爱旅行项目 iTrip—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 命名规范

【强制】类名、方法名、参数名、成员变量、局部变量、数据库表的字段名都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

正例： localValue / getHttpMessage() / inputUserId

下面将逐一详细说明介绍。

1.1.1.1 包名

1. 【强制】统一使用小写字母
2. 以 cn.itrip 开头，代码结构层次清晰
    a) cn.itrip.controller
    b) cn.itrip.service
    c) cn.itrip.dao
    d) cn.itrip.beans
        1> cn.itrip.beans.vo
        2> cn.itrip.beans.pojo
        3> cn.itrip.beans.dto
    e) cn.itrip.common
3. 配置文件统一放置在/resources 目录下

1.1.1.2 类名

1. Controller

以 Controller 结尾，比如：HotelController.java

2. 业务类

根据具体业务进行分包放置，比如：cn.itrip.service.itriphotel

1> 接口

以 Service 结尾，比如：ItripHotelService.java

2> 接口实现类

以 Impl 结尾，比如：ItripHotelServiceImpl.java

3. Dao

根据具体业务进行分包放置，比如：cn.itrip.dao.itriphotel

1> mapper 接口

以 Mapper 结尾，比如：ItripHotelMapper.java

2> mapper 文件

与 mapper 接口同名，比如：ItripHotelMapper.xml

4. 实体类

统一放置在 cn.itrip.beans 包下，具体分为：

1> pojo

与数据库表一一对应，并与表名同名，比如：ItripHotel.java，属性名与数据库字段名一一对应，均采用驼峰规则定义。

2> vo（view object）

根据业务功能创建各自 VO，并根据业务分包放置，命名规则：业务功能名+VO.java，（注：类名中的 VO 须大写）。

3> dto

数据传输对象，包括输出给前端的 Dto 以及接收前端传入的 InputDto。

1.1.1.3 变量/属性命名

必须统一遵从驼峰形式。前后端变量名、以及数据库字段名都使用同一套命名规则，可有效减少出错率，提高工作效率。

【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。

正例： MAX_STOCK_COUNT

反例： MAX_COUNT

1.1.1.4 方法命名

【参考】service/dao 层方法命名规约：

1> 获取单个对象的方法用 get 做前缀。

2> 获取多个对象的方法用 list 做前缀。

3> 获取统计值的方法用 count 做前缀。

4> 插入的方法用 save 或 insert 或 add（推荐）做前缀。

5> 删除的方法用 remove 或 delete（推荐） 做前缀。

6> 修改的方法用 update 做前缀。

1.1.3 注解的使用

【强制】该项目的 service 层和 controller 层均统一使用@Resource 注解，不允许使用@Autowired 注解。

1.1.4 模块化开发

iTripBackend 分为以下四个 Web 模块：

1. 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. 搜索模块（itripsearch）：爱旅行项目中所有搜索功能（solr）
3. 认证模块（itripauth）：负责用户身份验证，生成并维护 Token
4. 支付模块（itrptrade）：包括支付宝支付、微信支付

对于上述 Web 模块，都存在一些公用的部分（比如：bean、dao、utils），为提高代码复用性及可维护性，故采用 Maven 进行多 Module 管理，一共 7 个 Module(itripbiz、itripsearch、itripauth、itriptrade、itripdao、itriputils、itripbeans)，具体 Module 结构及 atifactId 如图 1、2 所示：

