爱旅行项目—开发规范

爱旅行项目架构从两个方向进行设计：前端架构和后端架构，将前端和后端完全分离。整个项目开发过程由项目组的前端团队与后端团队共同协作完成，需要遵循开发规范如下：

## 1.1 后端- iTripBackend开发规范

### 1.1.1模块化开发

iTripBackend分为以下四个Web模块：

1. itrip主业务模块（itripbiz）：包括酒店业务、旅游业务、机票业务、攻略业务
2. itrip搜索模块（itripsearch）：爱旅行项目中所有搜索功能（solr）
3. itrip Auth认证模块（itripauth）：负责用户身份验证，生成并维护Token
4. itrip支付模块（itrptrade）：包括支付宝支付、微信支付

对于上述web模块，由于存在一些公用的部分（比如：pojo、dao、utils等），为提高代码复用性及可维护性，故采用Maven进行多Module管理，具体结构入下：

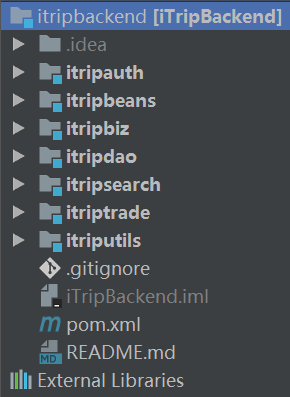


图1 项目模块结构

### 1.1.1 命名规范

#### 1.1.1.1 包名

1. 统一使用小写字母
2. 以cn.itrip开头，代码结构层次清晰
   1. cn.itrip.controller
   2. cn.itrip.service
   3. cn.itrip.dao
   4. cn.itrip.beans
      1. cn.itrip.beans.vo
      2. cn.itrip.beans.pojo
      3. cn.itrip.beans.dto
   5. cn.itrip.common
3. 配置文件统一放置在/resources目录下

#### 1.1.1.2 类名

1. 统一使用小写字母
2. 以cn.itrip开头，层次清晰
   1. cn.itrip.controller
   2. cn.itrip.service
   3. cn.itrip.dao
   4. cn.itrip.beans
   5. cn.itrip.common
3. 配置文件统一放置在/resources目录下

### 1.1.2 常用字典对照表

Token的数据结构为Key-Value，具体内容如下：

### 1.1.6 常用字典对照表

#### 1.1.6.1 常用值

|  |  |  |
| --- | --- | --- |
| **名称** | **值** | **说明** |
| 返回结果 | 0：失败/否/未激活/有待改善  1：成功/是/已激活/值得推荐 | 适用于：  数据库是否的标识值 |
| 性别 | 0：女  1：男 |  |
| 区域级别 | 0: 国家级  1: 省级  2: 市级  3: 县/区 |  |
| 国家 | 1: 国内  2：国外 |  |
| 商品类型 | 0: 旅游产品  1: 酒店产品  2: 机票产品 |  |
| 酒店级别 | 1: 经济酒店  2: 二星级酒店  3: 三星级酒店  4: 四星级酒店  5: 五星级酒店 |  |
| 订单类型 | 0: 旅游产品  1: 酒店产品  2: 机票产品 |  |
| 订单状态 | 0：待支付  1: 已取消  2: 支付成功  3: 已消费 |  |
| 订单支付方式 | 1: 支付宝  2: 微信  3: 到店付 | 订单所选择的支付方式 |
| 发票类型 | 0：个人  1：公司 |  |
| 客户端预定类型 | 0: WEB端  1: 手机端  2: 其他客户端 |  |
| 酒店预定支付方式 | 1: 在线付  2: 到店付  3: 不限 | 预定酒店所规定的支付方式 |
| 图片类型 | 0: 酒店图片  1: 房间图片  2: 评论图片 |  |
| 用户类型 | 0: 自注册用户  1: 微信登录  2: QQ登录  3: 微博登录 |  |
| 证件类型 | 0: 身份证  1: 护照  2: 学生证  3: 军人证  4: 驾驶证  5: 旅行证 |  |

