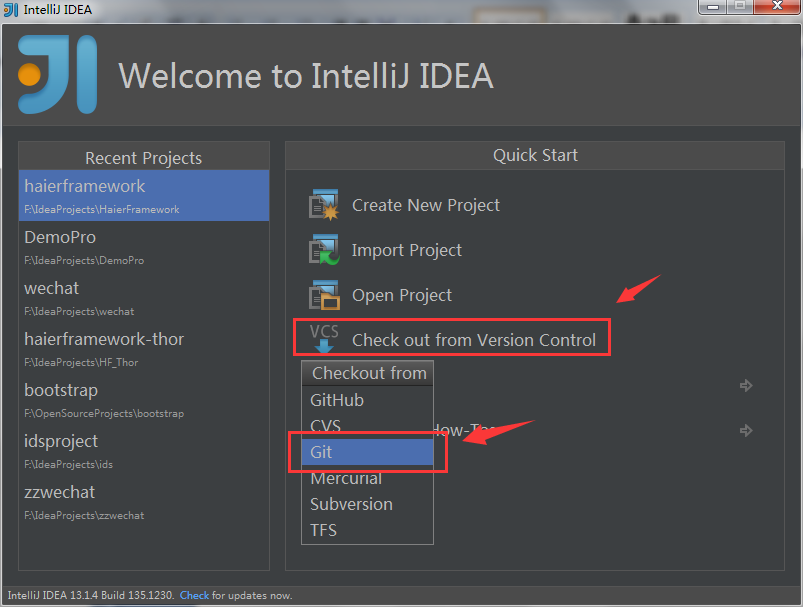
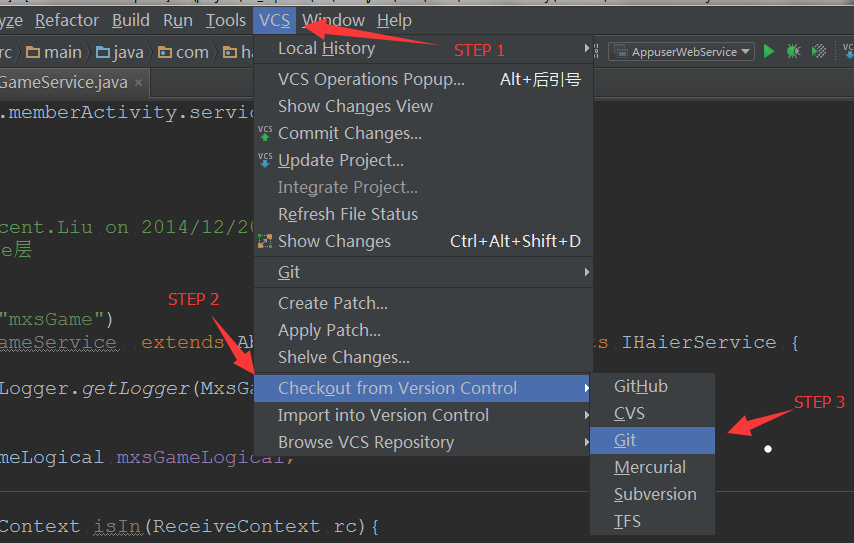
# 二、IDEA的使用

## 1.检出项目

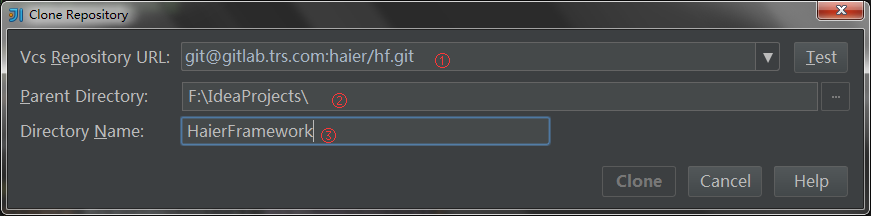
* 入口一，刚打开IDEA窗口时



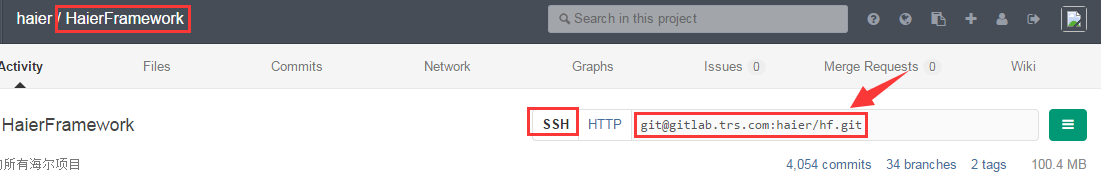
* 入口二，某一打开项目中

****

### 2）设置检出目录

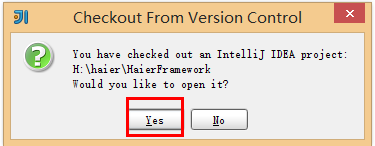


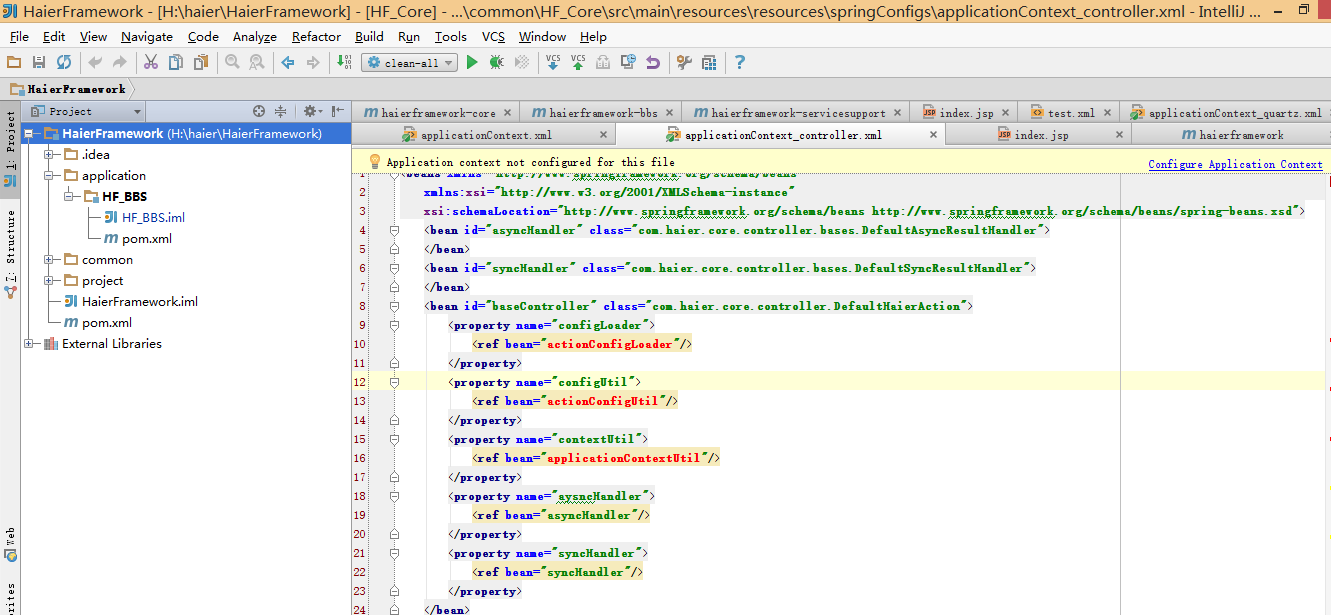
1. Git公共服务器地址，从gitlab上获取（注意选择Project和数据传输协议）



