

简介

1.基本描述

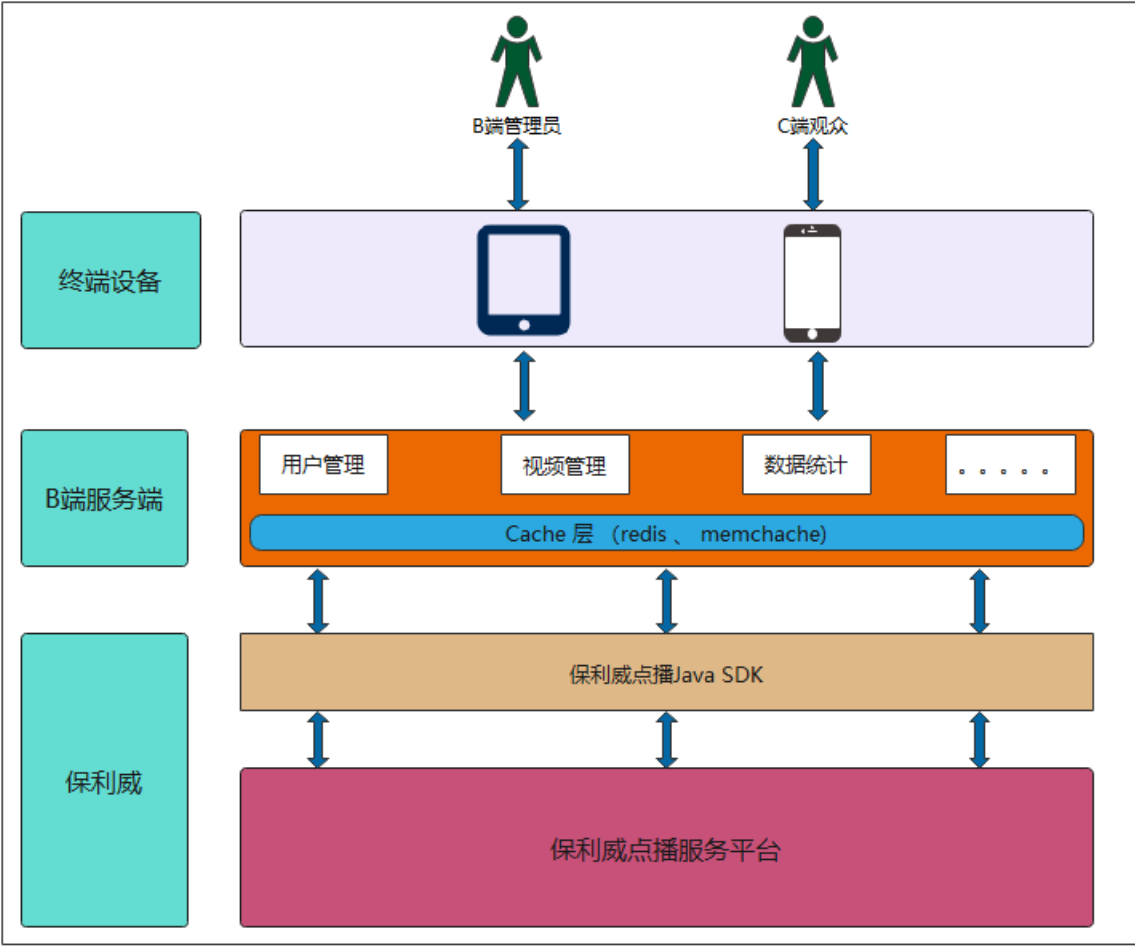
保利威点播源自公司多年视频技术沉淀，基于专业的视频加密技术、跨平台视频编解码技术和大规模视频内容分发网络，提供稳定流畅、安全、高并发的音视频服务。

