ALIBABA CLOUD

阿里云

云服务总线 CSB

服务调用

文档版本：20201015

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.调用服务

本文介绍API消费方应用调用API的多种方式。

背景信息

API消费方应用调用API有多种方式，例如HTTP API、HSF API、WebService API等。其中HSF API的消费调用沿用HSF原有的服务调用方式，无需任何专用SDK，如有必要可以指定CSB的服务IP进行HSF服务调用；而调用HTTP API和WebService API则可以使用CSB Client SDK和JWT Token。

- SDK：目前提供了Java版本的HTTP Client SDK和WebService Client SDK。
- JWT Token：目前CSB开放平台支持创建JWT类型的凭证，可直接用于调用服务。

调用CSB开放出来的服务

如何调用由CSB开放出来的HTTP或者WebService服务，详情请参见CSB发布的服务的端口说明。

调用说明

使用SDK调用服务，详情请参见SDK参考。

使用JWT Token调用服务，详情请参见如何使用JWT Token调用服务。

2.调用特性

2.1. 同名多参数

CSB通过SDK和Broker实现对同名多参数的支持。

前提条件

使用的CSB SDK版本为http-client-1.1.5.8和ws-client-1.1.5.8。

背景信息

HTTP请求需要支持同名多参数，实际效果为数组，如果后端也是HTTP类型，须保持该形态传递到后端服务。

例如：CSB接收到形如 `key=value1&key=value2` 的请求时，会作为 `key = [ value1, value2 ]` 处理。如果后端服务也是HTTP类型时，传递方式遵循后端接口定义，缺省保持为 `key=value1&key=value2` 。

代码示例

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:18086/CSB") // 设置请求的URL。
  .api("http2http1") // 设置服务名。
  .version("1.0.0") // 设置版本号。
  .method("post") // 设置调用方式, get或post。
  .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
  // 设置请求参数。
  builder.putParamsMap("name", "name1中文sdfs sdlkfsadfksdkfds").putParamsMap("times", "3")
    .putParamsMap("str2", "31", "32", "33")  //设置str2的数组入参。
    .putParamsMap("str3", Arrays.asList("aa", "bb", "cc")); //设置str3的List入参。
  HttpReturn ret = HttpCaller.invokeReturn(builder.build());
  System.out.println("------- ret=" + JSON.toJSONString(ret));
} catch (HttpCallerException e) {
  e.printStackTrace(System.out);
}
```

HTTP消息示例如下图所示：

2.2. 附件传输

CSB实现了接入RESTful服务，开放成RESTful服务后的附件传输功能，支持上传或下载附件。

前提条件

- 已经接入RESTful服务，并开放成RESTful服务，且设置公开访问或订购审批通过。详情请参见发布后端已有服务。
- 使用的CSB SDK版本为http-client-1.1.5.8和ws-client-1.1.5.8。

使用约束

- 仅适用于接入RESTful服务，并开放成RESTful服务场景。对于接入HSF服务、Dubbo服务和WebService服务场景，CSB Broker只能获取附件，不支持转发给后端的HSF服务、Dubbo服务和WebService服务。
- 目前仅实现部分HTTP附件交互场景。支持场景如下：
  - 上传附件：
    - HTTP Body二进制流：使用POST，设置 `Content-Type: application/octet-stream` ，适合于仅上传单个二进制流。
    - 多附件二进制流：使用POST，设置 `Content-Type: multipart/form-data` ，适合于有普通form的业务数据，同时还有多个附件上传场景。
  - 下载附件：HTTP Body返回二进制流，设置 `Content-Type: application/octet-stream` 。
- 仅支持新版本服务，即在**发布设置**页面选择**新版（推荐）**。如果您未进行发布设置，则在第一次发布服务时选择**新版发布（推荐）**。
- 单个文件最大15 M，可以通过 `jvm -Dcsb_httpAttachmentTotalMBSize=M字节数` 来修改，Broker端通过csb_httpAttachmentTotalMBSize配置。
- 最多同时发送5个附件，可以通过 `jvm -Dcsb_httpAttachmentMaxAmount=文件数` 来修改。Broker端通过csb_httpAttachmentMaxAmount配置。
- Broker向后端业务服务发送多附件请求的响应超时时间为120秒（csb_httpAttachmentClientTimeoutSeconds）。
- 不支持自定义消息转换的响应逻辑。
- 每个Broker建立最多60个多附件传输的HTTP连接（csb_httpAttachmentClientMaxConn）。

上传附件

- HTTP Body二进制流

- 修改CSB客户端代码

  CSB客户端示例代码：

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:8086/CSB") // 设置CSB服务地址。
  .api("http2http1") // 设置服务名。
  .version("1.0.0") // 设置版本号。
  .method("post") // 设置调用方式, get或post。
  .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
  // 设置请求参数。
  builder.putParamsMap("name", "name中文1").putParamsMap("times", "3");
  builder.contentBody(new ContentBody(new File("文件名.xxx")));
  HttpReturn ret = HttpCaller.invokeReturn(builder.build());
} catch (HttpCallerException e) {
  // error process
}
```