1. 存放项目的父目录
2. 项目目录名称（互动应用使用HaierFramework）

成功检出，直接打开项目

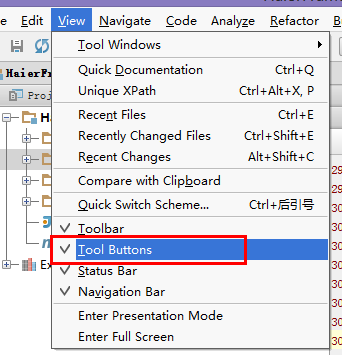




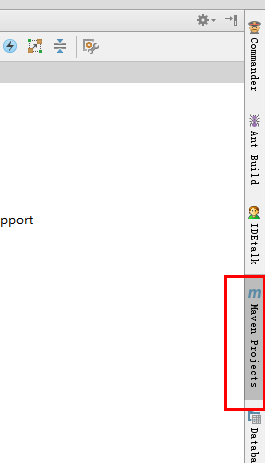
### 3）刷新项目

新检出的项目没有依赖，需要等待一段时间下载依赖jar

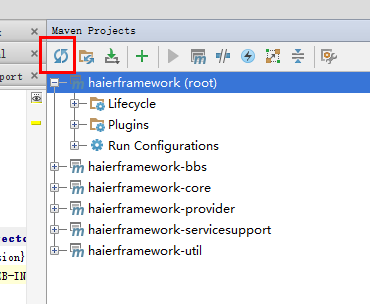
设置显示工具按钮



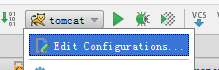
点击右侧的maven projects



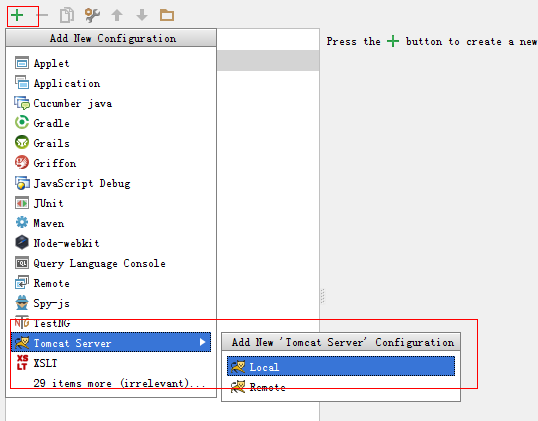
点击刷新



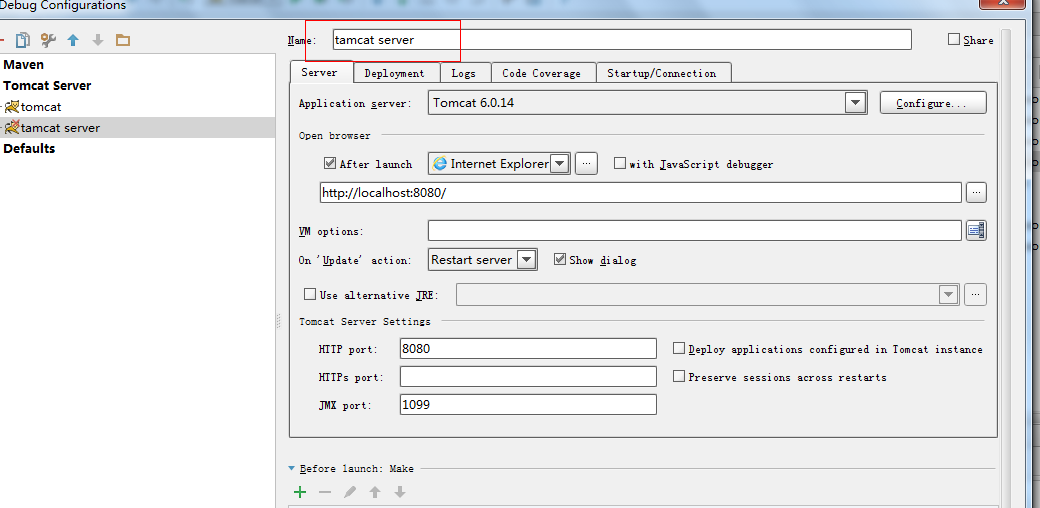
## 2.tomcat插件配置

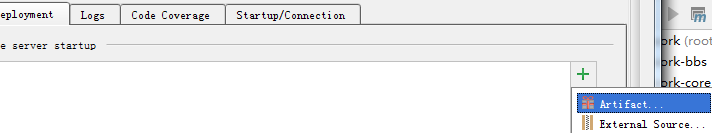
选择菜单中的**Run**，在弹出的菜单中选择**run-**>**Edit Configurations**，或工具栏的

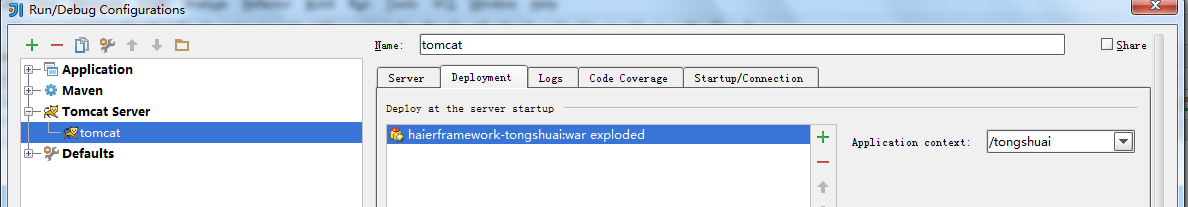
弹出设置窗口, 点击 **+** 新建配置，选择 **Tomcat Server** - **Local**