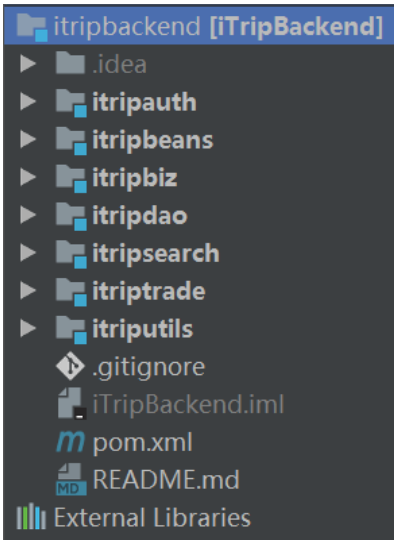

图 1　iTripBackend-模块结构

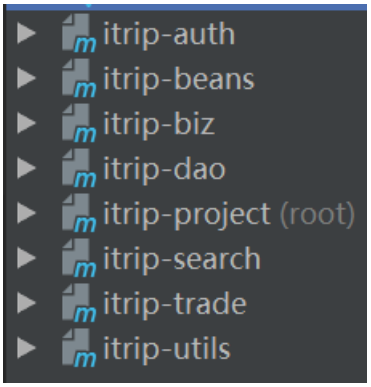

图 2　artifactId

注意：各模块之间存在依赖，切不可存在相互依赖。依赖关系如下：

1. itripdao 模块 depedency：itrip-utils
2. itriputils 模块 depedency：itrip-beans
3. itripbiz 模块 depedency：itrip-dao
4. itripsearch 模块 depedency：itrip-utils
5. itriptrade 模块 depedency：itrip-dao
6. itripauth 模块 depedency：itrip-dao

