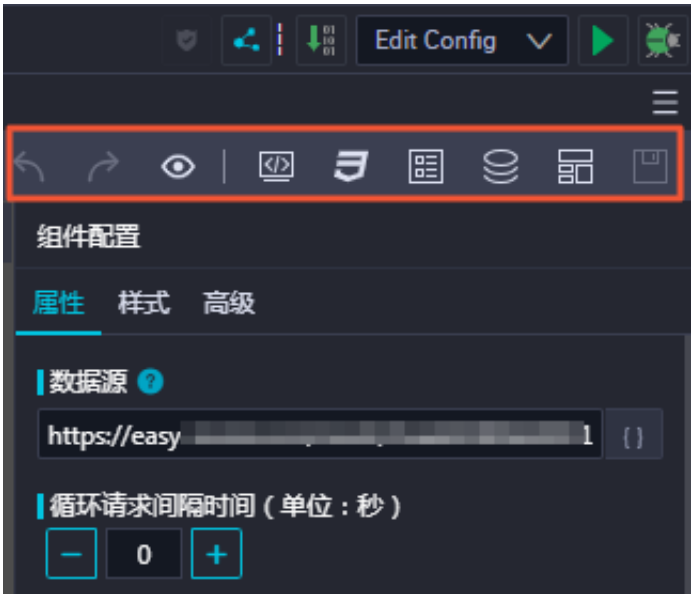
  组件列表为您展示可视化搭建系统中，所有的系统预设组件，包括布局、基础、表单、图表和高级等组件。

展开组件列表，拖拽某一个组件至可视化操作面板，单击该组件，即可在右侧进行组件配置。



- 操作面板

  操作面板包括撤销、重做、预览、代码模式、全局样式、导航配置、全局数据流配置、发布为模板和保存等操作。



单击操作面板中的导航配置图标，即可打开导航配置页面进行配置，详情请参见导航配置。

配置全局数据流

配置全局数据流的详情请参见全局数据流。

- 配置组件属性

  组件属性配置面板主要负责可视化的方式配置组件属性。

  根据组件的属性配置规则，组件属性配置面板将会生成一个可视化表单，让您输入组件的属性配置。在组件属性配置表单中更改组件属性后，可视化操作区域将会根据接收到的组件属性，进行重新渲染。您可以实时查看组件不同属性的渲染结果。

- 配置组件样式

  组件样式面板主要负责组件样式的相关设置。

  组件样式配置面板将会生成一个通用的样式配置可视化页面，您可以基于该面板定制组件基本的外观样式，包括布局、文字、背景、边框、效果等常用样式配置。

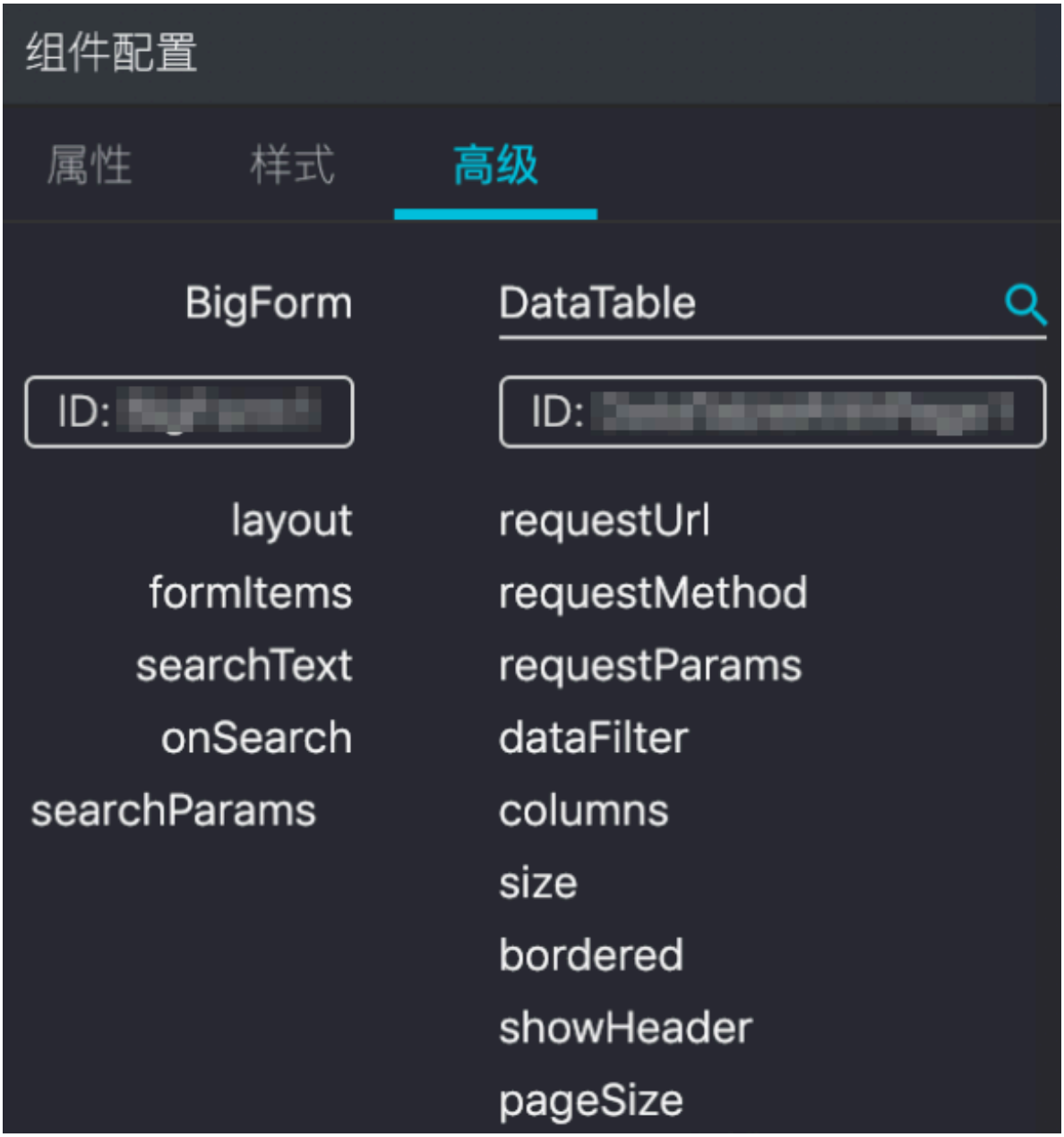
  在组件样式配置面板中添加、修改组件样式，可视化搭建系统将会收集所有的样式设置到组件上，可视化操作区域将会根据新的样式设置重新渲染对应组件，您可以实时查看配置后的组件效果。

