Git、Maven的使用规范

爱旅行项目整个开发过程使用Git进行项目代码的版本管理，并使用Maven进行项目构建及多模块管理。

## Git使用说明

### 1.1.1 Git环境准备

爱旅行项目在开源中国进行代码托管https://git.oschina.net/（团队成员须自行注册账号），接下来由Team Leader完成Git环境的相关准备工作。

1. 创建项目，如图1、2所示，注意：由于该项目采用前后端分离开发，故应创建相应的前端项目（itripfront）和后端项目（itripbackend），分开进行代码版本管理。



图1

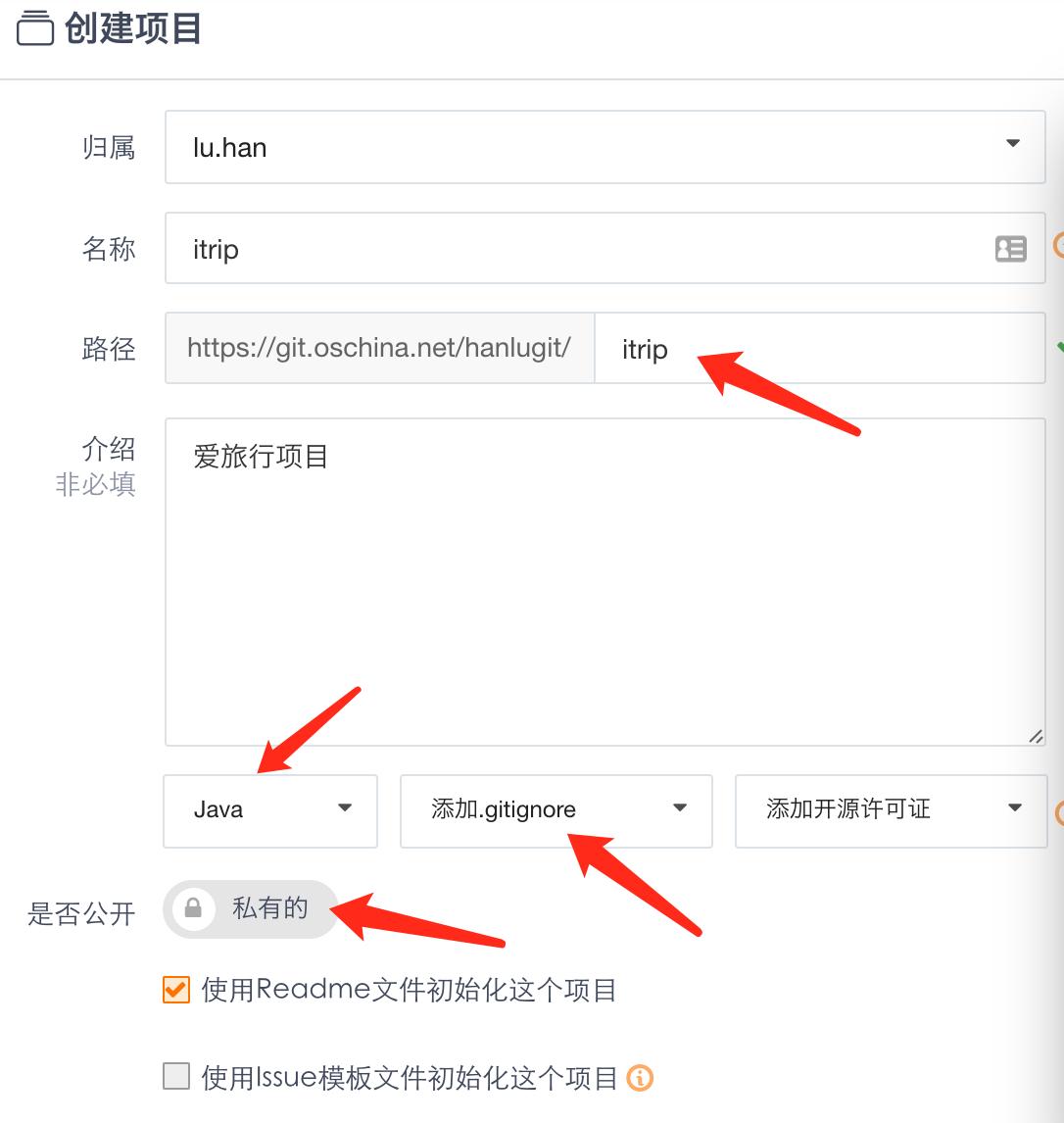


图2

1. 创建开发小组，如图3所示，把team成员拉进项目中，并赋予相应权限，一般开发人员授予开发者权限。