各模块具体说明如下：

1. itripbeans：包括 dto、pojo、vo

   注：该模块需要打成 jar 包，供其他 Web 模块使用

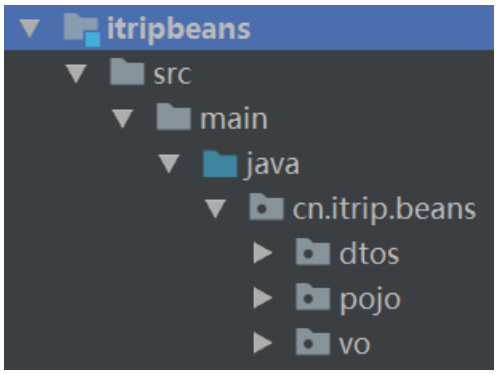

图 3 itripbeans

2. itripdao：包括 dao 层所有代码

   注：该模块需要打成 jar 包，供其他 Web 模块使用

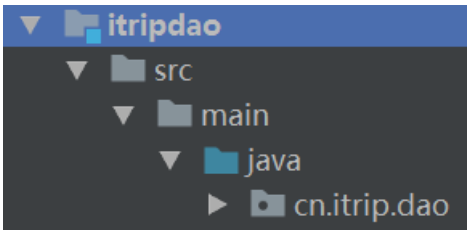

图 4 itripdao

3. itriputils：包括 common 包下所有工具类

   注：该模块需要打成 jar 包，供其他 Web 模块使用

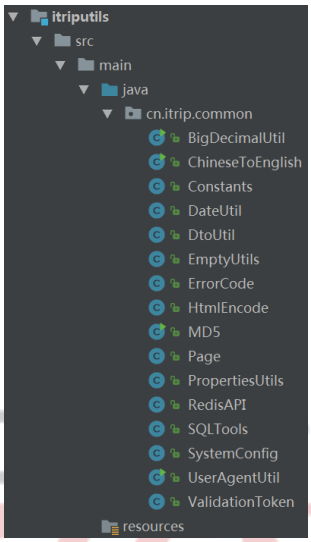

图 5 itriputils

4. itripbiz：包括 controller、service 层以及 resources、webapp

   注：该 web 模块需要打成 war 包，部署路径：/biz

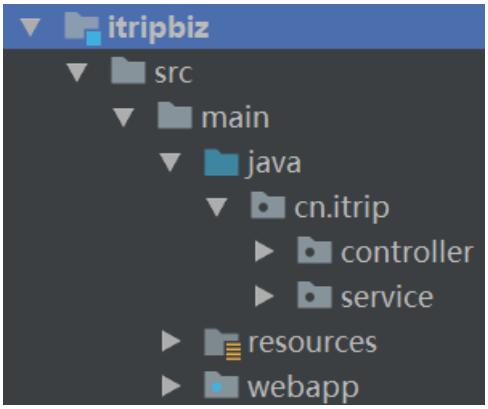

图 6 itripbiz

5. itripsearch：包括 controller、service、dao 层以及 resources、webapp

   注：该 web 模块需要打成 war 包，部署路径：/ search

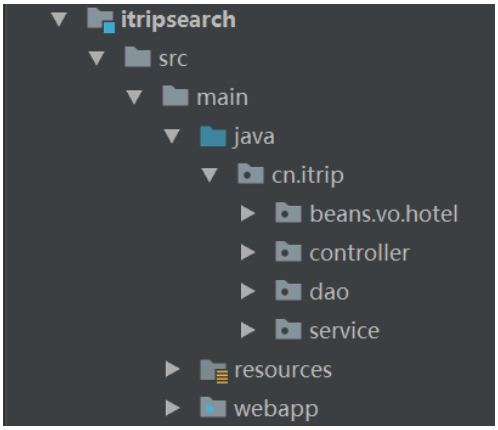

图 7 itripsearch

6. itripauth：包括 controller、service 层以及 resources、webapp

   注：该 web 模块需要打成 war 包，部署路径：/ auth

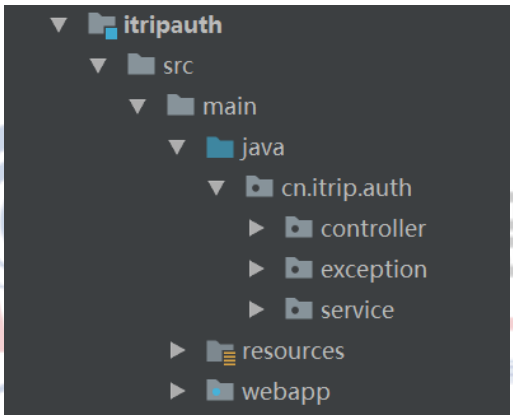

图 8 itripauth

7. itriptrade：包括 controller、service 层以及 resources、webapp

   注：该 web 模块需要打成 war 包，部署路径：/ itrade

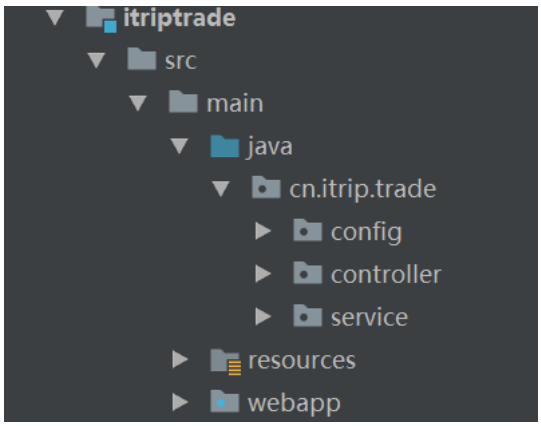

图 9 itriptrade

基于以上多模块开发，需要遵守以下的代码约定：

1.1.4.1 itripbeans 模块

1. pojo：代码自动生成
2. dto：数据传输对象，包括输出前端 Dto 以及接收前端传入的 InputDto，数据结构如下：

1> Dto .java

```
public class Dto<T>{
    private String success; //判断系统是否出错做出相应的true或者false的返回，与业务无关，出现的各种异常
    private String errorCode;//该错误码为自定义，一般0表示无错
    private String msg;//对应的提示信息
    private T data;//具体返回数据内容(pojo、自定义VO、其他)
```

图 10 dto

根据 service 返回的结果，Controller 层的 API 定义往前端输出的数据内容（json 格式），e.g：

图 11 正确返回

图 12 错误返回

注意：错误码 errorCode 根据业务定义

业务模块	模块规则
itripbiz	1 开头（100000）
itripsearch	2 开头（200000）
itripauth	3 开头（300000）
itrptrade	4 开头（400000）

2> InputDto.java

```
public class InputDto {

    @ApiModelProperty(value="单一参数传入")
    private String paramString;
    @ApiModelProperty(value="多个参数传入")
```