- **配置组件联动高级**

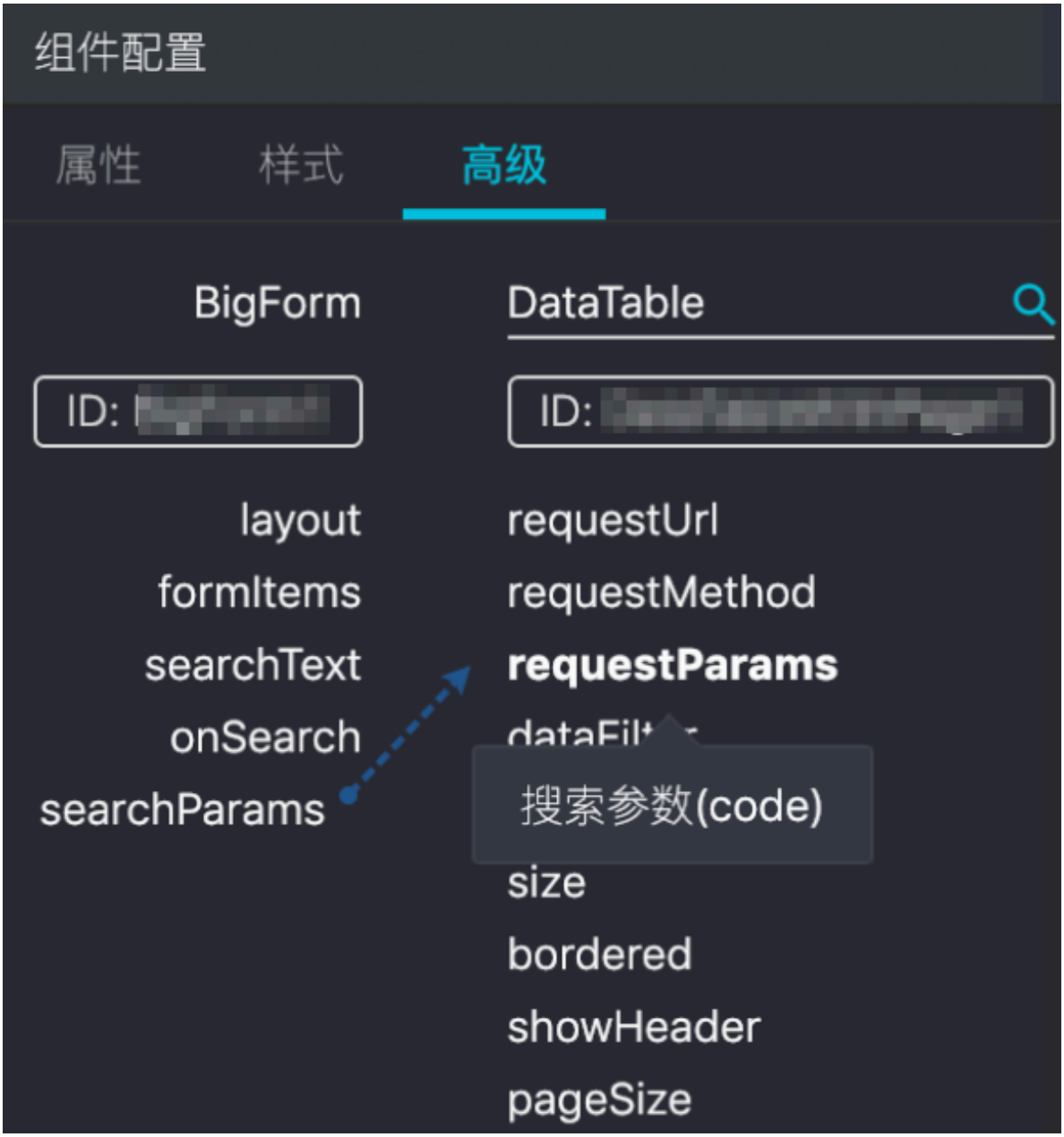
  组件联动高级设置面板主要负责组件之间的联动设置。

  单击可视化操作区域中的某一个组件，选中高级面板。高级设置面板中，将会在左侧列出当前选中组件对应的组件属性，单击右侧的放大镜按钮选择需要关联的另一个组件。

  选中需要关联的另一个组件后，高级设置面板右侧将会出现对应的组件属性。


组件配置
属性
样式
高级
BigForm
DataTable
ID:
ID:
layout
formItems
searchText
onSearch
searchParams
requestUrl
requestMethod
requestParams
dataFilter
columns
size
bordered
showHeader
pageSize

- **单击左侧属性列表中的某一个属性，连线至右侧属性列表中的另一个属性。**



该操作将会实现两个组件之间的属性联动，左侧组件的`searchParams`参数变更将会及时传递到右侧组件的`requestParams`参数，从而实现两个组件基于属性之间的联动配置。

代码模式

代码模式提供了一种更高级的方式来满足更复杂的交互场景的需求，详情请参见代码模式。

保存、预览、运行和热部署

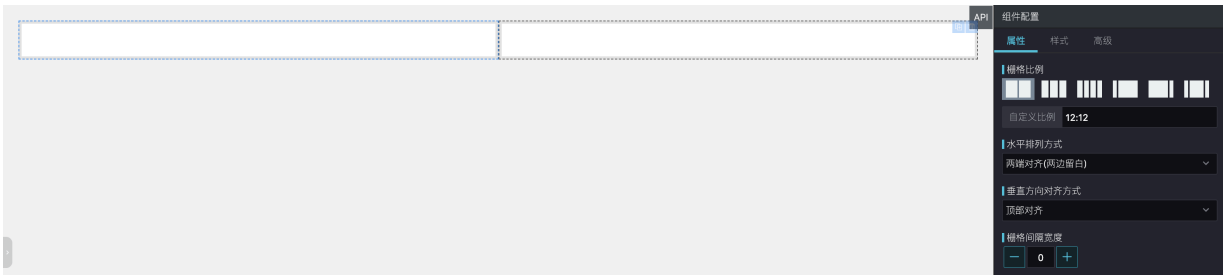
详情请参见保存、预览、运行和热部署。

1.4.9.3 常用组件

APP Studio可视化搭建系统自带80多个组件，可以满足您搭建基本页面的需求。本文将为您介绍可视化搭建系统默认自带的组件。

布局组件

布局组件为您提供一个24栅格系统组件。



- **栅格比例**

  系统默认将24栅格切割成一个12:12的栅格系统，您可以切换至其他常见的栅格比例，也可以自定义栅格比例。只需要保证所有栅格比例加起来是24的总数，布局组件将会根据各个栅格的比例进行布局切割。