## 常用名称

|  |  |
| --- | --- |
| **名称** | **英文** |
| 商品 | goods |
| 货品 | product |
| 规格 | spec |
| 订单 | order |
| 会员 | member |
| 积分 | point |
| 购物车 | cart |
| 结算 | checkout |
| 订单 | order |
| 品牌 | brand |
| 分类 | cat |
| 优惠劵 | coupon |
| 支付 | payment |
| 团购 | groupbuy |
| 虚拟 | virtual |
| 发票 | receipt |
| 属性 | prop |
| 参数 | param |
| 标签 | tag |
| 地区 | region |

## 1.2 前端- iTripFront开发规范

Token的数据结构为Key-Value，具体内容如下：

1. Key：token，其设计原则：必须保证在整个系统中唯一存在

根据不同客户端(PC、移动)，为了便于统一管理和维护，token生成算法设计如下：

PC端：token:PC-USERCODE[加密]-USERID-CREATIONDATE-RONDEM[6位]

移动端：token: MOBILE-USERCODE[加密]-USERID-CREATIONDATE-RONDEM[6位]

1. Value：存储登录用户的信息数据（数据格式为json），具体内容包括：
2. id：用户表主键ID
3. userCode：登录账号
4. userName：用户昵称
5. userPassword：登录密码
6. userType：用户类型
7. flatId：平台ID
8. activated：是否激活（0：否，1：是）

## 1.3 前后端交互规范

基于系统的安全性考虑，需要设置Token的有效期，并且为了维护Token的有效期，须把Token放入到Redis里进行维护管理。对于不同客户端（PC端、移动端）的Token所设置的有效期策略不同。

### 1.3.1 PC端

Token的有效期为2个小时，若2个小时内没有进行Token置换的话，就会自动在Redis里清除该Token，那么当该用户再次发送请求时，则会提示：Token失效，请重登录。此处应注意：前端须自行管理Token的生命周期，原因是Token存在cookie里，web的安全性较差。

### 1.3.2 移动端

Token永不失效，修改密码后须更换Token。

注意：由于移动端的Token一般不需要过期，只有当在PC页面进行个人密码修改后，移动端才会退出重登录，或者当在移动端修改密码操作，用户也不需要退出重登录，直接在Redis中更新该Token中用户修改的新密码即可。

补充说明：爱旅行项目的一期仅实现为在Auth系统里生成Token时，根据参数不同（PC/MOBILE）来生成唯一的Token，存入Redis时，设置不同的有效期即可。二期新增修改个人密码功能后，再进一步实现。

## 1.4 AS系统(Auth System)设计方案

该系统主要负责登录用户身份的验证，登录成功后生成唯一的Token，并将Token存入到Redis里进行维护管理，以及Token的置换（Reload）等。具体的设计流程如图1所示：