  HTTP请求示例如下（上传xml文本文件）：

- 业务服务端处理方式

  根据Content-Type: application/octet-stream判断请求消息内容格式，并正确处理。

- 多附件二进制流
  - 修改CSB客户端代码

    CSB客户端示例代码：

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:8086/CSB") // 设置请求的URL。
  .api("http2http1") // 设置服务名。
  .version("1.0.0") // 设置版本号。
  .method("post") // 设置调用方式, get或post。
  .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
  // 设置form请求参数。
  builder.putParamsMap("times", "2").putParamsMap("name", "we中文wesdsfsfdsasdefds");

  //设置上传附件。
  builder.addAttachFile("file1", new File("文件名1.xxx"));
  builder.addAttachFile("file2", "fileName2", new FileInputStream(new File("文件名2.yyy")), true); //
对文件进行压缩传输。

  HttpReturn ret = HttpCaller.invokeReturn(builder.build());
} catch (Exception e) {
  // error process
}
```

HTTP请求示例如下（上传xml文本文件）：

```
```

> 说明
>
> - 通过builder.putParamsMap()方式设置的参数都是以application/x-www-form-urlencoded; charset=UTF-8方式提交请求。服务端需正确解码，以便得到正确的参数值。
> - 所有附件均以Content-Type: application/octet-stream上传。

- 业务服务端处理方式
  - 根据Content-Type: multipart/form-data判断是否是多附件上传请求，并正确解析请求内容。
  - 根据每个form表单的Content-Type: application/x-www-form-urlencoded或Content-Type: application/octet-stream正确解析对应form数据。

下载附件

CSB HTTP SDK会自动解析HTTP响应，并根据HTTP Header的Content-Type: application/octet-stream设置HttpReturn对象的response或responseBytes。

示例代码如下：

```
HttpReturn ret = HttpCaller.invokeReturn(builder.build());
ret.responseBytes; //响应结果的二进制数据。
ret.response; //响应结果的字符串数据。
```

FAQ

- 什么场景使用“Body二进制流”，什么场景使用“多附件二进制流”？

上传方式	HTTP Query参数	HTTP Form参数	上传附件数量
Body二进制流	支持	不支持	1个
多附件二进制流	支持	支持	1个或多个

- 最大支持附件大小是多少？

  Broker目前最多支持15 M附件大小，不管附件数量，一次请求合计不能超过15 M。

2.3. HTTP请求和响应的压缩传输

CSB按照HTTP协议的规范实现了HTTP请求和响应内容的压缩传输。

使用约束

- 仅支持新版本服务（serviceModel version: 2.0）。
- 仅支持gzip压缩。

前提条件

使用的CSB SDK版本为http-client-1.1.5.8和ws-client-1.1.5.8。

application/x-www-form-urlencoded请求压缩

- 修改CSB客户端代码

  示例代码：

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:8086/CSB") // 设置CSB服务地址。CSB服务地址即创建该实例时绑定的SLB的地址。
  .api("http2http1") // 设置服务名。
  .version("1.0.0") // 设置版本号。
  .method("post") // 设置调用方式, get或post。
  .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
  builder.setContentEncoding(ContentEncoding.gzip);//设置请求消息压缩。

  // 设置请求参数。
  builder.putParamsMap("name", "name中文1").putParamsMap("times", "3");
  HttpReturn ret = HttpCaller.invokeReturn(builder.build());
} catch (HttpCallerException e) {
  // error process
}
```