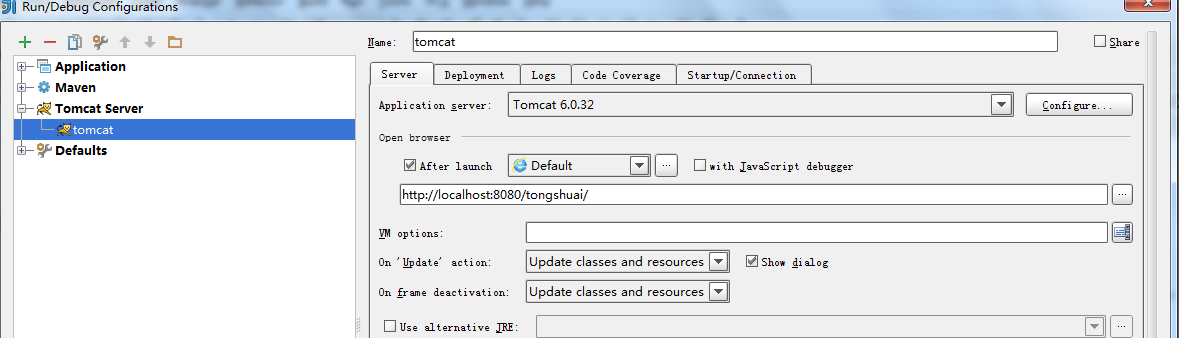
输入**Name**，点击"deployment"选项卡,选择要部署的项目,输入 application content,点击OK保存tomcat 配置信息





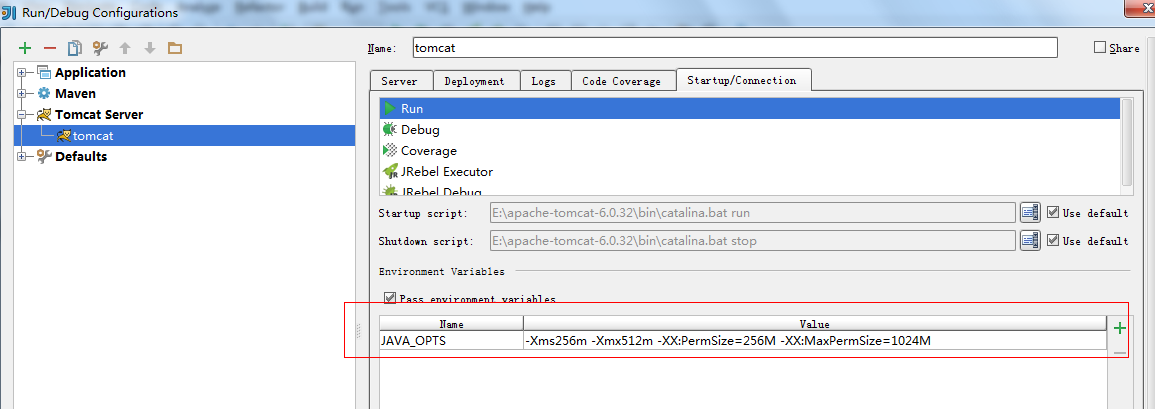


保存后的tomcat ，点击工具栏的配置信息，，On frame deactivation 下拉列表中值改为 Update classes and resourcees既可以在debug模式下自动部署修改的文件

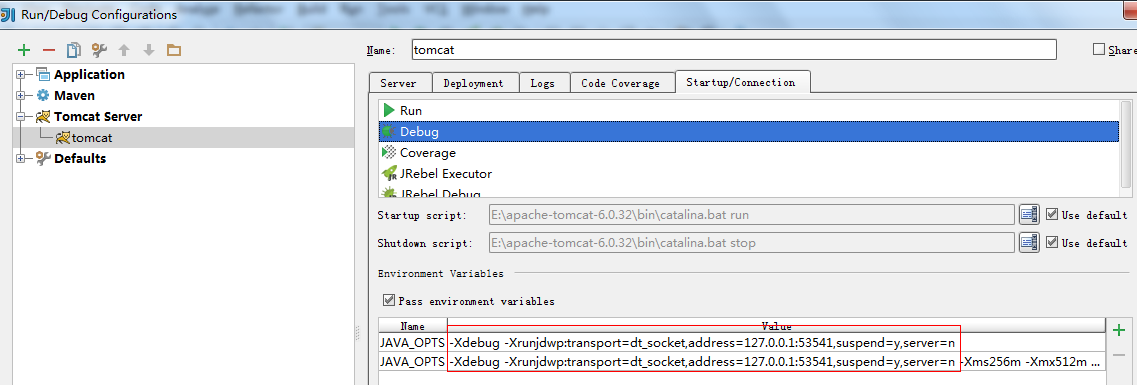


**tomcat 内存配置:**

点击工具栏的tomcat配置信息,选择Startup/Connection选项卡,选择Run选项,添加JAVA\_OPTS参数信息(-Xms256m -Xmx512m -XX:PermSize=256M -XX:MaxPermSize=1024M)



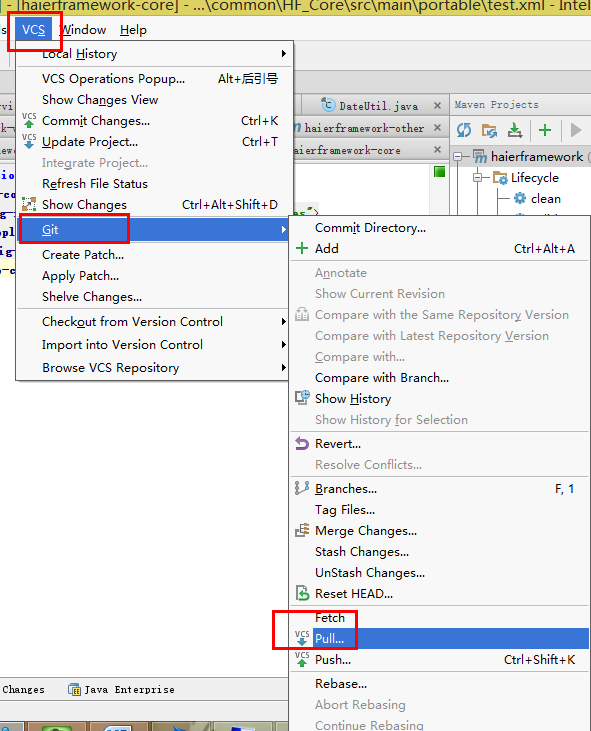
选择Debug 添加配置信息(须将debug下原JAVA\_OPTS的值copy到自己新建的JAVA\_OPTS中后再添加jvm内存配置信息)