保利威点播Java SDK让您不用复杂编程即可轻松接入保利威点播视频云服务，实现云点播相关视频服务。

保利威点播Java SDK依托保利威点播API实现，对其进行包装和优化，解放B端用户的共性工作。将API调用逻辑和异常处理进行了封装优化，B端用户只需将请求参数封装后，交给**保利威点播Java SDK**处理即可，**保利威点播Java SDK**处理完成后返回结果，B端依据返回数据继续完成B端业务逻辑。现阶段**保利威点播Java SDK**涵盖了视频上传、视频编辑、视频打点、字幕管理、弹幕管理、课件管理等 绝大部分经常使用的API操作。

如果您在使用**保利威点播Java SDK**的过程中遇到任何问题，直接使用[在线客服](#)找到售后技术支持提问。请将问题的运行环境、操作步骤、错误反馈信息、联系方式同步提交，便于问题的快速定位和解决。

2.SDK整体设计



- 1、B端管理员通过SDK上传视频信息，比如视频、视频水印、视频预览图、视频字幕等基本信息。
- 2、B端管理员通过SDK对视频进行编辑、管理等操作，比如视频基本信息、视频分类、加密设置、视频进度条打点等操作。
- 3、B端管理员把视频地址分发给C端观众，或者把嵌入播放器的观看页面地址分发给观众。
- 4、C端观众通过打开观看地址的链接或者二维码扫码观看B端管理员上传的视频。
- 5、B端管理员可查看C端观众的观看情况和统计分析数据，完成点播业务闭环。



3.SDK详细设计

image-202010231012489999

4.SDK业务逻辑分析

- 前置条件：**SDK全局初始化**，在调用SDK前必须配置全局参数，可配置参数包括账户信息（userId、secretKey）和HTTP链接池参数（timeout、maxClientNum），具体请见 [初始化](#)。
- SDK全局参数注入：SDK将全局配置的参数，注入到请求对象中。
- 签名生成：SDK采用点播签名规则，生成签名。
- 参数合法性校验：SDK采用自定义参数校验工具对输入参数进行校验，**如有参数不合格，将抛出 PloyvSdkException异常，exception的message包括具体校验不通过的字段信息，此异常是运行时异常，必须捕获处理相关业务逻辑。**
- 发送HTTP请求，获取返回数据：SDK在初始化阶段初始化了一个HTTP链接池，所有SDK请求都是通过该链接池来发送请求。
- 解析返回数据：解析返回数据，如SDK调用正常成功，将封装响应对象，正常返回，**如服务器返回错误信息，SDK将抛出PloyvSdkException异常，exception的message包括具体服务器执行错误信息，此异常是运行时异常，必须捕获处理相关业务逻辑。**

5.调用流程模板

上述业务流程图解析到代码层次如下，所有对**保利威点播Java SDK**的调用都可以参考如下调用模板（全局初始化只需要全局调用一次）。

```
public static void main(String[] args) {  
    String userId = "xxx";  
    String secretKey = "xxx";  
    VodGlobalConfig.init(userId, secretKey);  
    log.debug("--初始化完成--");  
    try {  
        //封装API请求对象  
        Calendar instance = Calendar.getInstance();  
        instance.set(year: 2020, month: 10, date: 13);  
        VodAccountSpaceDataRequest vodAccountSpaceDataRequest = new VodAccountSpaceDataRequest();  
        VodAccountSpaceDataResponse vodAccountSpaceDataResponse;  
        vodAccountSpaceDataRequest.setDate(instance.getTime());  
        //调用SDK请求保利威服务器  
        vodAccountSpaceDataResponse = new VodAccountServiceImpl().getAccountSpaceFlow(vodAccountSpaceDataRequest);  
        Assert.assertNotNull(vodAccountSpaceDataResponse);  
        //正常返回做B端正常的业务逻辑  
        if (vodAccountSpaceDataResponse != null) {  
            log.debug("测试获取用户空间及流量情况成功,{},", JSON.toJSONString(vodAccountSpaceDataResponse));  
        }  
    } catch (PloyvSdkException e) {  
        //参数校验不合格 或者 请求服务器端500错误，错误信息见PloyvSdkException.getMessage()  
        log.error(e.getMessage(), e);  
        // 异常返回做B端异常的业务逻辑，记录log 或者 上报到ETL 或者回滚事务  
    } catch (Exception e) {  
        log.error("SDK调用异常", e);  
    }  
}
```