HTTP请求示例如下：

```
```

- 业务服务端处理方式

  根据 `Content-Encoding: gzip` 判断请求消息是否压缩，并正确解压，然后再使用URL解码参数。

Content-Type: application/json请求压缩

- 修改CSB客户端代码

  示例代码：

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:8086/CSB") // 设置 CSB 服务地址。CSB服务地址即创建该实例时绑定的SLB的地址。
 .api("http2http1") // 设置服务名。
 .version("1.0.0") // 设置版本号。
 .method("post") // 设置调用方式, get或post。
 .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
 builder.setContentEncoding(ContentEncoding.gzip);//设置请求消息压缩。

 // 设置请求参数。
 builder.putParamsMap("name", "name1中文sdfs sdlkfsadfksdkfds").putParamsMap("times", "3");
 Map<String, String> kvMap = new HashMap<String, String>();
 for (int i = 0; i < 100; ++i) {
  kvMap.put(String.valueOf(i), "abc中文lksd" + i);
 }
 builder.contentBody(new ContentBody(JSON.toJSONString(kvMap)));
} catch (HttpCallerException e) {
 // error process
}
```

HTTP请求示例如下：

- 业务服务端处理方式

  根据 `Content-Encoding: gzip` 判断请求消息是否压缩，并正确解压。

application/octet-stream请求压缩

- 修改CSB客户端代码

  示例代码：

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:8086/CSB") // 设置 CSB 服务地址。CSB服务地址即创建该实例时绑定的SLB的地址。
  .api("http2http1") // 设置服务名。
  .version("1.0.0") // 设置版本号。
  .method("post") // 设置调用方式, get或post。
  .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
  builder.setContentEncoding(ContentEncoding.gzip);//设置请求消息压缩。

  // 设置请求参数。
  builder.putParamsMap("name", "name中文1").putParamsMap("times", "3");
  builder.contentBody(new ContentBody(new File("文件名.xxx")));
  HttpReturn ret = HttpCaller.invokeReturn(builder.build());
} catch (HttpCallerException e) {
  // error process
}
```

HTTP请求示例如下：

- 业务服务端处理方式

  根据 `Content-Encoding: gzip` 判断请求消息是否压缩，并正确解压。

multipart/form-data请求压缩

- 修改CSB客户端代码

  示例代码：

```
HttpParameters.Builder builder = new HttpParameters.Builder();
builder.requestURL("http://localhost:8086/CSB") // 设置请求的URL。
  .api("http2http1") // 设置服务名。
  .version("1.0.0") // 设置版本号。
  .method("post") // 设置调用方式, get或post。
  .accessKey("ak").secretKey("sk"); // 设置AccessKey ID和AccessKey Secret。

try {
  builder.setContentEncoding(ContentEncoding.gzip);//设置请求消息压缩。

  // 设置form请求参数。
  builder.putParamsMap("times", "2").putParamsMap("name", "we中文wesdsfsfdsasdefds");

  //设置上传附件。
  builder.addAttachFile("file1", new File("文件名1.xxx")); //默认按请求消息的压缩标记来判断是否压缩。
  builder.addAttachFile("file2", "fileName2", new FileInputStream(new File("文件名2.yyy")),ContentEnc
oding.none);  //明确不压缩

  HttpReturn ret = HttpCaller.invokeReturn(builder.build());
} catch (Exception e) {
  // error process
}
```