- **水平排列方式**

  水平排列方式定义了栅格在父节点中的排版方式。

- **垂直排列方式**

  垂直排列方式定义了子元素垂直方向的对齐方式。

- **栅格间隔宽度**

  栅格常常需要和间隔进行配合，您可以使用该配置来定义栅格间隔。

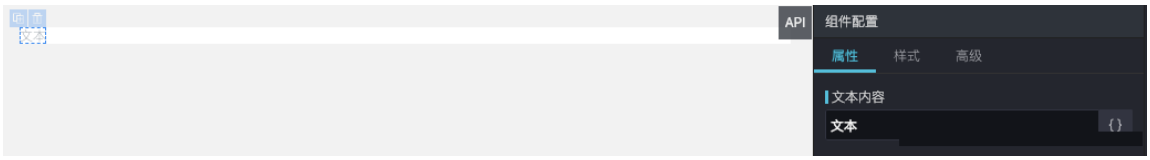
- **区块容器**

  区块容器是一个块状的容器组件，区块容器组件可以作为一系列组件的父组件，类似于HTML中的div容器。

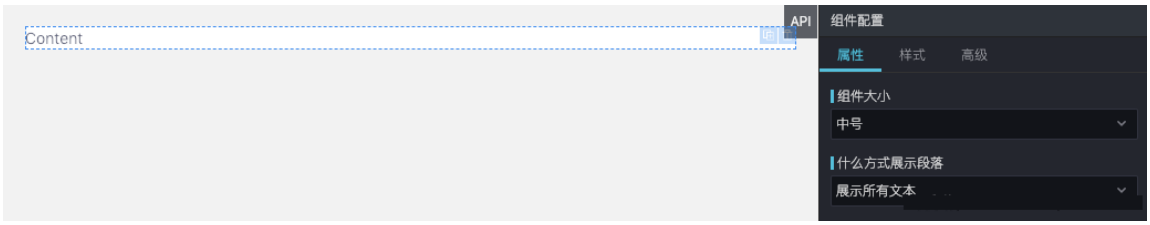
基础组件

基础组件均支持组件相关的常用属性设置。

- 文字
  - 文字



  - 段落



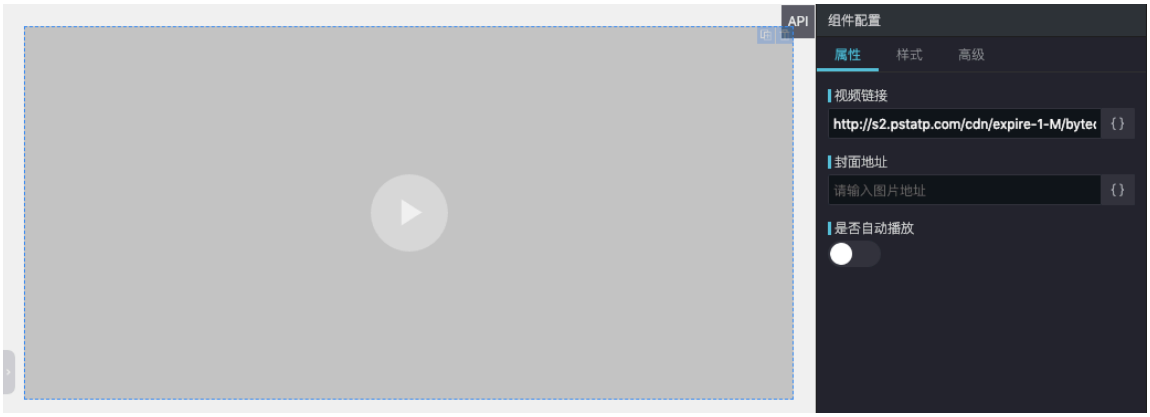
■ 组件大小

定义了段落文字大小。

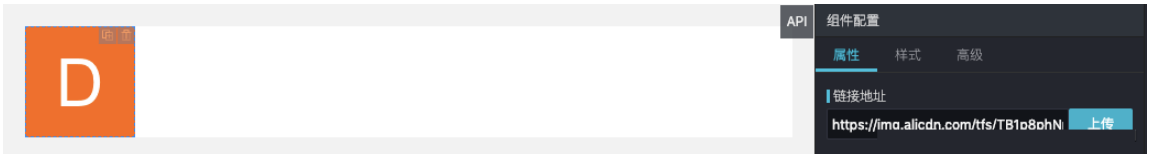
■ 什么方式展示段落

用于区分短文本和长文本，短文本的行间距会更小（通常三行以内）。

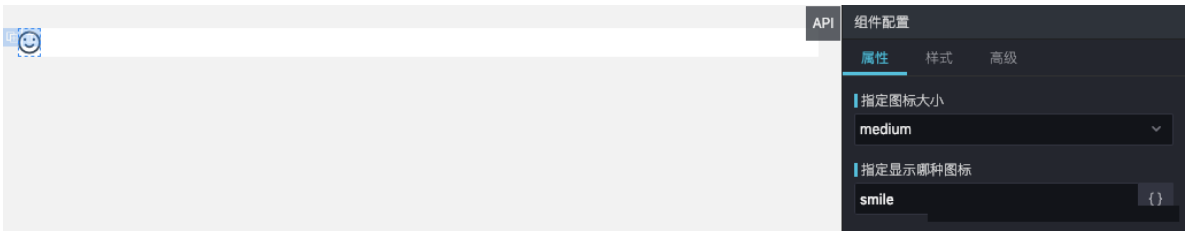
- 媒体
  - 视频



    - 视频链接：需要播放的视频地址。
    - 封面地址：视频封面图片地址。
    - 是否自动播放：是否在组件加载完之后自动播放视频。
  - 图片



    链接地址：显示的图片地址，可以上传图片。