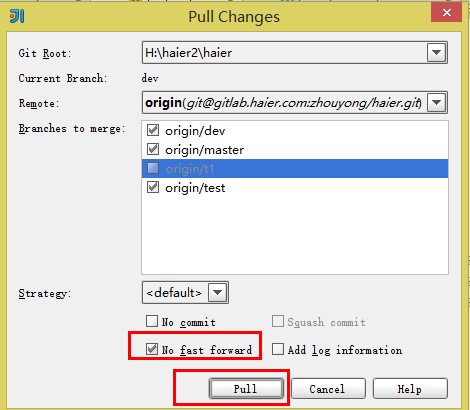


也可以配置jrebal插件来实现热部署

## 3.代码的更新

每次工作之前和commit之后，都要pull一下。





记得勾选 No fast forward

通常，合并分支时，如果可能，Git会用“Fast forward”模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用“Fast forward”模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

## 3.代码的提交

### 1）新建分支

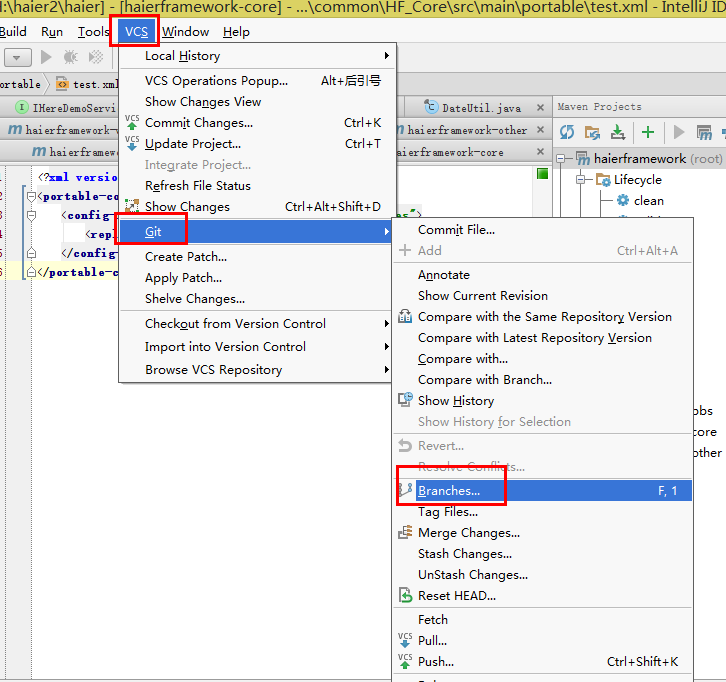
每次开发某个功能之前或修改bug都必须先建立本地分支，都在dev分支上新建分支，代码完成后都合并到dev分支，然后删除新建的分支。

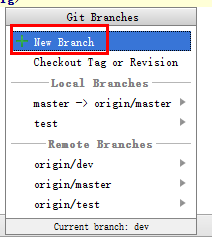
分支的命名必须要规范，方便查看，规则：

分支已自己的名字开头

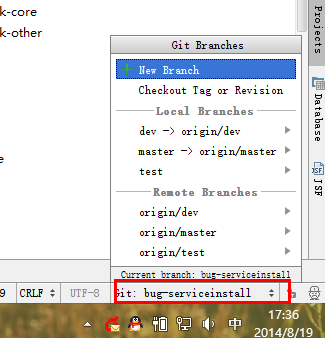
Bug类的，都是bug-开头的，例如修复在线报修的bug，可以命名为zhouyong-bug-serviceinstall

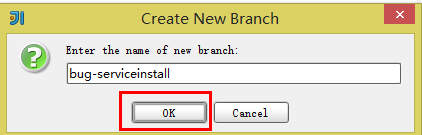
功能类，都是feature-开头，报修功能，zhouyong-feature-serviceinstall





