特别提示：

* 本文件必须按照“必须知道，最少授权”的原则交付，不得全部或部分提供给任何未经授权的人。
* 如果本文件不在您的访问权限范围内，应立即销毁，不得存储、传输、复制、打印或使用本文件的任何内容。

移动开发平台

客户端组件详细设计说明书

2023年03月29日

文档修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | \*变化  状态 | 修订说明 | 日期 | 变更人 | 备注 |
| V0.1 | A | 新建 | 2023/03/29 | persilee |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

\*变化状态：A——增加，M——修改，D——删除

目录

[1 引言 2](#_Toc893198940)

[1.1 编写目的 2](#_Toc1083892050)

[1.2 背景 2](#_Toc2017390496)

[1.3 术语定义 2](#_Toc1810247436)

[1.4 参考文档 2](#_Toc1427829803)

[2 通讯组件 3](#_Toc1553227443)

[2.1 系统功能 3](#_Toc282421569)

[2.1.1 功能架构 3](#_Toc720450313)

[2.1.2 功能概述 3](#_Toc1095608805)

[2.2 系统设计 4](#_Toc1372396257)

[2.2.1 Dio的封装 4](#_Toc1889522619)

[2.2.2 基础请求体 4](#_Toc218485697)

[2.2.3 基础响应体 5](#_Toc2039556756)

[2.2.4 API模块 5](#_Toc696424678)

[2.2.5 models模块 5](#_Toc1023686996)

[3 存储组件 6](#_Toc1615845655)

[3.1 系统功能 6](#_Toc439723623)

[3.1.1 系统架构 6](#_Toc943702434)

[3.1.2 功能概述 6](#_Toc1640075143)

[3.2 系统设计 7](#_Toc1790319156)

[4 元数据组件 8](#_Toc1500676775)

[4.1 系统功能 8](#_Toc1826607057)

[4.1.1 系统架构 8](#_Toc1506073134)

[4.1.2 系统概述 8](#_Toc181415949)

[4.2 系统设计 8](#_Toc1778559750)

[5 国际化组件 8](#_Toc1428835657)

[5.1 系统功能 8](#_Toc1278746445)

[5.1.1 系统架构 8](#_Toc2022645586)

[5.1.2 系统概述 9](#_Toc2085715539)

[5.2 系统设计 9](#_Toc1245493992)

[6 数据埋点组件 9](#_Toc1494416235)

[6.1 系统功能 9](#_Toc1832409980)

[6.1.1 系统架构 9](#_Toc251552233)

[6.1.2 系统概述 9](#_Toc1590562735)

[6.2 系统设计 10](#_Toc711449289)

# 引言

## 编写目的

本文档是详细说明了用户中心的整体架构、功能列表、业务流程设计、接口设计、数据库设计和与其他系统的接口关系。在项目实施阶段，将依据本文档进行系统的开发。

## 背景

## 术语定义

|  |  |
| --- | --- |
| 术语 | 定义 |
|  |  |

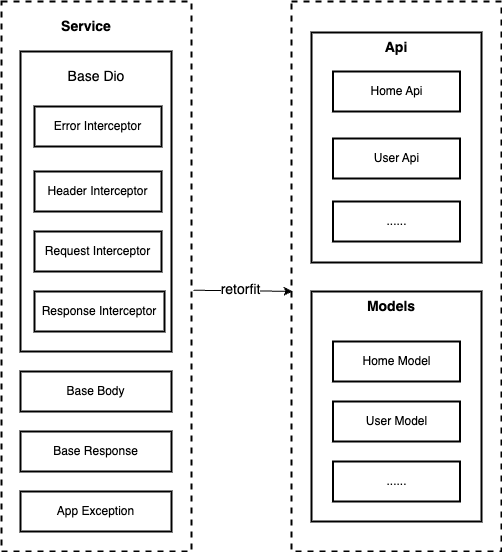
## 参考文档

1. 移动开发平台概要设计文档.docx

# 通讯组件

## 系统功能

### 功能架构



### 功能概述

通过统一封装，可以快速的接入App实现网络请求；可以灵活的切换通讯协议和通讯地址，底层网络请求库无耦合，可根据业务需求灵活更换以及灵活切换环境；可以处理通讯报文的加密。

可用于应用的请求服务端来获取数据，处理报文的加密，处理多个环境切换，如，DEV、SIT、UAT、PROD。

## 系统设计

整体结构分为Service、Api、Models三层：

* Service：对dio的封装和常见错误码的封装以及基础请求体和响应体的定义。
* Api：根据dio和retorfit自动生成的api实现类。
* Models：自动生成的序列化实例类。