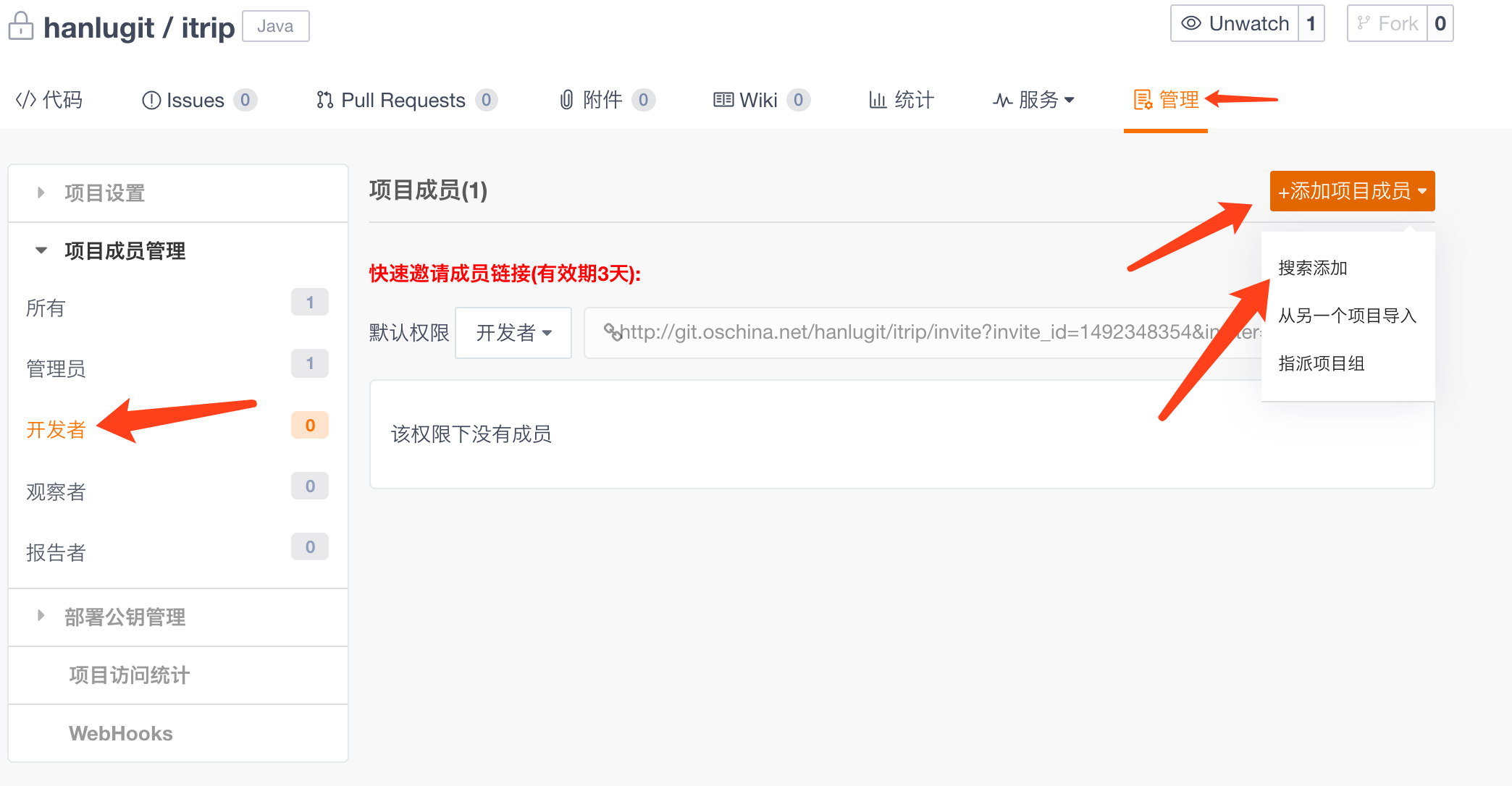


图3

1. clone 项目 【前提：客户端已成功安装git client并保证可用】

首先复制master上项目路径到内存中，如图4所示：

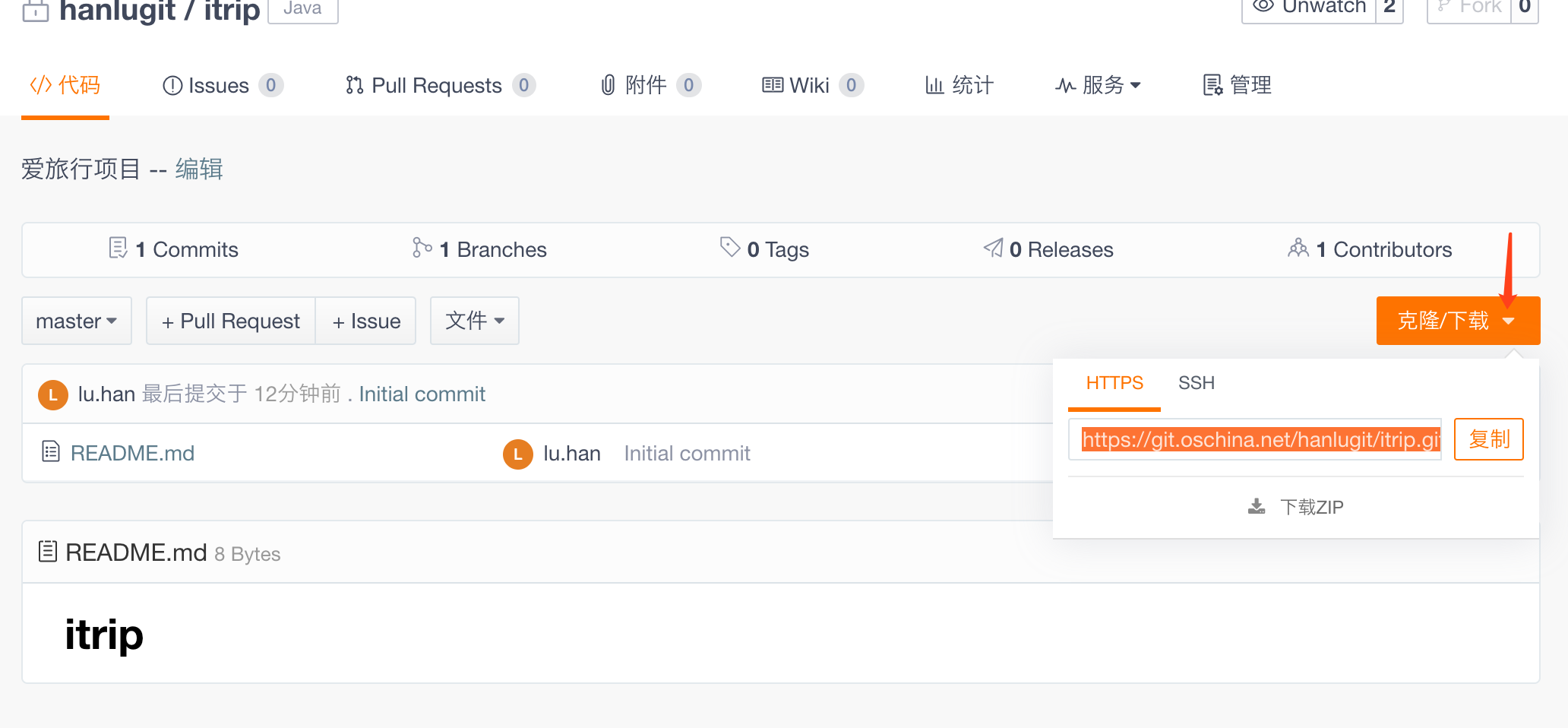


图4

选择本地目录（e.g: D:\ideaworkspace），即：本地工程所在目录，然后进入Git Bash，输入命令：

git clone https://git.oschina.net/hanlugit/itrip.git

clone完成之后，可查看到本地已创建itrip项目，并且项目中只有一个README.md文件（注：项目开发者可以使用该文件进行项目记录等），通过命令：git branch查看所在分支为master