也可以点击底部状态的图标出来Git Branches

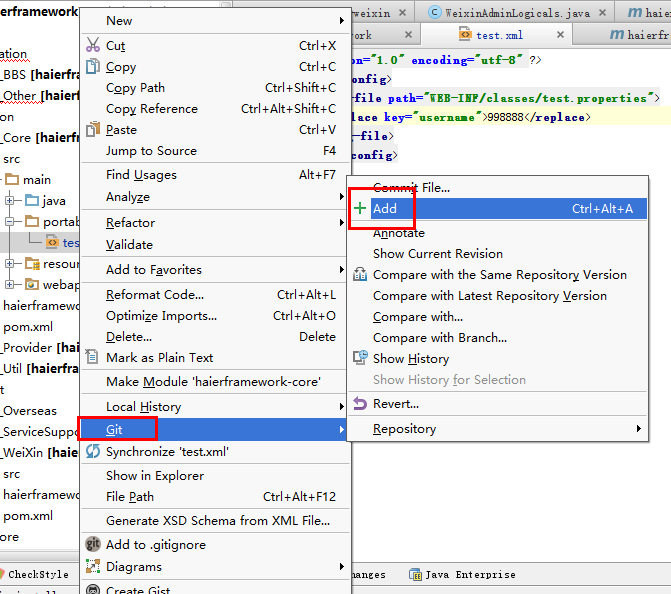




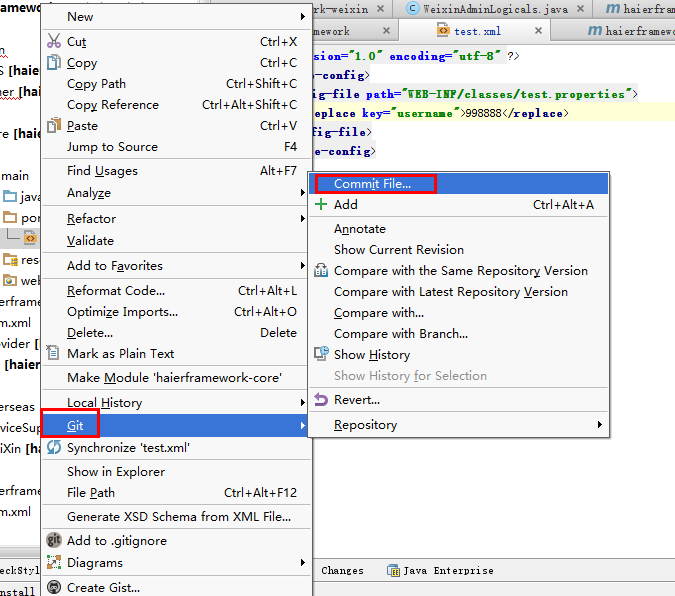
新建完成分支后，ide会自动切换到新建的分支上。

### 2）提交代码

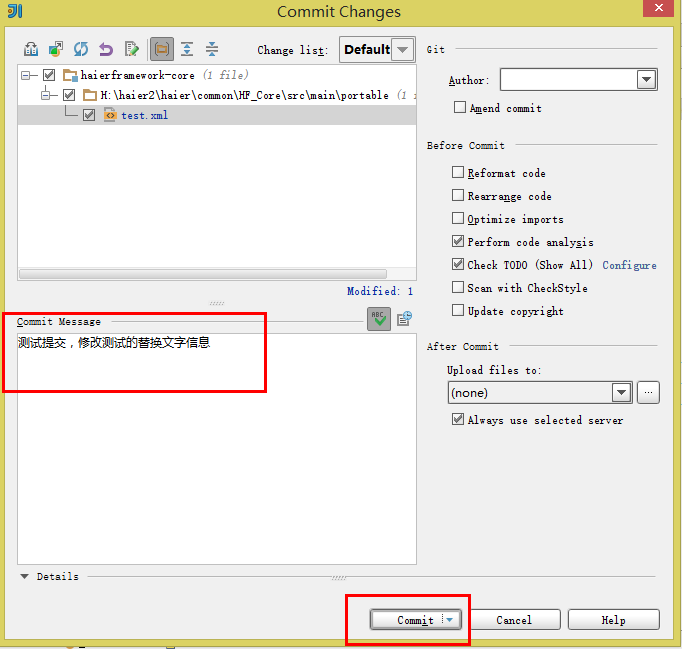
1.选中要提交的文件或文件夹，鼠标右键，Add



2.commit



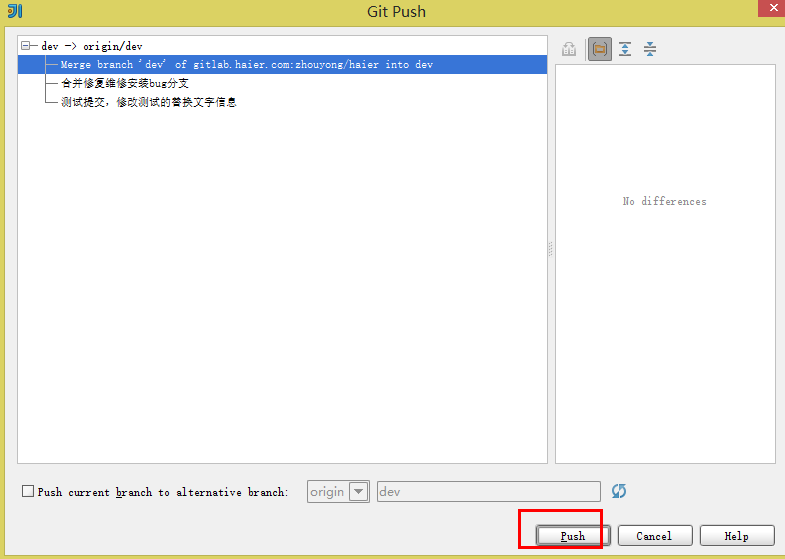
选择要提交的代码，提交信息必须填写，没有填的或乱填的抓到直接砍死。



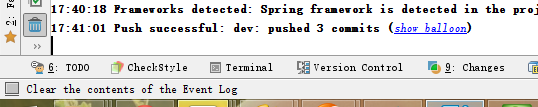
提交成功后，可以推送到远程服务器。

### 3）推送代码到远程服务器

如果要推送到远程服务器，则先pull一下，因为远程服务器的版本可能更新了。

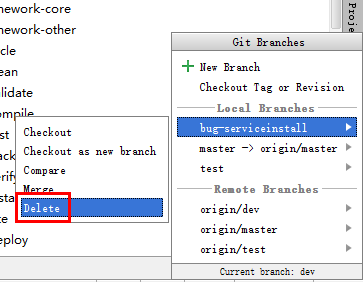


推送成功后可以看见成功的提示信息，失败的话一般都是没有pull或没有权限造成的。



### 4）删除分支

如果bug已经修复或功能已完成，则可以删除这个分支，当然你也可以选择不删除如果你不觉得乱的话。



### 5）合并远程分支

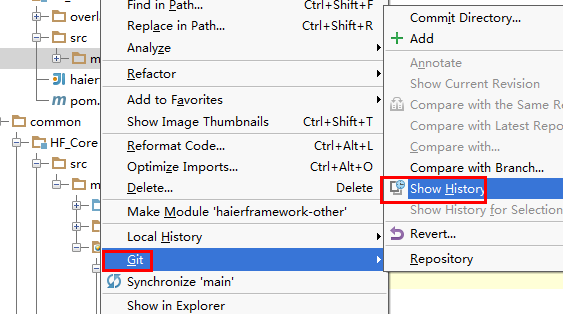
Push成功后，可以登录gitlab上查看到你提交的信息。



工程师都没有dev/master分支的权限，master分支的代码是和生产环境一致的，如果你提交的代码需要部署生产环境则在gitlab上，申请一个合并代码的请求，将你的代码从自己的分支上合并到dev上。测试通过后，由负责人将代码从dev合并到master，再部署生产环境。

