### JS通信使用方法说明书

|  |  |
| --- | --- |
| 发布日期 | 2017.1.17 |
| 控制级别 | 内部资料 |
| 制定部门 |  |

* 文档属性

|  |  |
| --- | --- |
| 属性 | 内容 |
| * 用户名称： |  |
| * 文档标题： | * JS通信使用方法说明书 |
| * 文档编号： |  |
| * 版本日期： | * 2017-01-17 |
| * 发布版本： | * V1.0 |
| * 适用范围： |  |
| * 作者： | * 杜鑫 耿远超 |

* 文档审批

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 审批人 | 职务 | 审批时间 | 审批意见 | 发文编号 |
|  |  |  |  |  |
|  |  |  |  |  |

* 修订内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 修正章节 | 日期 | 修正人 | 变更记录 |
| * 1.0 |  | * 2017-01-17 | * 杜鑫 | * 创建文档 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* 模板修订历史

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 生效时间 | 变更概要 | 作者 | 审核 | 批准 |
| * 1.0 | * 2017-01-17 | * 创建文档 | * 杜鑫 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 通信接口说明文档

# 1 背景

js端与前置进行数据交互的过程中，数据需要经过原生（ios/android）/模拟桥进行加、解密。因此数据在原生/模拟桥与js的传递过程中，可以进行分阶段处理。

# 2 数据包格式

根据pastry平台通信协议定义的数据包格式，原生/模拟桥从前置获取的是双层数据包。解密之后基本格式如下：

// 外层数据包

{

"pkgFlag" : "0",

"errCode" : "0",

"errMsg" : "",

// 内层数据包

"dataPackage" : {

"cryptFlag" : "0",

"hashFlag" : "0",

"hash" : "hash值",

"signatureFlag" : "0",

"signature" : "签名值",

"business" : "{\"xxx\":\"yyyyy\", \”xxx1\”:\”yyyyy1\”}"

}

PtSessionState: “4” //扩展字段 由原生写入，js读取；描述会话状态 具体描述见《pastry平台通信协议V1.0.1》

}

具体详见《pastry平台通信协议V1.0》

# 3 目标

设计：功能分离，明确各模块功能，

业务：简化js端操作，只需关注业务数据，不必管理非业务状态

# 4 设计思路

## 4.1 数据层面

Cordova保存原生与js的通信状态。

ptHttp接收到的是完整的外层数据包，在此阶段可以对外层数据包的错误信息进行处理；

mlHttp接收到的是内层数据包中的业务数据，在此阶段可以对内层数据包的错误信息进行处理；

正常情况下，业务端接收到的是交互成功的内层数据包。

## 4.2 逻辑层面

Cordova封装了js与原生的交互方法，实现js与原生数据交互。Cordova桥接收的是原生与js通信状态的描述；通信成功则ptHttp接收到双层数据包；约定不支持通信失败，只存在通信成功。

ptHttp是通过cordova与原生交互的；所有的请求都是在此组装后经原生加密发往前置；从前置接收的数据包经原生解密之后，在此开始传递到js端；

mlHttp是在ptHttp的基础上，根据接口所需要的参数，进行了再次封装；并且对token和accessToken进行统一管理，减少业务端的操作。

# 5 通信流程

## 5.1整个通信流程

发送请求：Js端--->mlHttp--->ptHttp--->原生端--->前置--->服务器

接收响应：服务器--->前置--->原生端--->ptHttp--->mlHttp--->js端

本文档关注的是到原生端的通信

## 5.2各层级

Js端：调用postNotoken/postToken/postAuthToken方法，配置参数，发送请求

mlHttp: 定义postNotoken/postToken/postAuthToken方法；调用ptHttp.post()方法；

ptHttp：定义ptHttp.post()方法；调用window.corrdova.exec()方法

## 5.3 回调说明（原生到js）

其中这些方法都会传递成功回调（success callback）和失败回调(error callback)。他们的所代表的含义也不同。

从源头window.cordova.exec()说起，success 代表基于cordova的js与原生通信桥成功的回调，error 是通信桥失败的回调。分别代表的js和原生成功建立连接和建立连接失败之后的回调。向后的整个传递过程都是在 success 的基础之上的。

因此，ptHttp接收的success和error都是在window.cordova.exec的success中。（详见流程图）

从Js端传进来的success和error是针对接口请求有没有得到预期结果的。Success 表示的是成功得到预期结果的回调。Error 是没有得到预期结果的回调。

# 6 具体实现

## 6.1 ptHttp

ptHttp是js与原生的联通桥；建立在cordova基础上。所有的js端请求都是通过ptHttp转发给原生，进行加密、再处理。

ptHttp有两个功能。密钥协商和转发请求。

密钥协商是在原生与前置握手成功之后进行密钥配送的步骤。原生端在程序启动时就会与前置进行握手及协商密钥，ptHttp提供了会话过期时重新协商密钥的方法。

转发请求分为网页请求和真机请求。

网页请求方法使用angular$http服务，允许客户端通过网页发送接口请求，模拟桥代替原生，经过模拟桥加密之后发往前置；接收到前置返回之后也是先在模拟桥解密之后再发往js。

并且，在接收到完整双层数据包之后，会先对外层数据包进行处理；如果外层数据包报错；会直接弹窗提示，并且终止向后交互传递数据。只有外层数据包正确时才会剥离内层数据包业务信息，向后传递给mlHttp。

## 6.2 mlHttp

mlHttp是在ptHttp的基础上，根据请求所需要的参数，对请求方法进行了分类封装。并且集中管理token和authToken令牌，以期简化业务端操作。

在对token管理中，提供了token对象的get和set方法，以便获取和设置token；

对authToken也提供了get和set方法，功能一样。

根据请求对token和authToken的依赖，提供了三种接口请求方法：

postNoToken： 该类请求不依赖token或authToken

postToken：该类请求依赖token不依赖authToken

postAuthToken：该类请求依赖token和authToken

各类的依赖在mlHttp中会自动添加；无需业务端传递。

其中postNoToken只适用获取token请求，并在获取成功之后调用set方法进行管理。

另外在postToken在用于登录申请时，会对成功回调进行改造，登陆成功之后会自动存储authToken，无需业务端管理。

并且，mlHttp在接收到ptHttp传递的内层业务数据之后，也会对数据进行预处理；如果业务数据报错；会直接弹窗提示，终止向业务传递；只有业务数据正确无误时，才会把数据传递给业务操作。

# 7 常见错误情况分析

## 7.1 外层数据包的常见错误及对应错误码

PTComPackage 的外层数据包 errCode :

1、 前置框架定义的状态码0：请求成功

2、 框架状态码1：会话状态错误

3、 框架状态码2：网络不通

4、 框架状态码3：未知的异常错误

5、 服务器状态码500：

6、 服务器错误404：

7、 前置框架自定义状态码 10001： 自定义状态码1

8、 前置框架自定义状态码 10002： 自定义状态码2

其中，分为3类错误：

1：框架报错 ；此时数据还在客户端；请求还未发出

2：前置报错；此时数据传输到前置，但是数据有错误，导致前置无法处理、转发

3：服务器错误；此时数据传输到服务器；可能会出现服务器连接错误或连接成功处理数据错误。

## 7.2 内层数据包常见错误及对应错误码

内层数据包的错误码、错误信息通常是由后台制订；不同的项目有不同通的规则。