HTTP请求示例如下：

- 业务服务端处理方式

  根据 `Content-Encoding: gzip` 判断请求消息是否压缩，并正确解压。

响应压缩

HTTP SDK会自动解析HTTP响应，并根据HTTP Header的 `Content-Encoding: gzip` 自动解压响应消息。

2.4. 通过bizId和traceId跟踪交易的调用请求

您可以通过设置、获取bizId的API，将请求链路各环节通过bizId串联起来，再结合traceId，对调用请求进行跟踪、分析。

前提条件

使用的CSB SDK版本为http-client-1.1.5.8和ws-client-1.1.5.8。

在SDK中设置bizId API和traceId

CSB目前支持两种协议的SDK，HTTP SDK和WebService SDK。同时，使用这两种SDK又包含两种调用方式，命令行调用和代码调用。所以下文分别介绍在HTTP SDK和WebService SDK中如何使用命令行和代码调用bizId API。

- HTTP SDK
  - 命令行调用

    将bizIdKey和bizId以命令参数的形式添加到命令中。格式如下：

    ```
    -bizIdKey <bizid的key> -bizId <bizId的value>
    ```

    其中，bizIdKey的默认值为_biz_id，不可配置。

    - 调用请求示例

      ```
      java -jar http-client-1.1.5.8.jar -api item.hsf.add -version 1.0.0 -method post \
      -bizIdKey _biz_id -bizId e48ffd7c1e7f4d07b7fc141f4350**** \
      -D "item={\"itemName\":\"benz\",\"quantity\":10}" \
      -url http://csb.broker.server:8086/CSB
      ```

    - 调用响应示例

      ```
      {
       "body": {
        "msg": "SUCCESS",
        "result": {
         "itemName": "benz",
         "trace": {
          "traceId": "1e195a1e15586942836161002d****",
          "rpcId": "0.1",
          "requestId": "1e195a1e15586942832381001d6898",
          "bizId": "e48ffd7c1e7f4d07b7fc141f4350****"
         },
         "quantity": 220
        },
        "code": "0"
       },
       "code": 200,
       "message": "SUCCESS",
       "requestId": "1e195a1e15586942836161002d6898"
      }
      ```

- 代码调用

  在代码中设置bizId，并启用Trace。bizIdKey的默认值为_bizId。

  a. 设置bizId

  ```
  HttpParameters.Builder builder = HttpParameters.newBuilder()
          .bizId(BIZ_ID);
  ```

  b. 启用Trace

  生成traceId、rpcId和requestId。

  - traceId：单次完整请求traceId相同（中间可能调用多次CSB服务）。
  - requestId：每次调用CSB服务重新生成。

  针对服务所在环境不同，又分为以下两种情况：

  - Web（独立部署，不能使用EagleEye组件）：包含引入trace filter和调用API两种方式。
    - 在web.xml文件中引入trace filter

      通过filter将接收到的trace header封装到请求上下文，最终封装到SDK请求header中。

      ```
      <filter>
      <filter-name>TraceFilter</filter-name>
      <filter-class>com.alibaba.csb.trace.TraceFilter</filter-class>
      </filter>
      <filter-mapping>
      <filter-name>TraceFilter</filter-name>
      <url-pattern>/*</url-pattern>
      </filter-mapping>
      ```

    - 调用API

      调用trace方法（将接收到的trace封装到SDK请求header）

      - builder.trace(httpServletRequest)
      - builder.setRequest(httpServletRequest).trace()
  - EDAS

    部署到EDAS的服务，可以使用EDAS的EagleEye，只需引入trace-eagleeye依赖。

    ```
    <dependency>
      <groupId>com.alibaba.csb.trace</groupId>
      <artifactId>trace-eagleeye</artifactId>
      <version>${http.sdk.version}</version>
    </dependency>
    ```