### Dio的封装

Dio的封装是通过重写4个拦截器，可以自由的控制请求、响应、异常。

* **Error Interceptor**：错误拦截器，统一处理异常问题
* **Header Intercepto**r：请求头拦截器，处理请求的header，请求头数据如下

**Access-Version**: 2

**Accept-Language**: km-KH

**Access-Token**:

**contentType**: application/json; charset=utf-8

**responseType**: ResponseType.json

**followRedirects**: true

**connectTimeout**: 0:00:30.000000

**receiveTimeout**: 0:00:30.000000

* **Request Interceptor**：请求拦截器，拦截请求数据，进行数据的二次加工，如：调整数据结构，加密等
* **Response Interceptor**：响应拦截器，拦截响应数据，进行数据的二次加工，如：调整数据结构，加密等

### 基础请求体

基础请全体是一个固定不变的结构，用于包装业务请求体，内容如下：

**data**：业务请求数据

**key**：应用的key

**sign**：应用的签名

**appId**：应用的Id

### 基础响应体

基础响应体是一个固定不变的结构，用于包装业务响应体，内容如下：

**system** : "wallet-app-backend"

**state** : 1

**code** : "PARAM\_ILLEGAL"

**message** : "请求参数有误"

**data** : {"authId":"2022100911014910001001","effectiveTime":120}