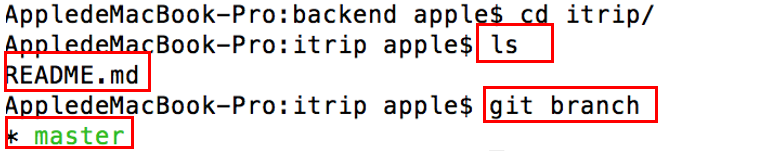
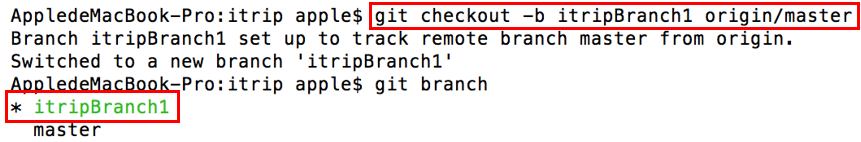


图5

在本地完成项目框架搭建后，通过命令推送到远程主干（master）上。命令如下：

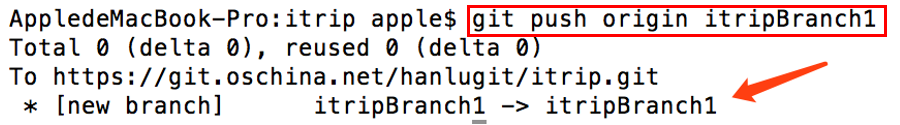
1. git add .
2. git commit -m ‘add pro framework’
3. git push
4. 创建分支itripBranch1，并push到远程服务器

1> checkout命令：git checkout -b itripBranch1 origin/master



2> push本地分支itripBranch1到远程服务器，完成远程分支的创建。

命令：git push origin itripBranch1



注：分支创建除了通过命令之外，也可在页面上进行创建。

1. 完成项目框架的搭建，并上传项目代码到分支itripBranch1上
2. 分支合并（注意：此操作由Team Leader完成）

一般情况下，当一个项目大版本完成，或者阶段完工，或者完整功能点开发完毕，需要部署到生产环境中，则会将开发中的分支（itripBranch1）合并到主干（master）上，注意：需合并的分支代码一定保证运行正常无报错，即：稳定运行版本。具体合并步骤如下：  
     1> 首先切换的master分支：git checkout master