## 4.查看文件的历史记录

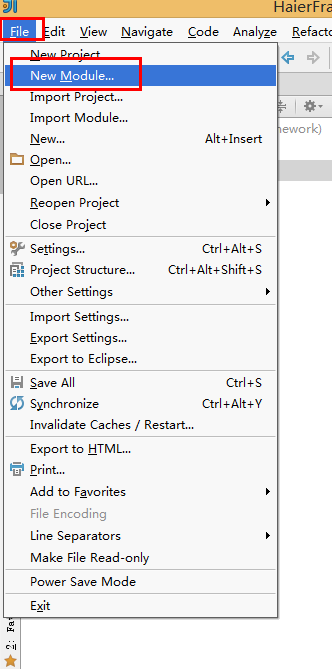
选中要查看的文件或文件夹，右键，Git-🡪show history



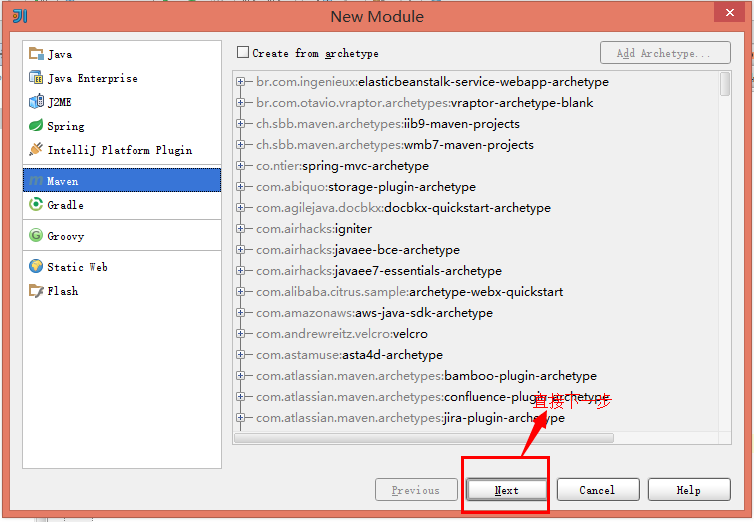
## 5.对比文件或目录

## 6.新建模块

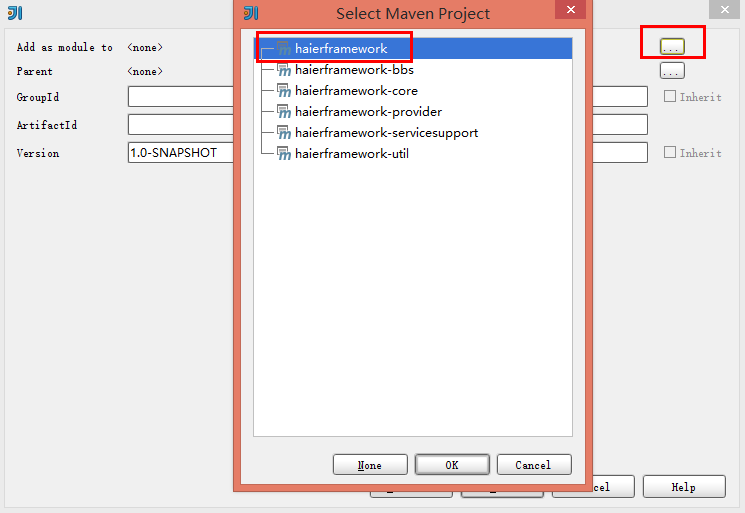
选中你要新建模块的文件夹，点击File---》New Module…



直接点击下一步



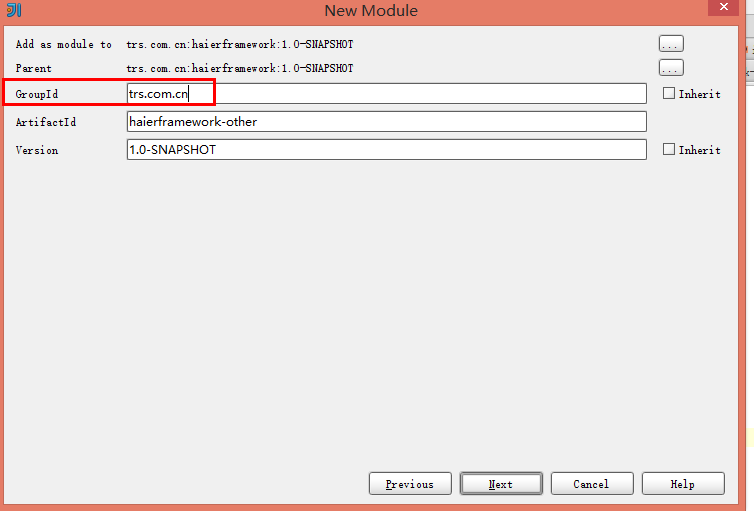
Add as module to 和Parent都选择根项目haierframework