- 图标

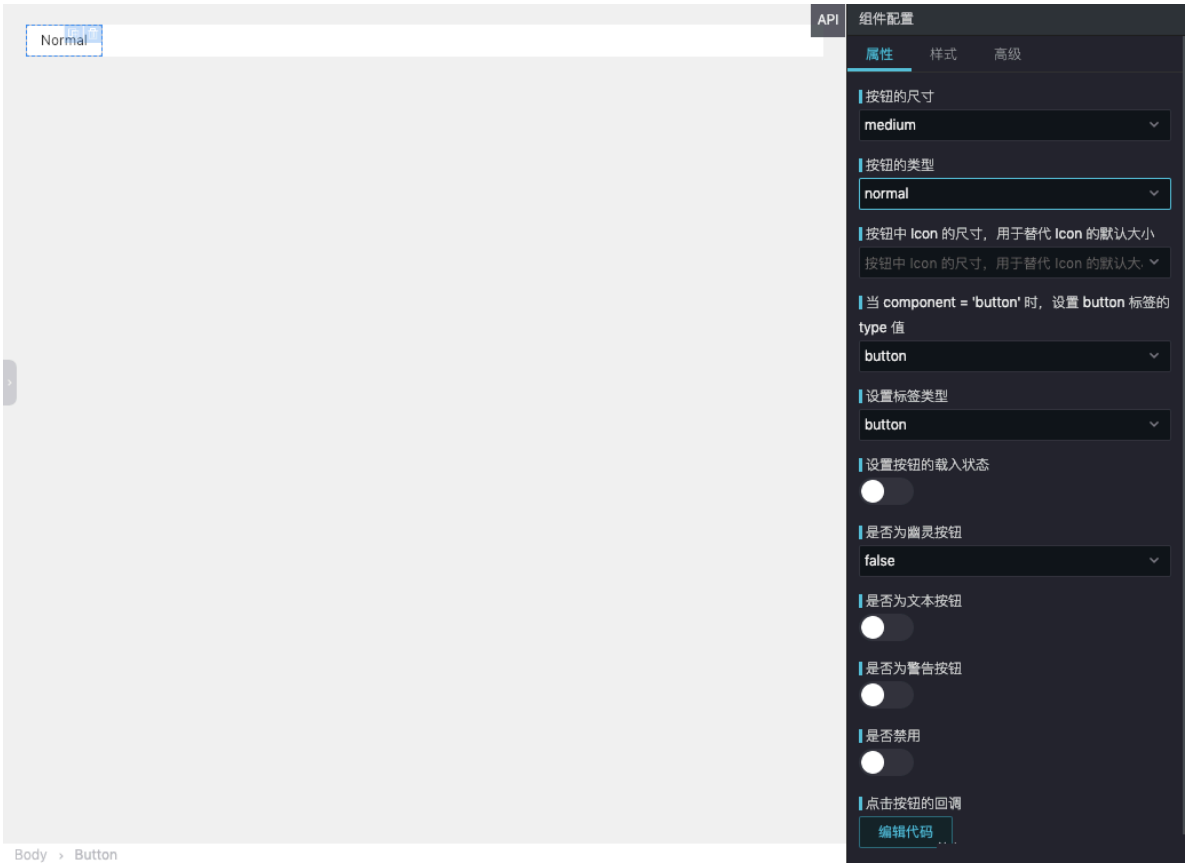


  - 指定图标大小

    指定图标的显示大小。
  - 指定显示哪种图标

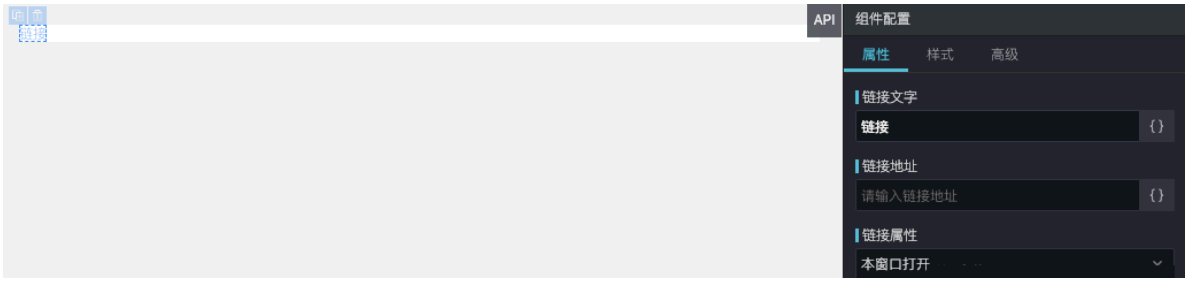
    指定图标的类型。

- 按钮



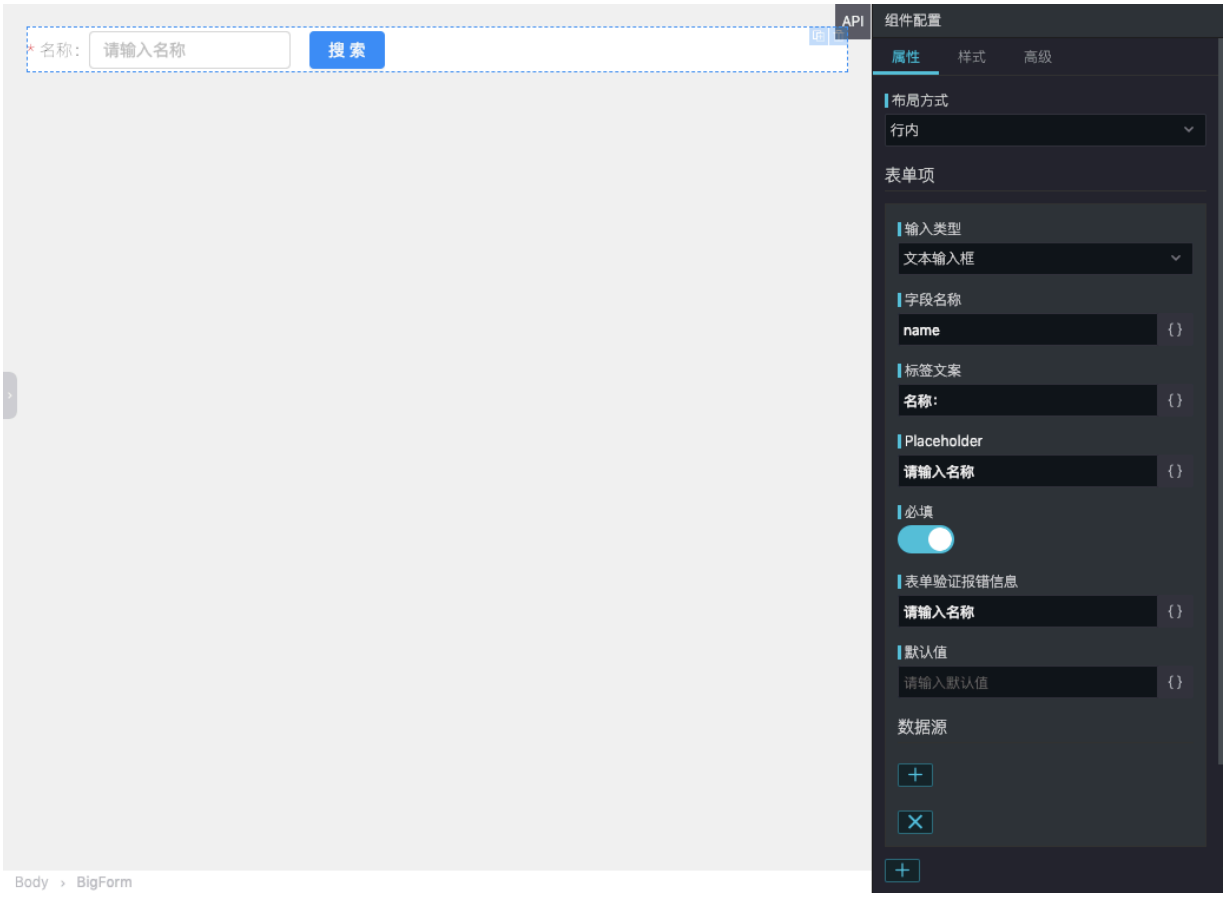
按钮属性的详情请参见按钮文档。

- 链接



  - 链接文字：显示的链接文字。
  - 链接地址：单击链接的跳转地址。
  - 链接属性：在本窗口打开和在新窗口打开。

表单组件



表单包括行内、水平和垂直三种布局方式。

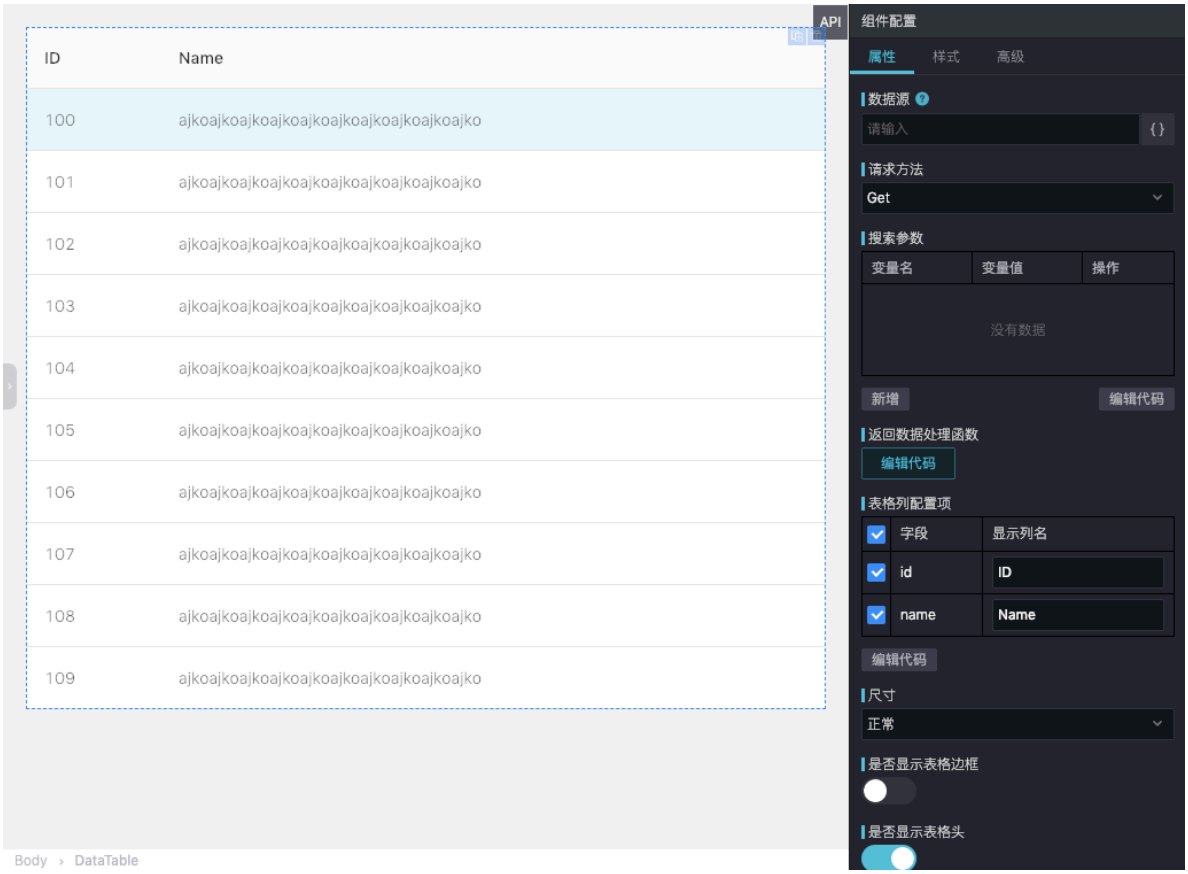
上传图片和附件详情请参见上传附件。

筛选详情请参见搜索。

输入框详情请参见输入框。

图表

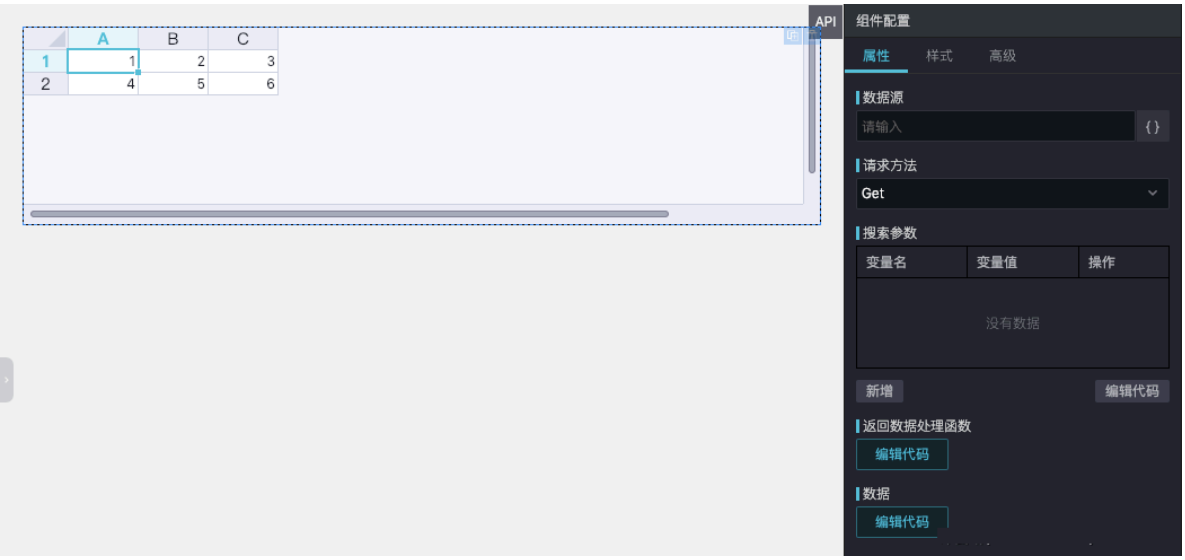
- **数据表格**



配置	说明
数据源	请求接口地址。
请求方法	请求方法：Get/Post/Put/Delete。
搜索参数	接口请求参数。
返回数据处理函数	接口数据返回后的数据处理函数。
表格列配置项	定义表格需要显示的表格列。
尺寸	设置表格尺寸。
是否显示表格边框	设置是否显示表格边框。
是否显示表格头	设置是否显示表格头。