 2> 然后执行合并操作：git merge itripBranch1

 3> 若有冲突，会提示，可调用git status查看冲突文件

 3> 解决冲突，并调用git add或git rm将解决后的文件暂存

 4> 所有冲突解决后，git commit 提交更改，git push

### 1.1.2 操作Git

开发者操作Git，可使用【推荐】Git Bash等客户端工具或者使用IDE（e.g：IntelliJ IDEA）集成工具进行代码的推送和拉取。具体操作过程如下：

1. git clone : 从master clone，是必须的，如图5：

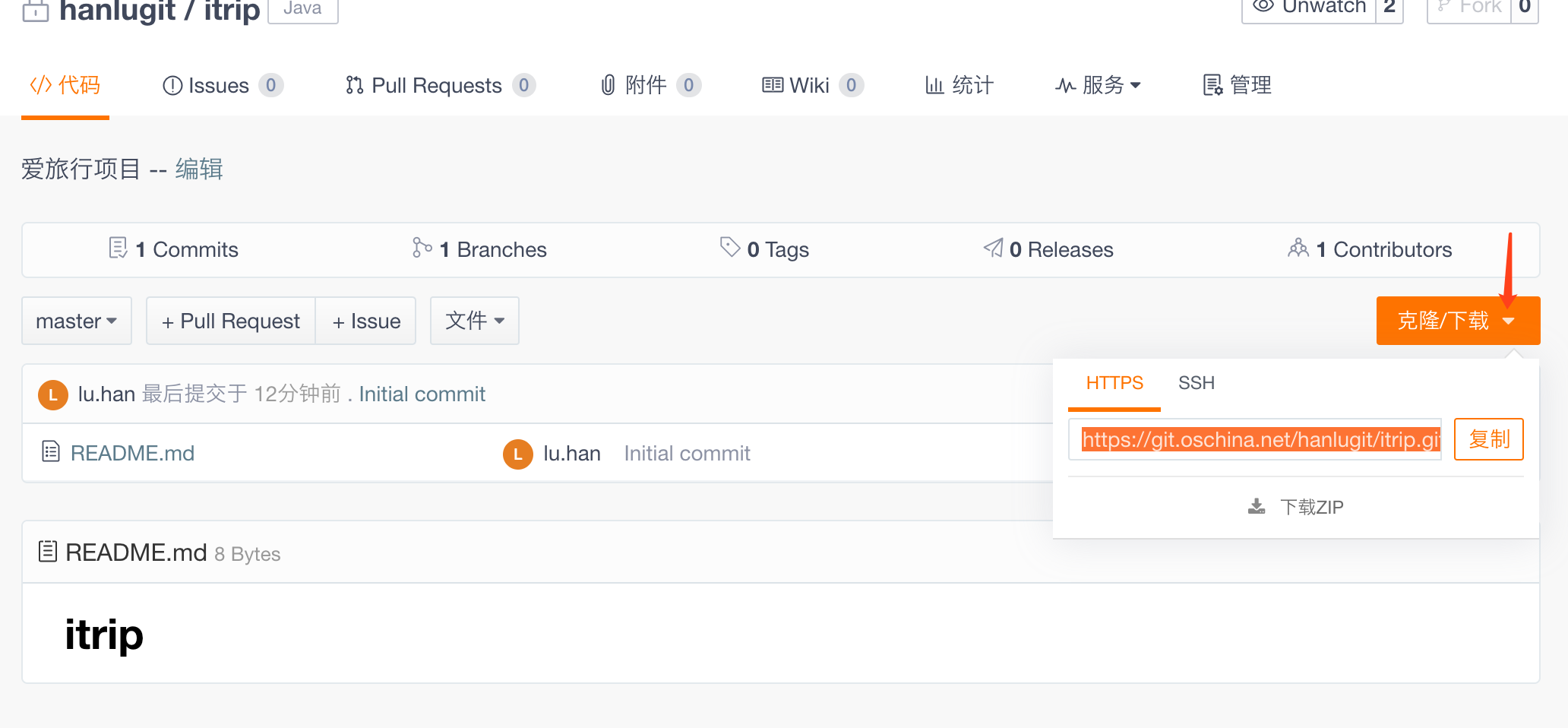


图5

1. 命令：git clone <https://git.oschina.net/hanlugit/itrip.git>
2. 命令：ls 查看一下，会发现本地目录中已经有了itrip
3. 命令：git branch 查看当前所在分支（目前应该是在master上）
4. checkout : 从项目管理者（Team Leader）处得知目前开发的分支，通过checkout 远程分支进行分支切换，即可在开发分支上工作。