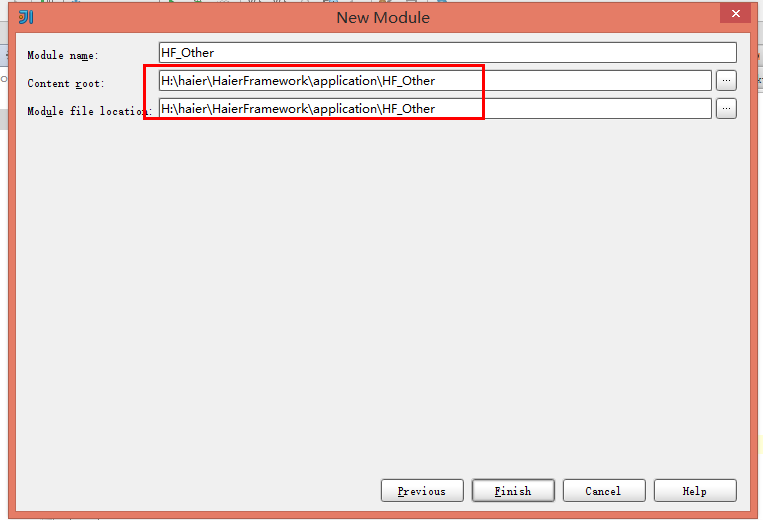
GroupId:必须为trs.com.cn，只是规范，不是技术上的必须填这个

ArtifactId:这个是在maven中的标识，用来定位你这个模块的，规定为haierframework为前缀，后面跟的是自己功能模块的名称

Version:版本号，一般默认就行，如果升级可调整。



注意选择路径，模块名称必须是HF开头



## 7.开发流程规范（新）

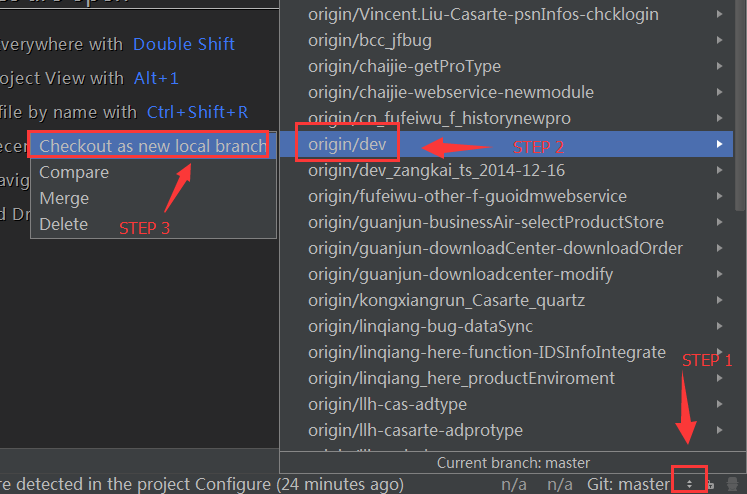
### 7.1项目检出（见 二-1）

### 7.2 开发中Git使用流程规范

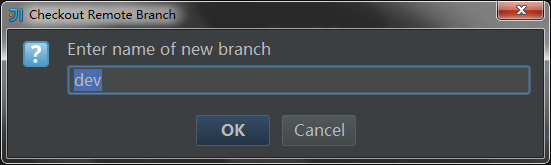
#### 7.2.1检出dev和test分支到本地

图示为dev分支检出到本地过程，test分支检出同理执行

* 检出

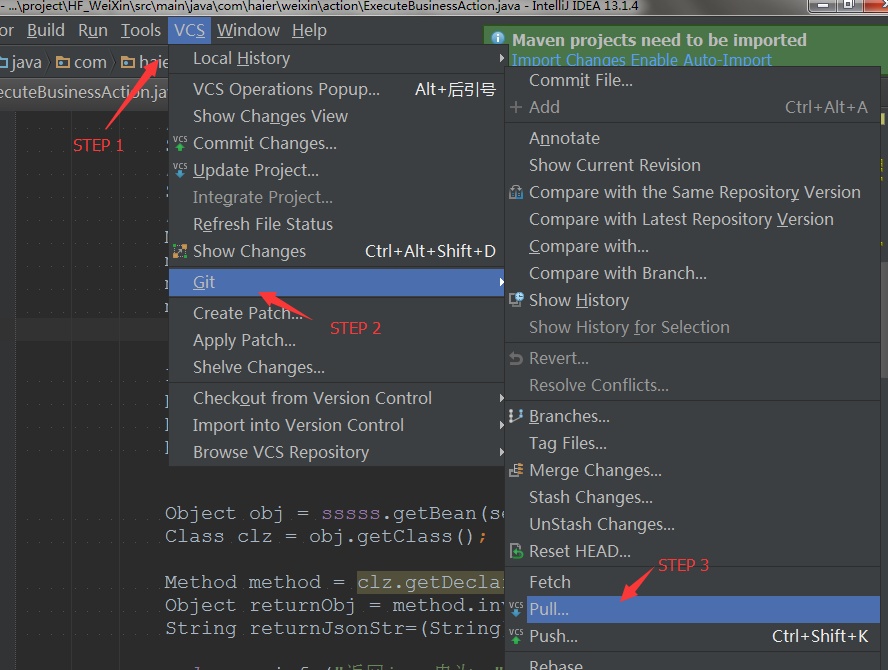


* 分支名称不变，点击ok检出分支到本地

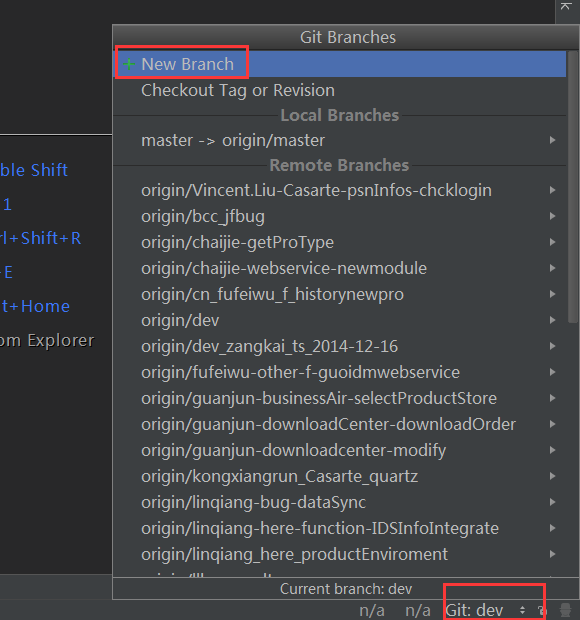


#### 7.2.2更新dev分支，并新建开发分支

* 切换到dev分支
* 更新dev
* 保证当前分支为dev，查看右下角分支名称
* 使用pull命令



* Pull在dev分支上，点击右下角dev分支名称，在弹出的菜单中选择New Branch



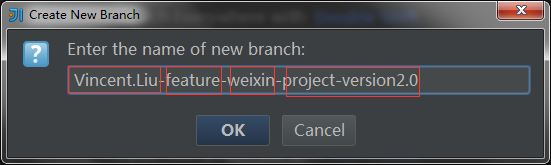
* 开发分支名称命名规则

AuthorName-feture/issue/hotfix-ProjectName-Description

feature：大规模性模块改版

issue：功能开发

hotfix：问题修复



### 7.3 代码提交本地仓库

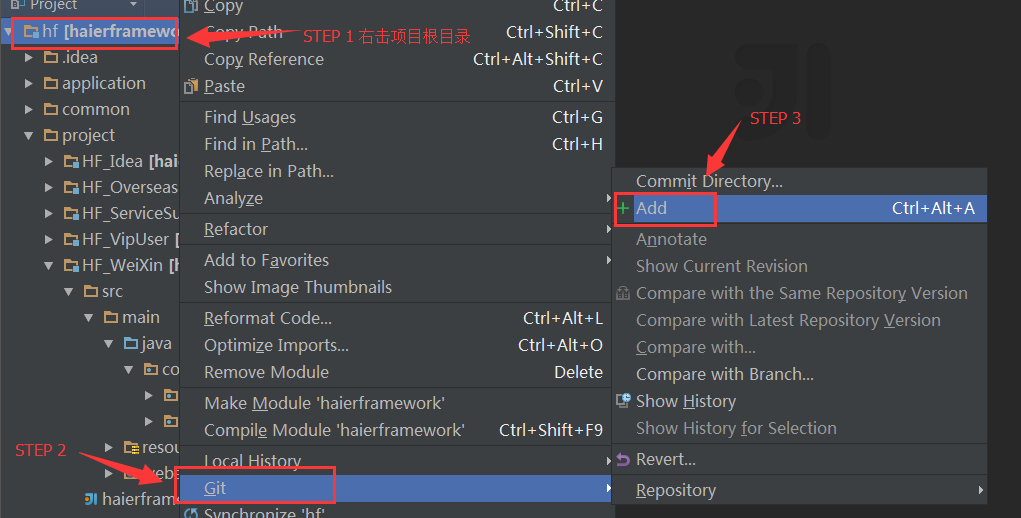
代码提交注意事项：

* 控制提交次数（减少commit数量），仅在当前分支在功能或者代码上具有可分割性时做一次提交操作（在需要切换分支但是不需要commit的情况下，可以使用stash功能，详见《GIT\_STASH\_v1.0.0.docx》）
* 当前分支应为自己的开发分支，查看右下角分支名称应为你的开发分支名称