带分页的数据表格多了一项每页显示数量的配置项，定义分页中每一页的显示数量。

- **Excel**



配置	说明
数据源	请求接口地址。
请求方法	请求方法：Get/Post/Put/Delete。
搜索参数	接口请求参数。
返回数据处理函数	接口数据返回后的数据处理函数。
数据	直接配置Excel需要显示的数据。

- 折线图

配置	说明
数据源	请求接口地址。
请求方法	请求方法：Get/Post/Put/Delete。
搜索参数	接口请求参数。
返回数据处理函数	接口数据返回后的数据处理函数。
图表配置	通过代码对图表进行配置。
是否显示图表标题	设置是否显示图表标题。
图表标题	显示图表标题。
图表数据	直接配置图表需要显示的数据。
X轴字段	定义返回数据中显示到X轴的数据字段名。
Y轴字段	定义返回数据中显示到Y轴的数据字段名。

说明:

柱状图、条形图、面积图、饼图、地图、词云和散点图等图表组件的配置，请参见折线图。

高级组件

高级组件均支持组件相关的常用属性设置。

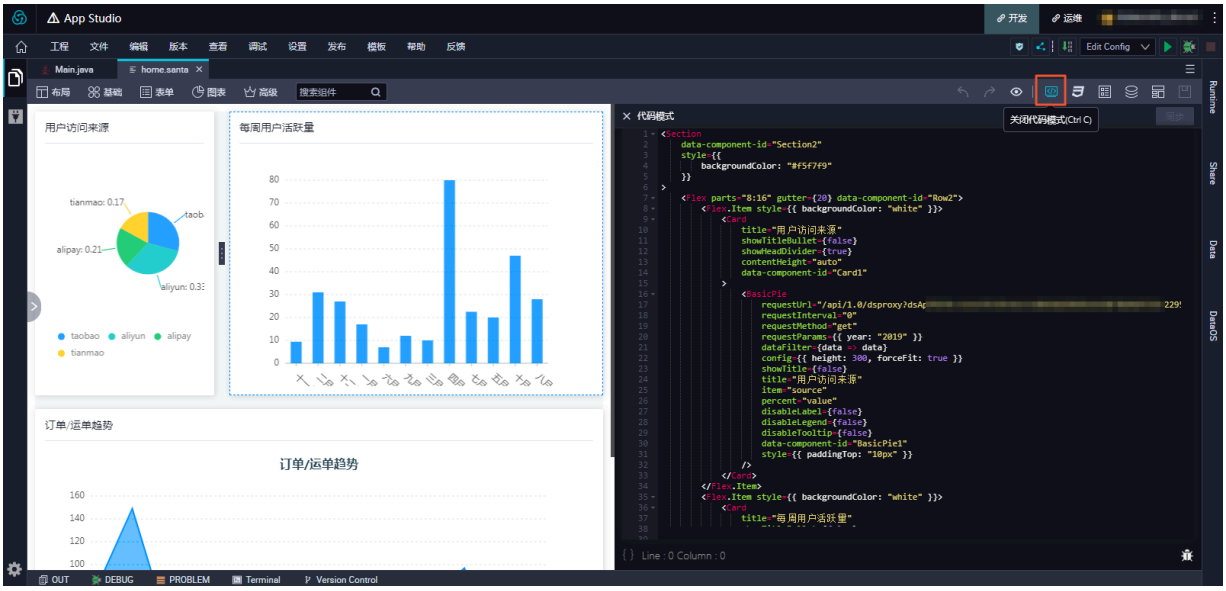
- 选择组件包括选择器、复选按钮、 级联选择、单选框、区段选择器、开关组件和评分。
- 交互：您可以通过Tab选项卡，在不同子任务、视图、模式之间切换，它具有全局导航的作用，是全局功能的主要展示和切换区域。详情请参见*Tab*选项卡。
- 轮播图：轮播组件以幻灯片的方式，在页面中横向展示诸多内容的组件。详情请参见图片轮播。
- 步骤条：默认情况下，Step定义为展示型组件。上层组件可以通过修改传入的current属性值来修改当前的步骤，同时可以设置每个节点的click事件，来自定义回调。详情请参见步骤。
- 进度条：进度指示器可以为您展示操作的当前进度。详情请参见进度指示器。
- 菜单：您可根据自身需求选择相应的菜单，详情请参见菜单。
- 导航：导航包括顶部导航和侧边导航。顶部导航提供全局性的类目和功能，侧边导航提供多级结构来收纳和排列网站架构。详情请参见导航。

1.4.9.4 代码模式

代码模式提供了一种更高级的方式来满足更复杂的交互场景的需求。

打开新建工程中`santa/pages`目录下的.santa文件，进入可视化搭建。

单击操作面板中的代码模式图标，即可在页面右侧出现代码区域。



可视化搭建使用DSL描述语言作为中间层的代码，基于该DSL进行可视化与代码模式的互转。可以简单地将DSL看作简化版的React，语法与React基本一致。

DSL将一个组件使用标签进行描述，标签的属性即组件的Props属性。属性值支持简单的数据类型，例如STRING或NUMBER。属性值也支持表达式，您可以直接输入`state.xxx`来获取全局数据流中的数据。

代码模式具有以下特点：

- 在可视化视图中进行的拖拽、组件属性配置等操作，会实时更新至代码。
- 代码中的修改会实时更新至可视化区域。
- 在可视化视图中进行的拖拽、组件属性配置等操作，与代码模式的修改可以互相转换。

1.4.9.5 DSL语法

DSL是一种以React JSX与Vue template的语言特性为基础，更符合UI编排的组件化语言。

JSX

DSL语法类似于React.render方法中的JSX部分，JSX的简单理解如下：

- 通过`{}`，将HTML作用域切换为JS作用域。JS作用域可以写任何合法的JS表达式，返回值会输出到页面上，例如`<div>{'Hello' + ' Relim'}</div>`。

  说明：

  `{ }`内可以写任何计算语句或字面量等JS表达式。

- 通过HTML标签，将JS作用域切换为HTML作用域，例如`{<div>Hello Relim</div>}`。
- HTML和JS作用域切换可以嵌套进行，例如`{<div>{'Hello' + ' Relim'}</div>}`。

JSX的更多详情请参见*React JSX*。

合法的JS表达式

```
//计算语句的情形
{aaa} // √   变量aaa需要有定义
{aaa * 111} // √
{1 == 1 ? 1 : 0} // √
{/^123/.test(aa)} // √
{[1,2,3].join('')} // √
{(()=>{return 1})()} //自执行函数 √

//字面量
{1}
{true}
{[11,22,33]} // √
{{aa:"11",bb:"22"}} // √
{()=>1} //描述一个函数，合法，但无意义 √
```

说明：

如果遇到较为复杂的逻辑，一条计算语句不能实现，需拆分为多条语句的需求。可以将其包装为自执行函数，自执行函数是合法的表达式。示例如下：

```
{(function(){
    //将一个数字数组的偶数为求和。
    var input = [1,2,3,4,5,6,7,8,9,10];
    var temp = input.filter(i => i % 2 == 0)
```

```
    return temp.reduce((buf, cur) => buf + cur, 0)
})()}
```