命令：git checkout -b  itripBranch01 remotes/origin/itripBranch01

通过git branch来查看当前所在分支（目前应该是在itripBranch01 上），以后就在该分支上进行代码的开发，push即可。

注意：开发者只在分支上进行开发，不操作master！

### 1.1.3 Git常用命令

1. git clone
2. git branch （注：作为初学者，应每次push之前，查看当前所在分支是否正确）
3. git status（注：每次push之前，要先执行该命令，确保提交正确）
4. git add .
5. git commit -m ‘填写commit信息描述’（注：添加commit信息必须完整正确）
6. git push
7. git checkout 分支名
8. git pull （注：每次push之前，要先pull）

### 1.1.3 Git



### 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：

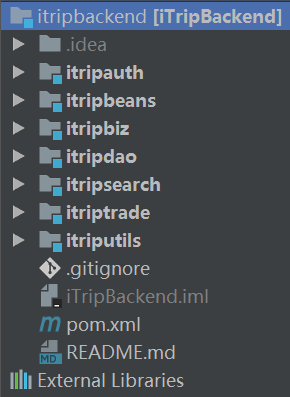
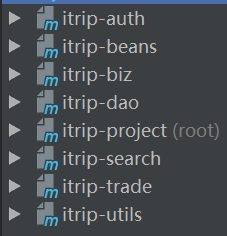
 