图 11 Inputdto

3. vo：根据业务功能创建各自 VO，并根据业务分包放置，如图 12 所示：

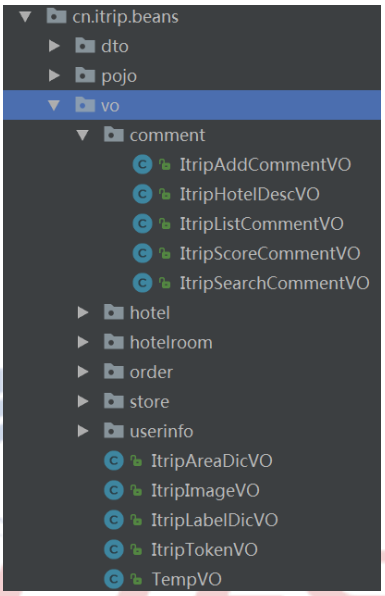

图 12 vo

注意：VO 是前后端参数传递的接收对象，需要加入 Swagger 的@ApiModel，进行属性的详细说明及输入要求。

1.1.4.2 itripdao 模块

该模块的初始化代码（普通的增、删、改、查）均是自动生成，若因业务需求，则需新增接口方法。

1.1.4.3 itriputils 模块

所有的工具类都放置在此模块内，注意：不能使用注解，以降低耦合度。

1.1.4.4 itripbiz 模块

1. service 层

   1> 均需 throws Exception，以便事务控制。

   2> 若业务需进行事务处理，则需在接口方法名加上前缀 itriptx，比如：

   public Integer itriptxModifyItripHotelOrder(ItripHotelOrder itripHotelOrder)throws Exception;

2. controller 层

   1> 提供 API 接口，各 web 工程都需要加入 Swagger，以便给前端生成 API 文档

   2> Controller 必须加统一前缀：/api/***，如图 13 所示：

```
@Controller
@Api(value = "API", basePath = "/http://api.itrap.com/api")
@RequestMapping(value="/api/comment")
public class SystemCommentController {
```

注：其他模块（itripsearch、itriptrade、itripauth）与该模块规范一致，此处不再赘述。

1.1.5 常用字典对照表

1.1.5.1 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 4：已点评	
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	

1.1.5.2 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
优惠券	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

1.1.7 注释规约

1. 【强制】方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释使用/* */注释，注意与代码对齐。
2. 【推荐】与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持英文原文即可。

   反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。
3. 【推荐】代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改。

   说明：代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后，就失去了导航的意义。
4. 【参考】注释掉的代码尽量要配合说明，而不是简单的注释掉。

   说明：代码被注释掉有两种可能性：

   1）后续会恢复此段代码逻辑。

   2）永久不用。前者如果没有备注信息，难以知晓注释动机。

   后者建议直接删掉（代码仓库保存了历史代码）。
5. 【参考】对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作。
6. 【参考】好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。

反例：

```
// put elephant into fridge
put(elephant, fridge);
```

方法名 put，加上两个有意义的变量名 elephant 和 fridge，已经说明了这是在干什么，语义清晰的代码不需要额外的注释。

1.2 前端开发规范

1.2.1 规范目的

- 提高团队协作效率，实现代码一致性
- 通过代码风格的一致性，降低维护代码的成本以及改善多人协作的效率
- 方便新进的成员快速上手
- 输出高质量的代码
- 同时遵守最佳实践，确保页面性能得到最佳优化和高效的代码

本规范文档一经确认，前端开发人员必须按本文档规范进行前台页面开发。本文档如有不对或者不合适的地方请及时提出，经讨论决定后可以更新此文档。

1.2.2 基本原则

1.2.2.1 结构、样式、行为分离

尽量确保文档和模板只包含 HTML 结构，样式都放到样式表里，行为都放到脚本里。

1.2.2.2 缩进

统一 4 个空格缩进或者 Tab 缩进，不要使用 Tab 和空格混搭。

1.2.2.3 命名

变量，使用 Camel 命名法。

```
var loadingModules = {};
```

私有属性、变量和方法以下划线 _ 开头。

```
var _privateMethod = {};
```

常量，使用全部字母大写，单词间下划线分隔的命名方式。

```
var HTML_ENTITY = {};
```

- 函数，使用 Camel 命名法。
- 函数的参数，使用 Camel 命名法。

```
function stringFormat(source) {}
function hear(theBells) {}
```

- 类，使用 Pascal 命名法
- 类的 方法 / 属性，使用 Camel 命名法

```
function TextNode(value, engine) {
    this.value = value;
    this.engine = engine;
}
TextNode.prototype.clone = function () {
    return this;
};
```

- 枚举变量 使用 Pascal 命名法。
- 枚举的属性， 使用全部字母大写，单词间下划线分隔的命名方式。

```
var TargetState = {
    READING: 1,
    READED: 2,
    APPLIED: 3,
    READY: 4
};
```