非法的JS表达式

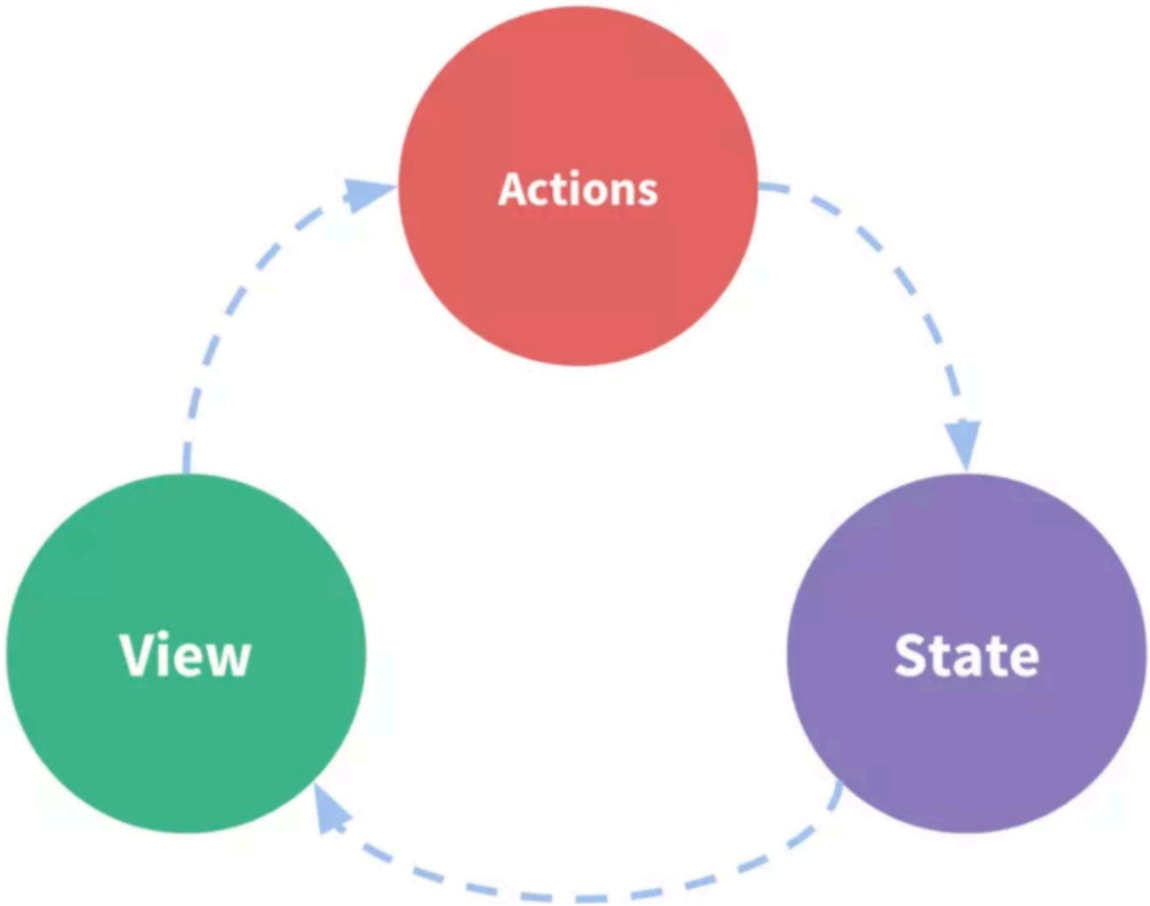
```
{ var a = 1 } // 赋值语句。
{ aaa * 111; 2} // 出现分号的多条语句。
```

1.4.9.6 全局数据流

全局数据流是前端数据管理的概念，多个组件为共享状态时，共享状态和组件间通信较为困难。此时将共享状态抽取出来，用全局数据流的方式使之变得简单。

全局数据流的原理

全局数据流使用了单一的数据流转方式，来实现全局数据的传递。在全局数据中声明的数据，只要变更后便会执行如下图所示的数据流转。



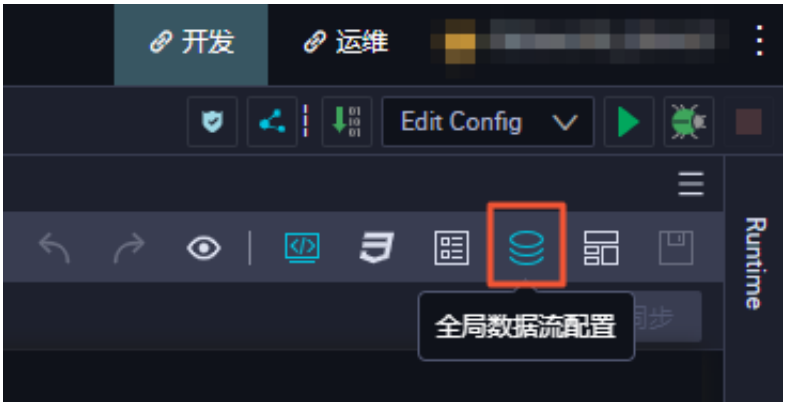
1. **组件触发一个Action（例如通过鼠标单击触发）。**
2. **Action触发全局数据变更。**
3. **全局数据变更会自动触发引用了该全局状态的相关组件的重新渲染。**

全局数据流的适用场景

全局数据流适用于页面中两个组件或者多个组件之间的组件联动，可以通过将公共数据提炼到全局数据中进行统一管理，再利用全局数据流机制串联两个或多个组件。

全局数据流的定义

1. **单击操作面板中的全局数据流配置图标。**



2. **在全局数据流配置对话框中，填写变量名和变量值。**

   - **变量值可以为数字、字符串或JSON串。**
   - **变量值声明为一个接口地址，接口获取到的数据将会成为变量名对应的值。**

3. **单击保存。**

使用全局数据流

- **获取全局数据**

  组件中通过`state.name`来获取全局数据。

  ```
  <Input value={state.name} />
  ```

- **修改全局数据**

  组件中通过$setState方法修改全局数据。

  ```
  <Input onChange={value => $setState({ name: value })} />
  ```

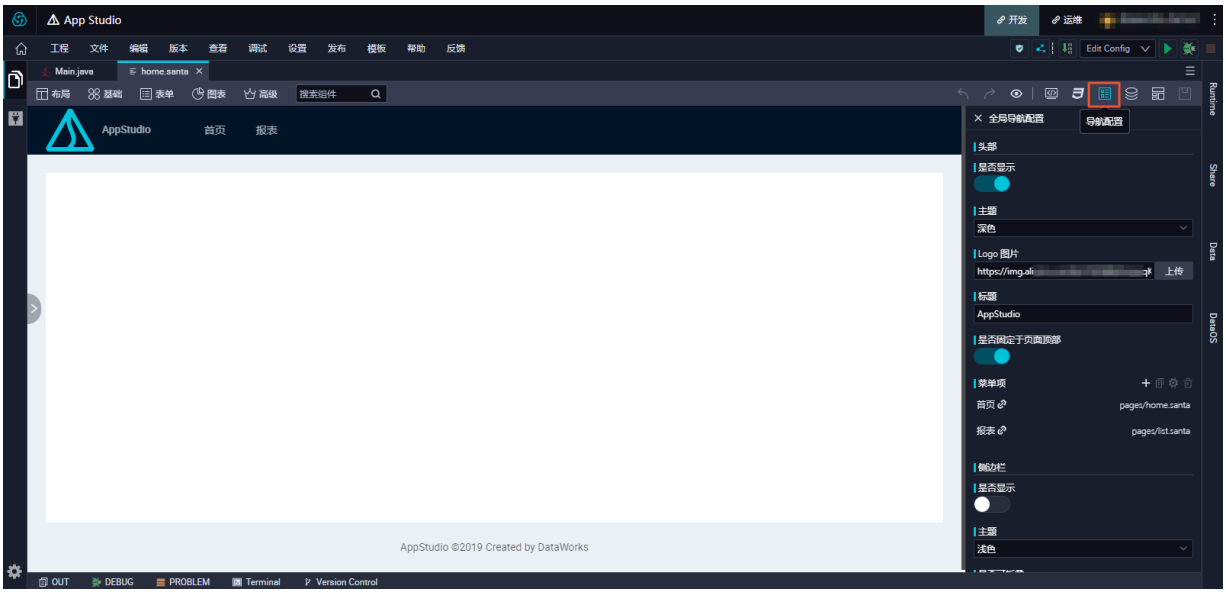
  说明:
  请务必使用$setState方法修改全局数据，如果使用`state.name = 'new value'`，将会无法触发重新渲染。