图1 iTripBackend-模块结构 图2 artifactId

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

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

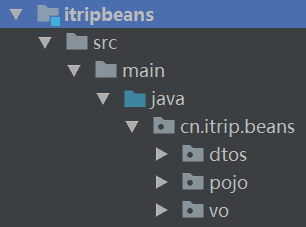


图3 itripbeans

1. 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

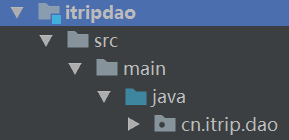


图4 itripdao

1. 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

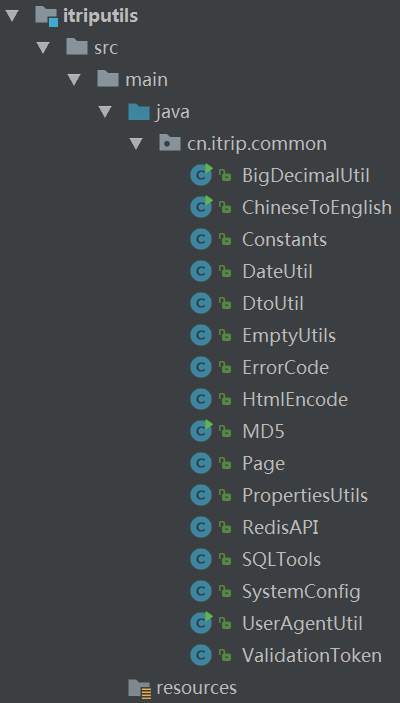


图5 itriputils

1. 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

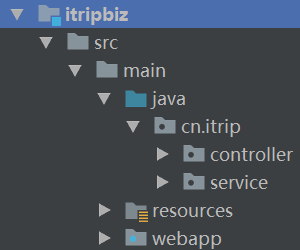


图6 itripbiz

1. 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

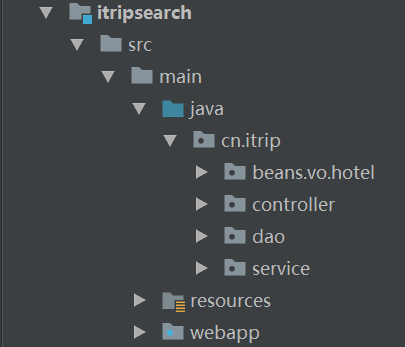


图7 itripsearch

1. 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

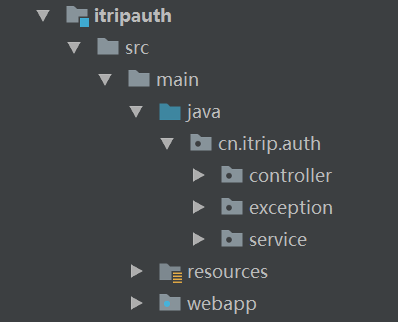


图8 itripauth

1. 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

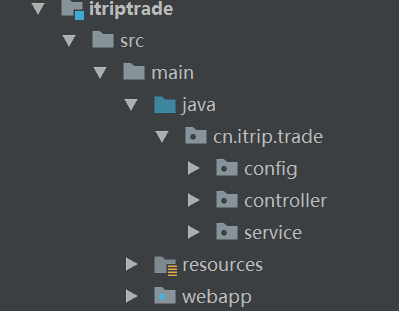


图9 itriptrade

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

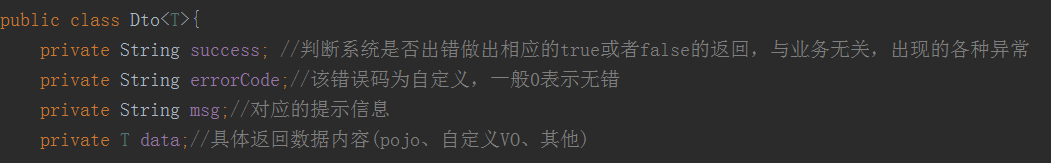


图10 dto

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

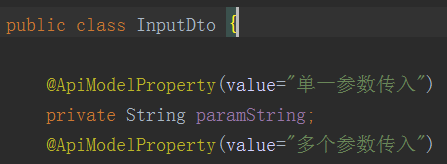


图11 Inputdto

1. 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：

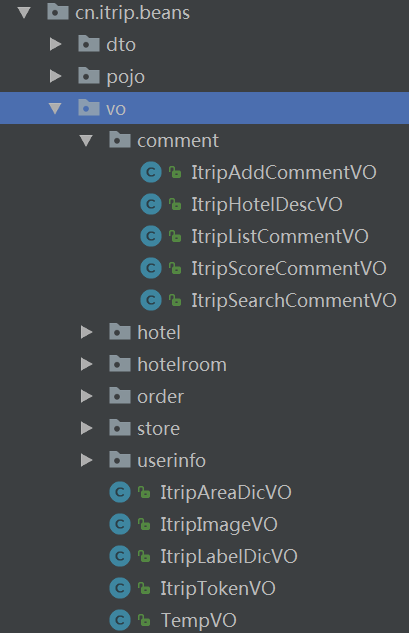


图12 vo

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。

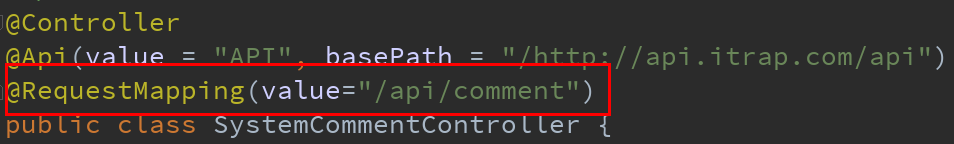
2> 若业务需进行事务处理，则需在接口方法名加上前缀itriptx，比如：

public Integer itriptxModifyItripHotelOrder(ItripHotelOrder itripHotelOrder)throws Exception;

1. 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：



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 |  |
| 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 |

### 1.1.7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。

1. 【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。

1. 【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）。

1. 【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。
2. 【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 Maven使用说明

1、【强制】前端的健壮性开发，需要达到：在无API的情况下，前端一样能够完美奔跑

2、【推荐】前端采用mock开发