由多个单词组成的 缩写词，在命名中，根据当前命名法和出现的位置，所有字母的大小写与首字母的大小写保持一致。

```
function XMLParser() {}
function insertHTML(element, html) {}
var httpRequest = new HTTPRequest();
```

1.2.2.4 命名语法

类名，使用名词。

```
function Engine(options) {}
```

函数名，使用动宾短语。

```
function getStyle(element) {}
```

boolean 类型的变量使用 is 或 has 开头。

```
var isReady = false;
var hasMoreCommands = false;
```

Promise 对象用动宾短语的进行时表达。

```
var loadingData = ajax.get('url');

loadingData.then(callback);
```

1.2.2.5 接口命名规范

- 可读性强，见名晓义；
- 尽量不与 jQuery 社区已有的习惯冲突；
- 尽量写全。不用缩写，除非是下面列表中约定的；（变量以表达清楚为目标，uglify 会完成压缩体积工作）

常用词	说明
options	表示选项，与 jQuery 社区保持一致，不要用 config, opts 等
active	表示当前，不要用 current 等
index	表示索引，不要用 idx 等
trigger	触点元素
triggerType	触发类型、方式
context	表示传入的 this 对象
object	推荐写全，不推荐简写为 o, obj 等

常用词	说明
element	推荐写全，不推荐简写为 el, elem 等
length	不要写成 len, l
prev	previous 的缩写
next	next 下一个
constructor	不能写成 ctor
easing	示动画平滑函数
min	minimize 的缩写
max	maximize 的缩写
DOM	不要写成 dom, Dom
.hbs	使用 hbs 后缀表示模版
btn	button 的缩写
link	超链接
title	主要文本
img	图片路径（img 标签 src 属性）
dataset	html5 data-xxx 数据接口
theme	主题
className	类名
classNameSpace	class 命名空间

1.2.2.6 文件编码

使用不带 BOM 的 UTF-8 编码；

在 HTML 中指定编码 <meta charset="utf-8"> 。

1.2.2.7 一律使用小写字母

```
<!-- Recommended -->
<img src="google.png" alt="Google">
<!-- Not recommended -->
<A HREF="/">Home</A>
```

1.2.2.8 统一注释

- HTML 注释
  - 模块注释

```
<!-- 文章列表列表模块 -->
<div class="article-list">
...
</div>
```

  - 区块注释

```
<!--
@name: Drop Down Menu
@description: Style of top bar drop down menu.
@author: Ashu(Aaaaaashu@gmail.com)
-->
```

- CSS 注释
  - 组件块和子组件块以及声明块之间使用一空行分隔，子组件块之间三空行分隔

```
/*
================================================================================
   组件块

================================================================================ */
/* 子组件块

================================================================================
*/.selector {
  padding: 15px;
  margin-bottom: 15px;}


/* 子组件块

================================================================================
*/.selector-secondary {
  display: block; /* 注释*/
}
.selector-three {
  display: span;
}
```

- JavaScript 注释
  - 单行注释

    必须独占一行。

    // 后跟一个空格，缩进与下一行被注释说明的代码一致。

  - 多行注释

    多行注释以/*开始，以*/结束。

    多行注释总是出现在将要出现的代码段之前，注释与代码之间没有空行间隔。

    多行注释之前应当有一个空行，且缩缩进层级与器描述的代码保持一致。

  - 文件注释用于告诉不熟悉这段代码的读者这个文件中包含哪些东西。 应该提供文件的大体内容，它的作者，依赖关系和兼容性信息。如下:

```
/**
 * @fileoverview Description of file, its uses and information
 * about its dependencies.
 * @author user@meizu.com (Firstname Lastname)
 * Copyright 2015 Meizu Inc. All Rights Reserved.
 */
```

1.2.2.9 代码验证

- 使用 W3C HTML Validator 来验证你的 HTML 代码有效性
- 使用 W3C CSS Validator 来验证你的 CSS 代码有效性

1.3 前后端交互规范

1. 【强制】前后端开发前必须先定义 schema
2. 【强制】API 不能干涉产品，只提供数据
3. 【强制】前后端边界处理
4. 【强制】开发环境下，后端开发人员必须保证 API 本地测试通过，统一使用 API 测试工具：Postman
5. 【强制】开发环境下，后端开发人员须保证使用 Swagger 生成 API 文档对接口描述清晰明确