1.4.9.7 导航配置

本文将为您介绍如何设置可视化搭建站点的导航。

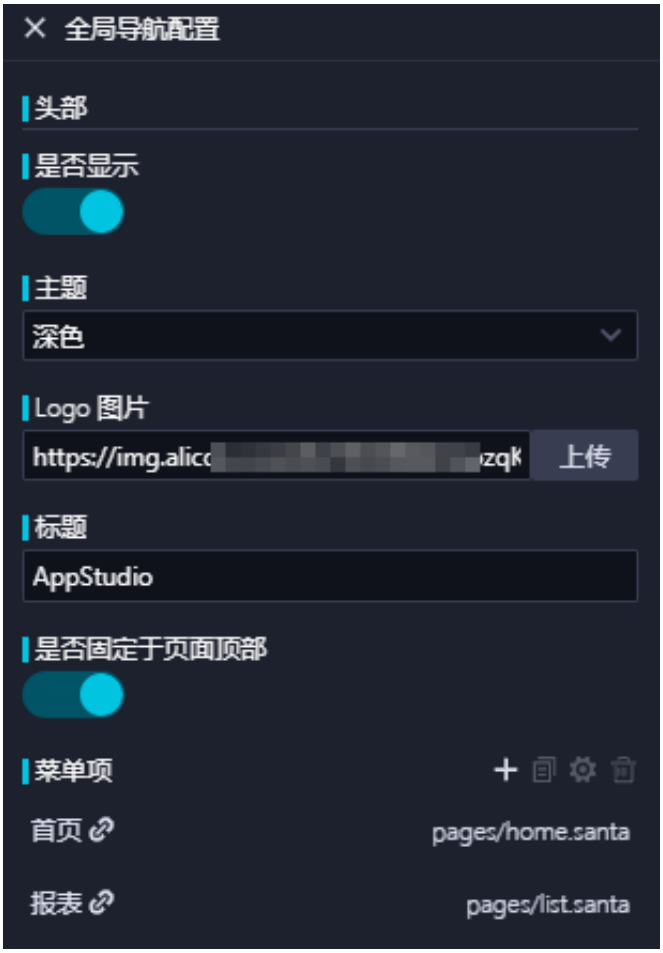
App Studio可视化搭建为应用提供公共头部、底部和侧边栏，提供了丰富的菜单配置、主题配置。如果您不需要显示系统提供的公共头部和侧边栏，可以进行配置。

单击操作面板中的导航配置按钮，即可打开导航配置页面。



配置公共头部

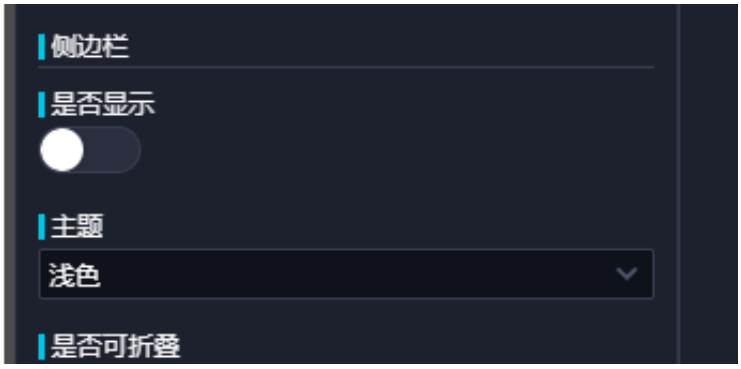
您可以根据自身需求对公共头部进行配置。



配置	说明
是否显示	设置是否显示公共头部。
主题	您可以选择深色或浅色的主题样式。
Logo图片	显示的站点Logo图片，您可以输入一个图片地址，或者选择本地上传一张图片。
标题	设置显示的站点标题。
是否固定于页面顶部	是否让公共头部一定固定于页面顶部（页面滚动时，公共头也将一直位于页面顶部）。
菜单项	您可以定义公共头部可以显示的链接名称、链接地址等菜单项。

配置侧边栏

您可以根据自身需求对侧边栏进行配置。



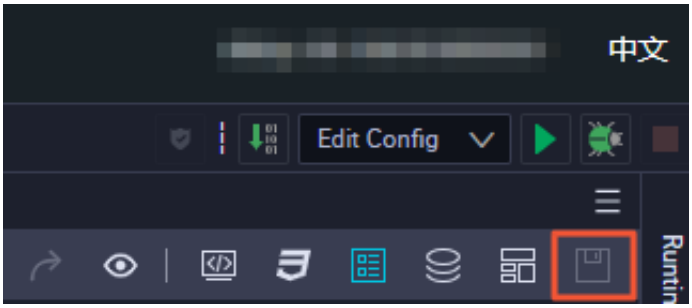
配置	说明
是否显示	设置是否显示侧边栏。
主题	您可以选择深色或浅色的主题样式。
是否可折叠	设置侧边栏菜单是否具有折叠功能。

1.4.9.8 保存、预览、运行和热部署

可视化搭建系统支持保存、预览、操作和热部署等操作。

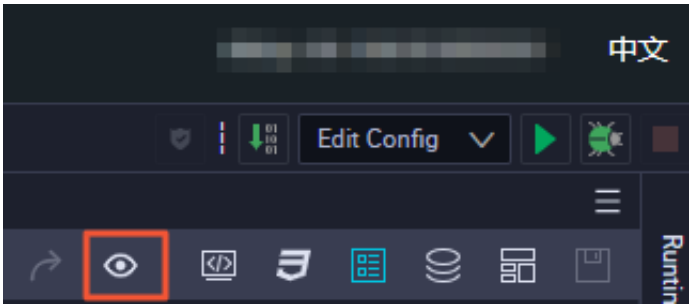
保存

可视化搭建系统会定时保存您的修改，您也可以单击操作面板中的保存图标，进行保存。



预览

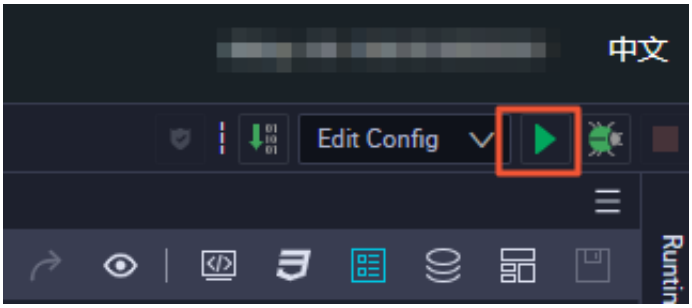
在可视化搭建系统中，可视化操作区域处于编辑的状态。有部分组件针对编辑状态进行特殊处理，只有在正式的运行状态下才能执行正常的渲染逻辑。如果您想查看正常的渲染结果，可以单击操作区域的预览图标。