- WebService SDK
  - 命令行调用

    将bizIdKey和bizId以命令参数的形式添加到命令中。格式如下：

```
-bizIdKey <bizid的key> -bizId <bizId的value>
```

其中，bizIdKey的默认值为_biz_id，不可配置。

- 调用请求示例

```
java -jar ws-client-1.1.5.8.jar -api item.hsf.add -version 1.0.0 \
-bizIdKey _biz_id -bizId e48ffd7c1e7f4d07b7fc141f43503cb2 \
-wa http://csb.broker.server:9081/item.hsf.add/1.0.0/add?wsdl \
-ea http://csb.broker.server:9081/item.hsf.add/1.0.0/add \
-ns http://itemcenter.carshop.edas.alibaba.com/ \
-sname item.hsf.add \
-pname addPortType \
-rd '
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:test="http://itemcenter.carshop.edas.alibaba.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <test:add>
      <item>
        <itemName>benz</itemName>
        <quantity>5</quantity>
      </item>
    </test:add>
  </soapenv:Body>
</soapenv:Envelope>
';
```

- 调用响应示例

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <soap:Body
    xmlns:ns1="http://itemcenter.carshop.edas.alibaba.com/">
    <ns1:addResponse>
      <return>
        <code>0</code>
        <msg>SUCCESS</msg>
        <innerMsg/>
        <result>
          <itemName>benz</itemName>
          <quantity>250</quantity>
          <trace>
            <traceId>1e195a1e15586975657511003d****</traceId>
            <rpcId>0.1</rpcId>
            <bizId>e48ffd7c1e7f4d07b7fc141f435****</bizId>
            <requestId>1e195a1e15586975655741001d7506</requestId>
          </trace>
        </result>
      </return>
    </ns1:addResponse>
  </soap:Body>
</soap:Envelope>
```

- 代码调用

  在代码中设置bizId，并启用Trace。

  > 说明 bizIdKey的默认值为_bizId，不可配置。

  a. 设置bizId

```
HttpParameters.Builder builder = HttpParameters.newBuilder()
        .bizId(BIZ_ID);
```

b. 启用Trace

生成traceId/rpcId/requestId。

- traceId：单次完整请求traceId相同（中间可能调用多次CSB服务）。
- requestId：每次调用CSB服务重新生成。

针对服务所在环境不同，又分为以下两种情况：

- Web（独立部署，不能使用EagleEye组件）：包含引入trace filter和调用API两种方式。
  - 在web.xml文件中引入trace filter

    通过filter将接收到的trace header封装到请求上下文，最终封装到SDK请求header中。

```
<filter>
<filter-name>TraceFilter</filter-name>
<filter-class>com.alibaba.csb.trace.TraceFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>TraceFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

  - 调用API

    调用trace方法（将接收到的trace封装到SDK请求header）

    - `wsparam.trace(request)`
    - `wsparam.setRequest(request).trace()`

- EDAS

  部署到EDAS的服务，可以使用EDAS的EagleEye，只需引入trace-eagleeye依赖。

```
<dependency>
  <groupId>com.alibaba.csb.trace</groupId>
  <artifactId>trace-eagleeye</artifactId>
  <version>${http.sdk.version}</version>
</dependency>
```

获取请求链路信息

完成在SDK中设置bizId API和traceId 后，就可以通过TraceData、TraceId、HTTP Header或rpcId等方式获取链路信息。不同的服务类型，方式有所不同。

- HTTP/WS
  - TraceFilter(HTTP/WS)