**detail** : [{"field":"data.mobile","value":"null","issue":"mobile NotBlank"}]

### API模块

API模块是每个业务模块的api的定义，是一个抽象类，可以通过retorfit

注解自动生成实体类，可以节省大量编码时间。

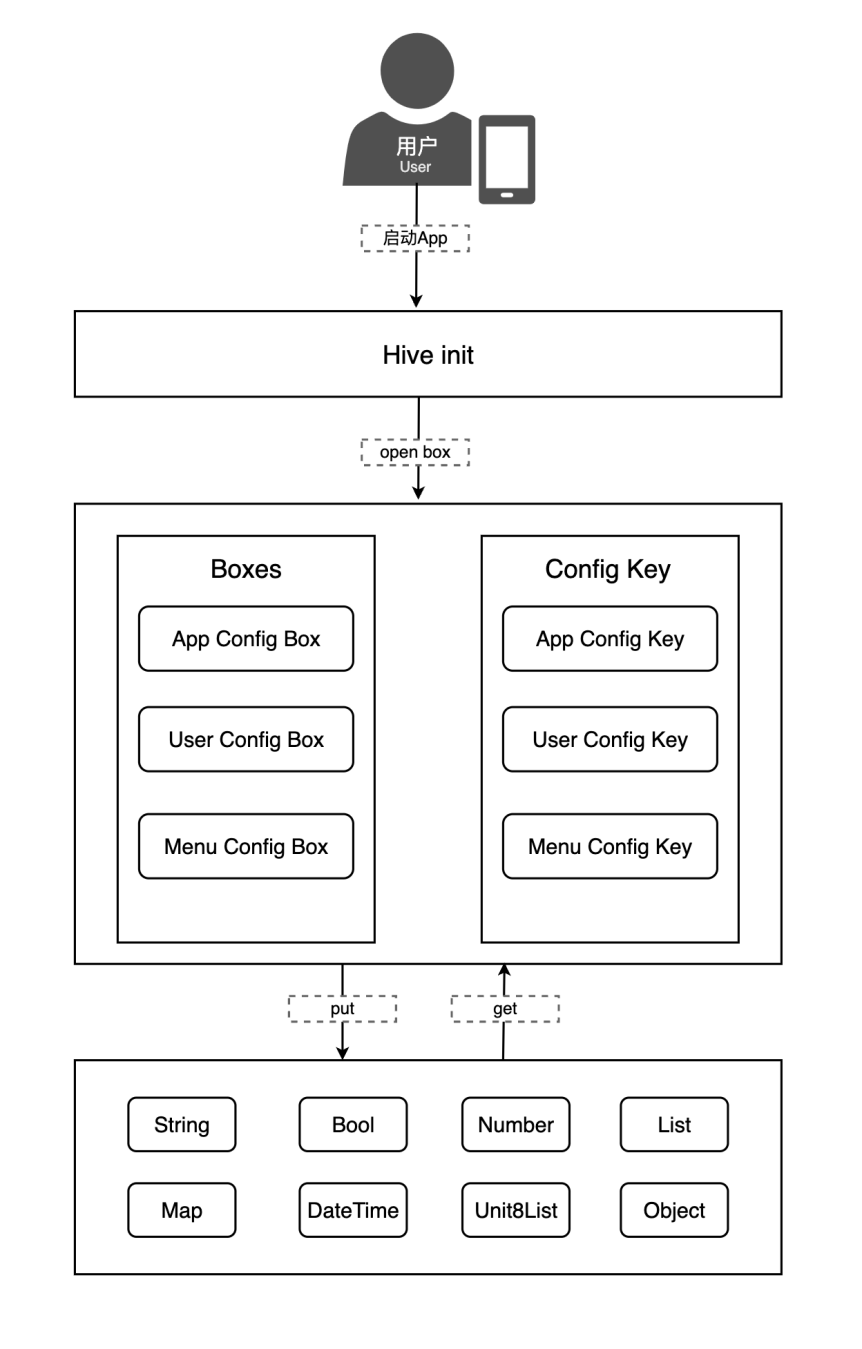
### models模块

Models模块是一些用于序列化的实例类，通过 json\_annotation 或者编辑器插件生成的实例类。

# 存储组件

## 系统功能

### 系统架构



### 功能概述

存储组件提供完整的持久化解决方案，组件提供了多种存储方式以满足不同的存储需求。目前，存储组件主要由Hive组成。

[Hive](https://links.jianshu.com/go?to=https://pub.flutter-io.cn/packages/hive" \t "/Users/lishaoying/Documents\\x/_blank) 是一个纯 Dart 编写的、基于文件存储的、轻量且功能强大的 Key-Value 型数据库。

统一存储技术，可以应用于所有涉及数据存储、数据持久化的业务。提供客户端持久化存储解决方案，底层提供多样化的存储方式以满足不同的存储需求，支持数据加密功能。

## 系统设计

通过对Boxes和Config Key的封装，会统一给应该提供以下功能：

* **Box Name**：通过Boxes类可以方便的找到每个存储库。
* **Open Box**：通过Boxes类统一打开所有的存储库。
* **Close Box**：通过Boxes类统一关闭所有的存储库。
* **Config Key**：通过Config Key可以方便的找到每个存储库了存储的数据。