运行

可视化搭建系统单次只能打开一个可视化文件进行编辑。如果您想要以整个应用的视角进行查看，可以运行整个应用来查看结果。

您可以单击App Studio Debug面板中的启动图标，来运行整个应用。



热部署

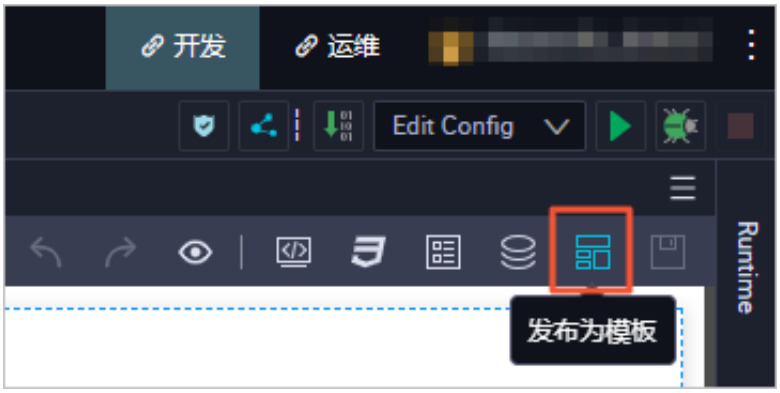
应用启动后，如果您发现页面不符合预期，可以继续返回到可视化搭建系统进行调整。

调整完成后进行保存，您的修改将会支持热部署的方式生效至运行的页面。

1.4.9.9 发布为模板

您可以将搭建好的前端页面发布为模板，后续基于该模板进行开发。

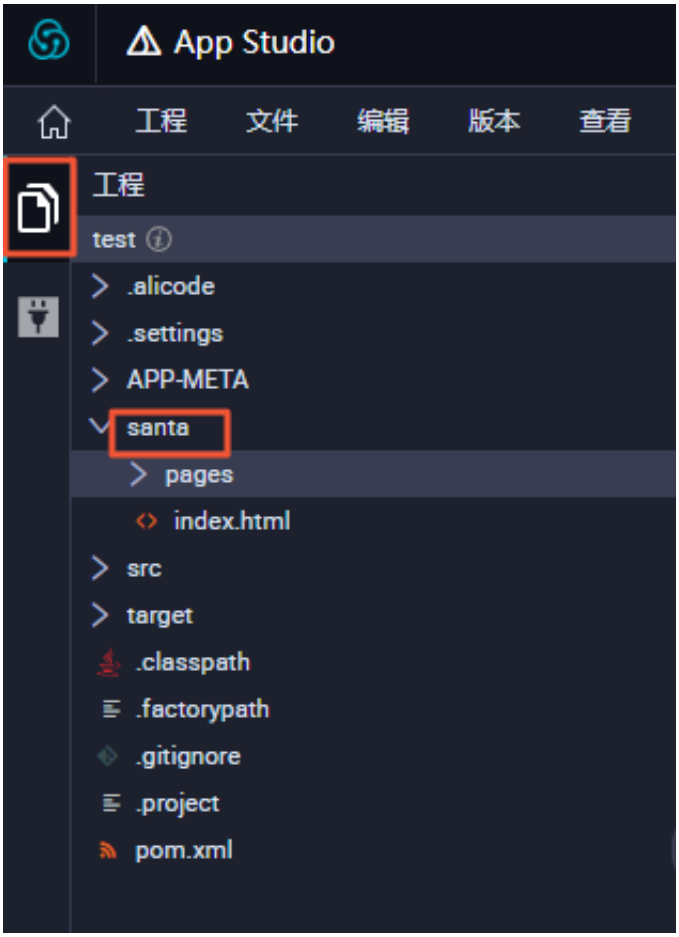
1. 打开可视化搭建文件，单击操作面板右上角的发布为模板图标。



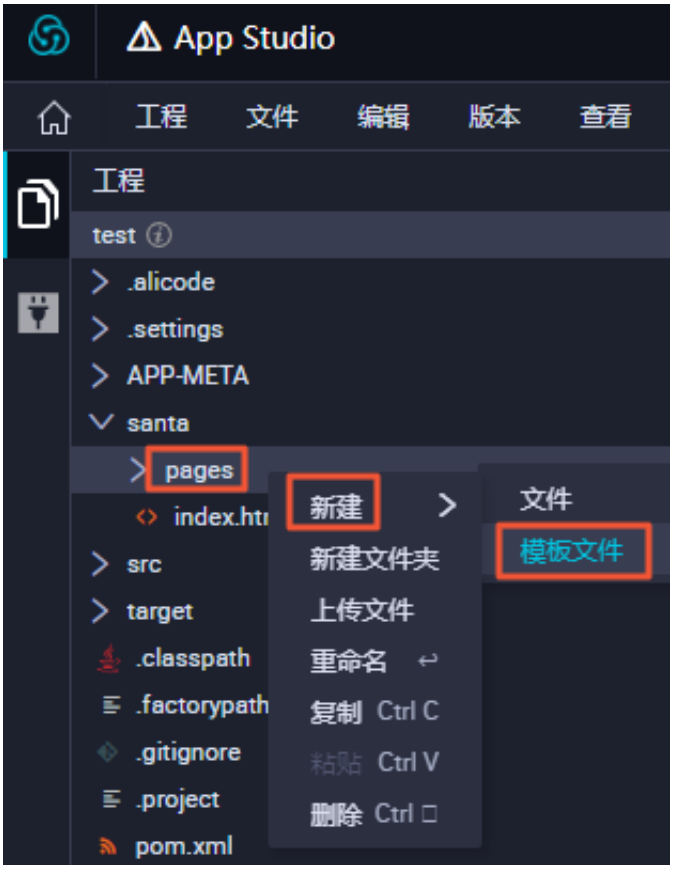
2. 在模板发布对话框中，单击截图图标。

3. 待右侧的等待录制框中显示截图内容后，填写模板名称和模板类型，单击确认。

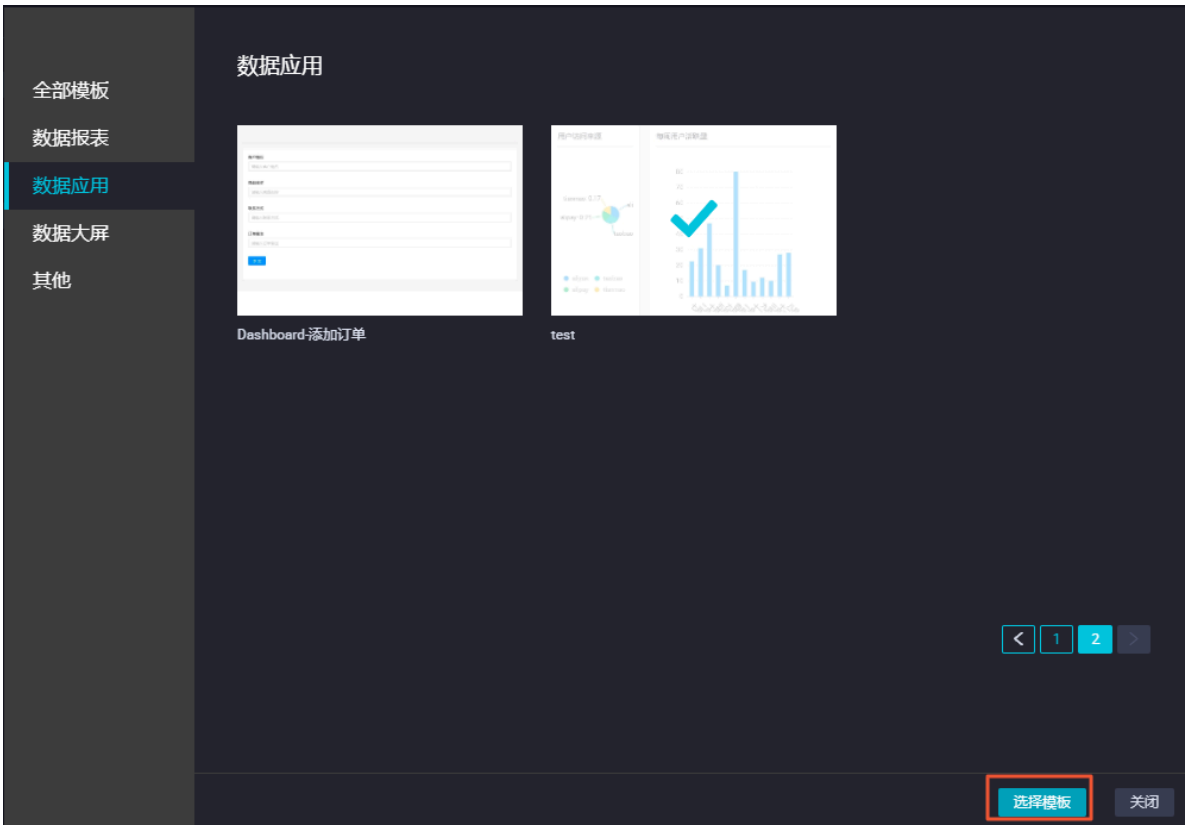
4. 单击左侧菜单栏中的工程文件，打开santa目录。



5. 右键单击pages，选择新建 > 模板文件。



6. 选择发布为模板的文件，单击选择模板。



7. 在添加文件对话框中填写文件名称，单击创建，即可新建一个页面，并基于模板进行开发。