    `TraceFactory.getTraceData()`

  - EDAS(WEB)
    - EagleEye.getTraceId()
    - EagleEye.getRpcId()
    - EagleEye.getUserData($bizIdKey)
    - EagleEye.getUserData(TraceData.REQUESTID_KEY) //_inner_ecsb_request_id
- HTTP/WS
  - request.getHeader(TraceData.TRACEID_KEY) //_inner_ecsb_trace_id
  - request.getHeader(TraceData.RPCID_KEY) //_inner_ecsb_rpc_id
  - request.getHeader(HttpCaller.bizIdKey()) //设置的bizIdKey。
  - request.getHeader(TraceData.REQUESTID_KEY) //_inner_ecsb_request_id
- HSF
  - EagleEye.getTraceId()
  - EagleEye.getRpcId()
  - EagleEye.getUserData(HttpCaller.bizIdKey()) //设置的bizIdKey。
  - EagleEye.getUserData(TraceData.REQUESTID_KEY) //_inner_ecsb_request_id
- Dubbo
  - RpcContext.getContext().getAttachment(TraceData.TRACEID_KEY) //_inner_ecsb_trace_id
  - RpcContext.getContext().getAttachment(TraceData.RPCID_KEY) //_inner_ecsb_rpc_id
  - RpcContext.getContext().getAttachment(HttpCaller.bizIdKey()) //设置的bizIdKey。
  - RpcContext.getContext().getAttachment(TraceData.REQUESTID_KEY) //_inner_ecsb_request_id

查看日志

- 引入log4j

  完成在SDK中设置bizId API和traceId后，您还可以引入log4j，从而基于bizId查看相关日志。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="WARN" monitorInterval="30">
 <appenders>
  <File name="csbsdk" fileName="logs/csbsdk.log">
   <PatternLayout pattern="%m%n"/>
  </File>
  <Async name="async">
   <AppenderRef ref="csbsdk"/>
  </Async>
 </appenders>

 <loggers>
  <logger name="CSBSDK" level="INFO" additivity="false">
   <appender-ref ref="async" />
  </logger>
 </loggers>
</configuration>
```

- 查看客户端日志

  客户端日志格式如下：