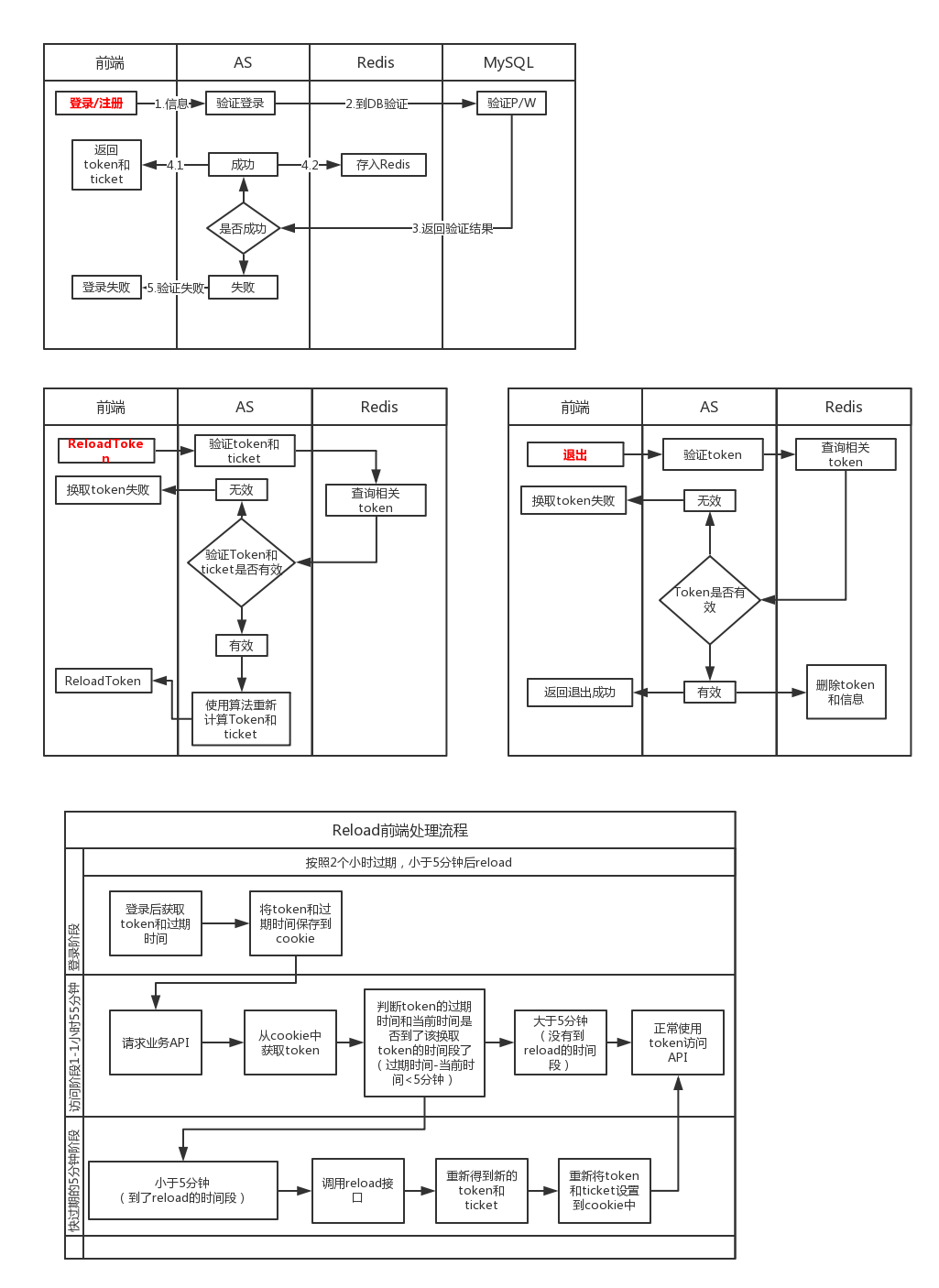


图1 爱旅行-Token设计流程

### 1.4.1 生成Token

当用户进行系统登录时，Auth系统会进行用户名和密码的校验，验证成功后生成Token，存入到Redis中，如图2所示：

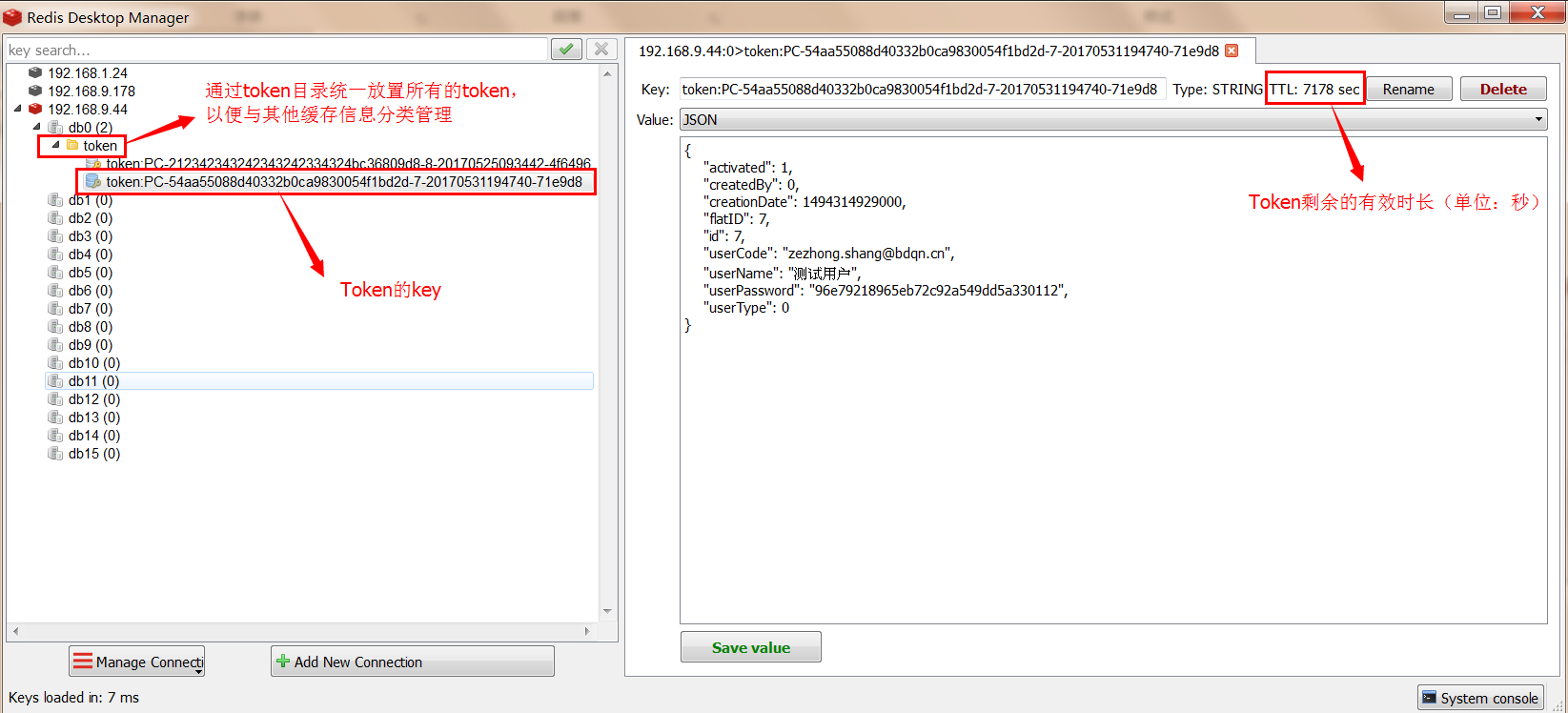


图2 Redis中统一管理Token

同时将该Token的key返回给前端，前端调用的所有API都必须传递该Token，并规定Token统一放在请求头（header）里，后端需要获取当前用户信息时，就可以直接从header中获取即可。

为了保证绝对的安全，正常业务下的API返回给前端的Token信息如下：

{

"status":"200",

"data":{

"token":"token:PC-3066014fa0b10792e4a762-23-20170531133947-4f6496",

"ticket":"yhe736dyfhfyw",

"expTime":"6474783",

"genTime":"7565746"

},

"errorcode":"0",

"msg":{}

}

通常情况下返回给前端的Token数据只有四项内容：token(key)、ticket、Token的生成时间、Token的失效时间。为了安全考虑，不会包含当前用户的任何信息，每次后端需要获取Token信息时，应从header中获取到token(key)和ticket，通过它们两个去Redis中进行k-v匹配获取相应的Token信息，即：当前用户的相关信息。