全局初始化

请求参数封装

保利威点播SDK业务逻辑

正常业务返回业务逻辑处理

异常返回业务逻辑处理

6.依赖组件版本说明

目前最新版本的保利威 Java SDK 使用的常见组件信息如下。

组件	Maven 坐标	版本
Lombok	<code>org.projectlombok:lombok</code>	1.18.16
Apache HttpClient 4	<code>org.apache.httpcomponents:httpclient, httpcore</code>	4.5.13
SLF4J API	<code>org.slf4j:slf4j-api</code>	1.7.30
Gson	<code>com.google.code.gson:gson</code>	2.8.5
Fastjson 1.x	<code>com.alibaba:fastjson</code>	1.2.83
Jackson	<code>com.fasterxml.jackson.core:jackson-core, jackson-databind, jackson-annotations</code>	2.13.0

Jackson 组件相关建议

如果您的项目也使用了 Jackson 组件，可能存在的风险：当运行时出现 **Jackson 小版本不一致**（例如 `jackson-databind` 与 `jackson-core / jackson-annotations` 不同小版本，或与你业务中其他依赖要求不一致），可能触发：

- `NoSuchMethodError`
- `NoClassDefFoundError`
- `ClassCastException`

建议：

1. 在你的项目中统一 **Jackson 三件套版本**（`jackson-bom` 或 Spring Boot BOM）。确保 `jackson-core / jackson-databind / jackson-annotations` 保持同一小版本。
2. 如果你的项目必须使用特定 Jackson 版本（例如因 Spring Boot 管控），可以在引入保利威 SDK 时 **排除 Jackson**，再由你项目统一提供 Jackson 版本。

示例：排除保利威 SDK 传递的 Jackson

```

<dependency>
  <groupId>net.polyv</groupId>
  <artifactId>polyv-java-live-sdk</artifactId>
  <version>xxx</version>
  <exclusions>
    <exclusion>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-annotations</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<!-- 再由你的项目统一指定 Jackson 版本（示意） -->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.fasterxml.jackson</groupId>
      <artifactId>jackson-bom</artifactId>
      <version>${jackson.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

```

其他常见组件的相关建议

slf4j-api

- Spring Boot 2.x 生态通常仍在 SLF4J 1.7.x 体系
- Spring Boot 3.x 生态升级到 SLF4J 2.0.x
- 建议：
 - 使用 Spring Boot 时，优先让 **Spring Boot BOM** 管控日志相关依赖版本；避免在业务工程里显式固定一个与 BOM 不一致的版本。
 - 若你项目显式引入了 `slf4j-simple`、`log4j-slf4j-impl` 等实现，确认只保留一个实现（否则会出现“multiple bindings/providers”类问题）。

httpclient

- HttpClient 4.x (`org.apache.http.*`) 与 HttpClient 5.x (`org.apache.hc.*`) 可并存, 一般不会有问题, 但建议避免在同一项目中混用两套版本。

排查依赖冲突的推荐方法

在你的项目根目录执行 (Maven):

1. 查看最终生效版本 (建议加 `-Dverbose`):

```
mvn dependency:tree -Dverbose
```

2. 定位特定组件 (示例: Jackson / HttpClient):

```
mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core:jackson-databind
mvn dependency:tree -Dverbose -Dincludes=org.apache.httpcomponents:httpclient
```

3. 判断是谁引入了某个版本 (观察树上“nearest-wins”路径), 再决定:

- 通过 `dependencyManagement` 统一版本
- 或在引入点上做 `exclusions`