# 元数据组件

## 系统功能

### 系统架构

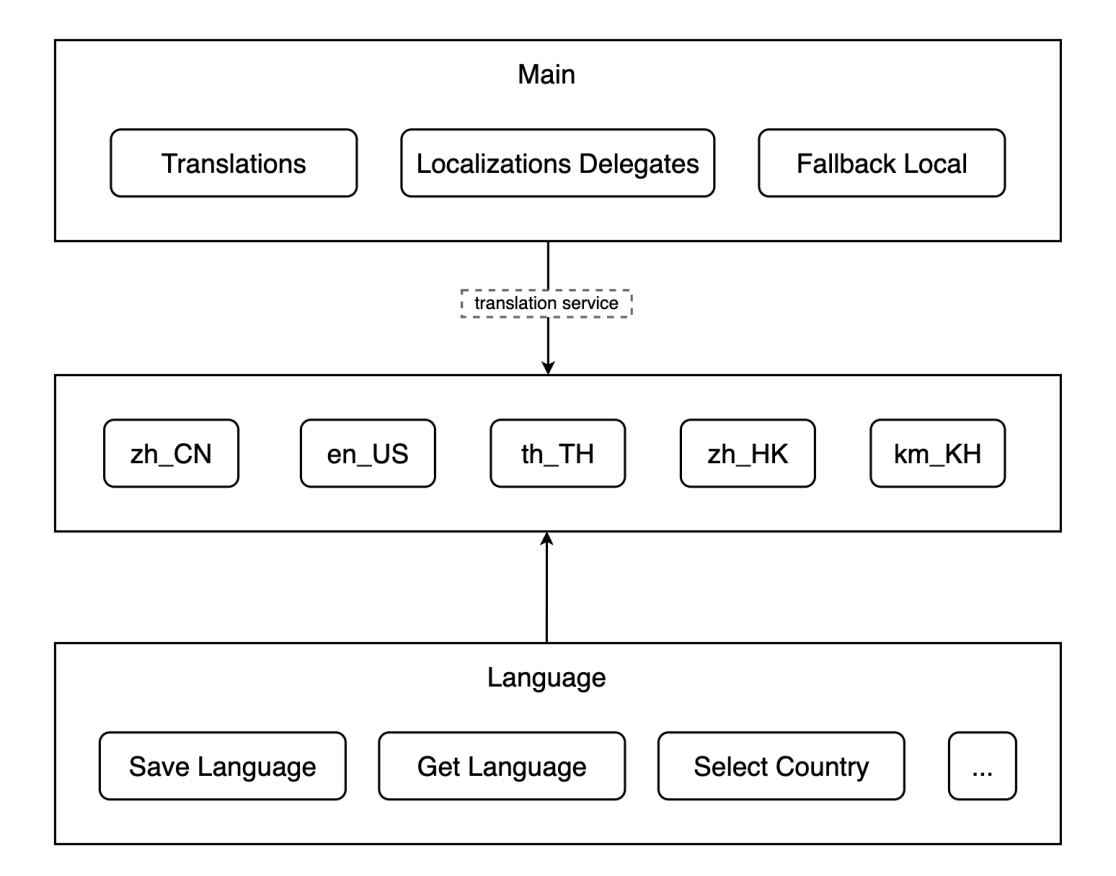
### 系统概述

## 系统设计

# 国际化组件

## 系统功能

### 系统架构



### 系统概述

国际化组件用于处理应用的多语言，是UI字符串以不同语言显示，可以在应用里进行语言的切换。

应用通过调用 [Localizations.of(context,type)](https://api.flutter-io.cn/flutter/widgets/Localizations/of.html) 来引用这些对象。如果设备的语言环境变化了，Localizations widget 会自动地加载新的语言环境的值，然后重建那些使用了语言环境的 widget。

## 系统设计

应用使用Getx实现国际化功能，通过Getx的Translation Service实现应用的多语言切换，首先，会在入口文件设置国际化的语言、代理和默认语言，如：

**Translations**：支持的语言列表

**Localizations Delegates**：本地化语言代理

**Fallback Local**：获取语言失败时的备用语言

通过Language类的封装，提供给应用设置语言、获取语言、保存语言等功能。

# 数据埋点组件

## 系统功能

### 系统架构

### 系统概述

* **功能按钮埋点**

1. 应用的顶部tab、底部tab、内容筛选tab的点击
2. 应用内各入口，如，点击了贷款模块、点击了定期模块等
3. 应用内具体的内容点击，如点击了贷款模块里的banner、点击了贷款模块里的’如何提示额度’等
4. 应用内设置选项的点击
5. 应用内功能按钮的点击

* **主要行为埋点**

1. 用户核心行为，包括登录、注册、查看贷款额度、提交贷款申请等
2. 用户其他行为，包括修改了语言、重置了密码、修改了用户名等

* **页面埋点**
* **支付流程埋点**
* **搜索流程埋点**
* **数据埋点**

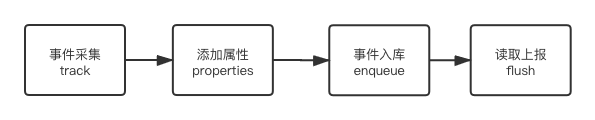
1. 用户账号、用户的UDID、机型、操作系统
2. 用户登录ip、经纬度、国家、省份、城市、区县、街道等
3. 用的操作时间，如果是页面时间还记录了开始时间和结束时间
4. 用户进行了什么操作，如应用的id、页面来源、应用所在位置、应用来源模块、是否操作成功等

## 系统设计

数据采集流程中主要包括事件采集、添加属性、事件入库、读取上报等流程，详细的步骤如下所示：

* 在产品、服务转化的某些关键点，调用埋点相关接口采集事件；
* 获取有意义的属性丰富该事件，保证数据的广度与深度；
* 数据采集完成，转换成标准 JSON 数据格式，以队列的形式存储到 SDK 的数据库内；
* 定时读取数据库中的数据，封装请求并上报数据，并在上报成功后，删除数据库内存储的已上报数据。

整体流程如图所示：



代码埋点的实现原理比较简单，主要是初始化 SDK 之后，在某个事件发生时调用相关接口，将触发的事件和属性保存到数据模型中。并将数据模型转化为 JSON 串，存储到本地数据库中。然后，按照发送策略将数据发送到指定的服务端。例如：我们想统计 App 里面某个按钮的点击次数，可以在这个按钮对应的点击方法里面调用 SDK 提供的接口来采集事件。