此处我们系统设计简单化处理，去掉ticket。但是Token的生成时间和失效时间必须保留，并传递给前端，这是由于仅有失效时间还不足以让前端去判断是否应该进行Token的置换，还需要Token的生成时间（即：Redis服务器时间），这也是由于爱旅行项目采用分布式架构，可能会存在服务器时间同步问题。故：前端需要拿到这两个时间进行判断。简单的理解就是：若前端服务器时间若比后端服务器时间晚两个小时，那么前端拿到Token就会过期。

### 1.4.2 Token置换

Token置换规则定义：前端获取Token的1.5时后可进行Token置换，若在最后的半个小时内，客户端发出请求，则会进行Token置换，拿到重新生成的Token（包括：token(key)、生成时间、失效时间），若客户端在最后的半个小时内没有发送任何请求，那么两个小时后自动过期，即：该Token自动从Redis里清除，用户须重新登录。

需要注意事项：

1. 不论是最后半个小时的置换时间还是Token的2个小时有效期，都是根据系统的业务需求所设计的策略方案。
2. 为了防止客户端恶意的进行Token置换，需要保证生成Token后的1个小时内不允许置换。
3. 需要保证客户端传递有效的Token进行置换。
4. 为了解决页面的并发问题，在进行置换Token时，生成新Token，但是旧Token不能立即失效，应设置为置换后的时间延长2分钟。比如：当加载在某个页面（比如：酒店详情页，如图3所示）时，该页面中至少有5个模块同时进行异步请求（房型列表、酒店介绍、酒店设施、酒店政策、用户点评），这样必然会存在严重的并发问题，那么在此处，前端必须保证只有一个请求（比如：请求）去进行Token的置换，即：置换Token的请求放置只能在页面中被调用一次，那么此时在Token的置换期间，该页面的其他请求（请求、请求、请求、请求）必定还是拿着老Token进行数据交互，故老Token需要继续保持有效，即：它的失效时间必须延长2分钟。若非如此，就会出现页面因为Token失效而无法完成置换。



图3 酒店详情页-多模块异步请求

## 1.5 爱旅行平台的各个业务系统对Token的使用

作为爱旅行项目的业务系统，则不需要对Token关心过多，只需根据业务需求，从header中获取token，然后在Redis中查询相对应的value值即可，便可获取到当前用户的相关信息，具体实现如下：

在Controller的处理方法中通过request.getHeader("token")来获取token字符串，为了方便进行Token的验证，提供统一的ValidationToken.java，该工具类主要负责通过传入的token(key)去Redis里进行value的查找（ItripUser currentUser = validationToken.getCurrentUser(token);），若找到相应的value，则返回currentUser（当前用户），若无，则返回null。

## 前/后端具体实现

### 1.6.1 后端

Auth系统需要提供API如下：

1. 生成Token

该接口返回的数据内容包括：Token的key（注：需要对敏感信息进行加密处理）、Token的生成时间、Token的失效时间（注：过期时间减去生成时间一定是两个小时）

1. Token置换

该接口返回新Token。实现过程中需要注意如下几点：

1. 生成Token后的1个小时内不允许置换（注：主要是为了防止客户端恶意的进行Token置换）
2. 由于需要保证客户端传递的置换Token为真实存在并有效的，故需要在该API方法内首先判断Token是否有效。
3. 在进行置换Token，生成新Token，旧Token不能立即失效，应设置为置换后的时间延长2分钟。

### 1.6.2 前端

1. 登录成功后，接收Token放入cookie中，请求的时候从cookie中取出放入到header里，如下：

$.ajax({

headers:{

Accept:"application/json;charset=utf-8",

Content-Type:"application/json;charset=utf-8",

//从cookie中获取

token:"token:PC-3066014fa0b10792e4a762-23-20170531133947-4f6496"

},

type："post",

.....

})

1. 负责服务器时间同步（根据API返回的Token生成时间、失效时间进行同步）
2. 置换Token需要同步处理，即：保证只有一个请求置换Token