```
startTime|endTime|cost|HTTP/WS|localhost|dest|bizId|requestId|traceId|rpcId|api|version|ak|sk|method|ur|httpcode|httpreturn|msg
1559179173797|1559179173850|53|HTTP|192.168.*.*|csb.target.server|1e195a2815591791594031001d6512|1e195a2815591791737961004d****|1e195a2815591791737961005d****|0|item.hsf.remove|1.0.0|ak|sk|GET|http://csb.target.server:8086/CSB|200|HTTP/1.1 200 OK|
1558949495655|1558949497782|62|WS|192.168.*.*|csb.target.server|1e195a2715589494944221001d5b76|1e195a2715589494954281002d****|1e195a2715589494969271003d****|0|item.dubbo.add|1.0.0|ak|sk|GET|http||add|http://csb.target.server:9081/item.dubbo.add/1.0.0/add|200||
```

- 查看Broker日志

  Broker日志格式如下：

| 1.日志打印时间 | 11.签名 | 21.服务组名称 |31.返回数据大小
| 2.实例名称 | 12.调用后端服务开始时间 | 22.服务拥有者 |32.requestId
| 3.traceId | 13.是否成功 | 23.输入参数 |33.bizId
| 4.rpcId | 14.错误类型 | 24.输出参数
| 5.服务名称 | 15.错误代码 | 25.内部requestId
| 6.凭证名字 | 16.错误信息 | 26.来源IP
| 7.消费者 | 17.调用总耗时 | 27.目标IP
| 8.请求类型 | 18.后端服务调用耗时 | 28.请求写buffer时间
| 9.请求时间 | 19.后端服务调用结束时间 | 29.请求写成功时间
| 10.accessKey | 20.请求结束时间 | 30.请求数据大小

3.调用统计

3.1. 查看服务组调用量

在服务组调用量统计中可以看到当前用户创建的所有服务组的调用量统计信息，还可以查看单个服务组调用量统计的监控图表。

查看服务组调用量

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 服务组调用量统计**。
6. 在**服务组调用量统计**页面查看当前用户所创建的所有服务组的调用量统计信息。

   包括**服务组名称**、**调用量**、**错误量**、**最大RT(ms)**、**最小RT(ms)** 和**平均RT(ms)**。

   也可以输入**服务组名称**关键字，选择**时间范围**，单击**查询**，进行查看。

查看单个服务组调用量的监控图表

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 服务组调用量统计**。
6. 在**服务组调用量统计**页面具体服务组的**操作**列，单击**查看监控图表**。
7. 在弹出的对话框中选择要查看的时间周期，单击**查询**，查看当前选中的服务组在选定的时间周期内的调用量统计的监控图表。

   图表里面展示了该服务组在选定时间内的调用量和错误量的图表曲线。

3.2. 查看服务调用量

在服务调用量统计中可以看到当前用户所有服务的调用量统计信息，还可以查看单个服务调用量统计的监控图表。

查看服务调用量

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。

5. 在**实例概览**页面左侧导航栏选择**调用统计 > 服务调用量统计**。
6. 在**服务调用量统计**页面查看当前用户所创建的所有服务的调用量统计信息。

   包括服务**调用量**、**错误量**、**最大RT(ms)**、**最小RT(ms)**和**平均RT(ms)**。

   也可以输入**服务名称**关键字，选择**时间范围**，单击**查询**，进行查看。

查看单个服务调用量的监控图表

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 服务调用量统计**。
6. 在**服务调用量统计**页面具体服务的**操作**列，单击**查看监控图表**。
7. 在弹出的对话框中选择要查看的时间周期，单击**查询**，查看当前选中的服务在选定的时间周期内的调用量统计的监控图表。

   图表里面展示了该服务在选定时间内的调用量和错误量的图表曲线。

3.3. 查看凭证调用量

在凭证调用量统计中可以看到当前用户所有凭证的调用量统计信息，还可以查看单个凭证调用量统计的监控图表。

查看凭证调用量

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 凭证调用量监控**。
6. 在**凭证调用量监控**页面查看当前用户所有凭证的调用量统计信息。

   包括**调用量**、**错误量**、**最大RT(ms)**、**最小RT(ms)**和**平均RT(ms)**。

   也可以输入**凭证名称**关键字，选择**时间范围**，单击**查询**，进行查看。

查看单个凭证调用量的监控图表

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 凭证调用量监控**。

6. 在**凭证调用量监控**页面具体凭证的**操作**列，单击**查看监控图表**。
7. 在弹出的对话框中选择要查看的时间周期，单击**查询**，查看当前凭证在选定时间周期内的调用量统计监控图表。

   图表里面展示了该凭证在选定时间内的调用量和错误量的图表曲线。

3.4. 查看单一订阅调用量

在单一订阅调用量统计中可以看到当前用户所有单一订阅的调用量统计信息，还可以查看单个单一订阅的调用量统计监控图表。

查看单一订阅调用量

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 单一订阅调用量统计**。
6. 在**单一订阅调用量统计**页面查看当前用户所有单一订阅的调用量统计信息。

   包括**调用量**、**错误量**、**最大RT(ms)**、**最小RT(ms)**和**平均RT(ms)**。

   也可以输入**服务名称**和**凭证名称**关键字，选择**时间范围**，单击**查询**，进行查看。

查看单个单一订阅服务的调用量监控图表

1. 登录CSB控制台。
2. 在顶部菜单栏选择**地域**。
3. 在左侧导航栏单击**实例列表**。
4. 在**实例列表**单击具体实例名称。
5. 在**实例概览**页面左侧导航栏选择**调用统计 > 单一订阅调用量统计**。
6. 在**单一订阅调用量统计**页面具体单一订阅的**操作**列，单击**查看监控图表**。
7. 在弹出的对话框中选择要查看的时间周期，单击**查询**。

   图表里面展示了该单一订阅在选定时间内的调用量和错误量的图表曲线。

   > **说明** 一条单一订阅由凭证和服务共同确定，所以单一订阅调用量监控图标标题会同时显示该调用的凭证和服务名称。