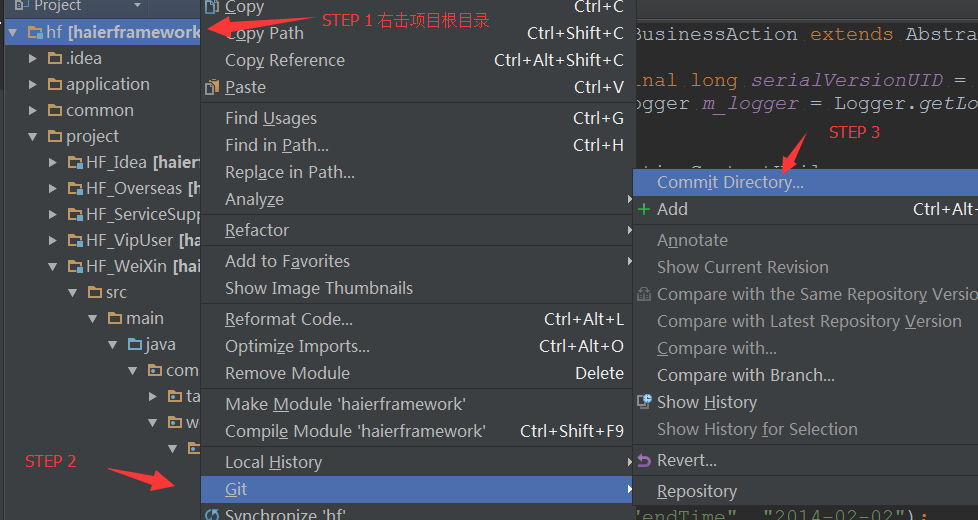
#### 7.3.1代码暂存

* 使新增的文件进入Git管理系统
* 使修改过的文件生成可被commit对象感知的快照对象

当add按钮不可点时，则无需进行此操作



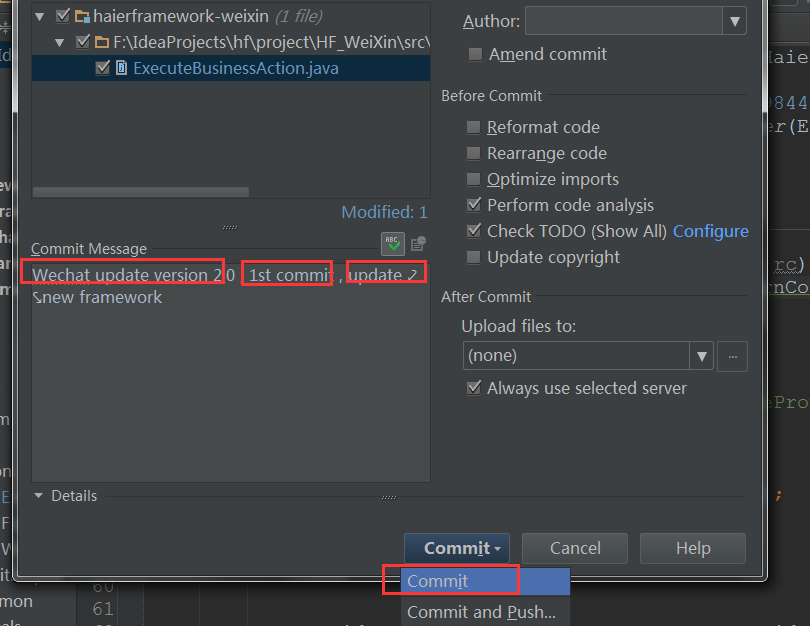
#### 7.3.2提交本地仓库



* 注释填写规范，以及完成commit

规范：

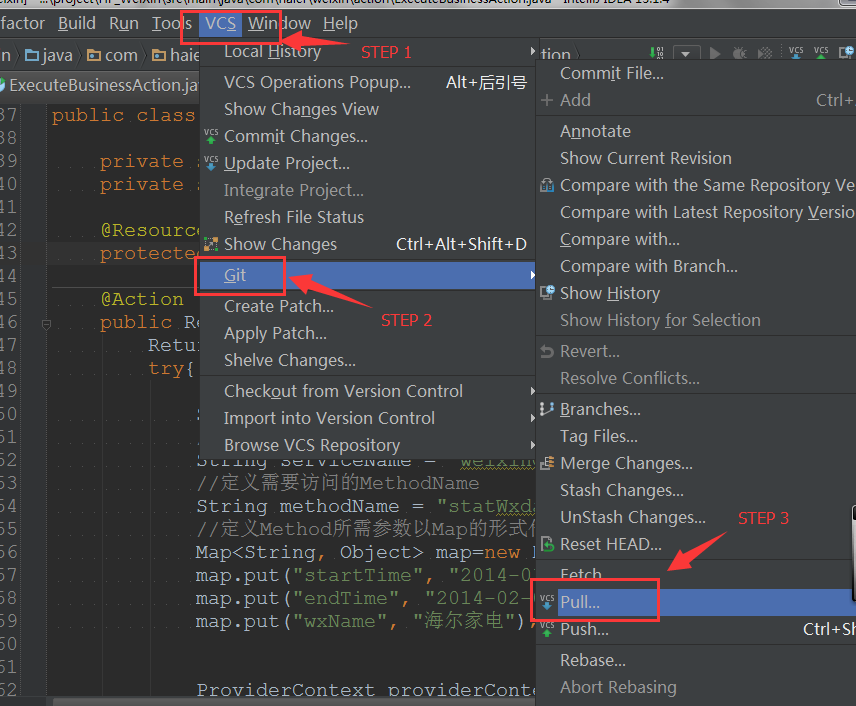
分支功能，第几次提交，本次提交内容



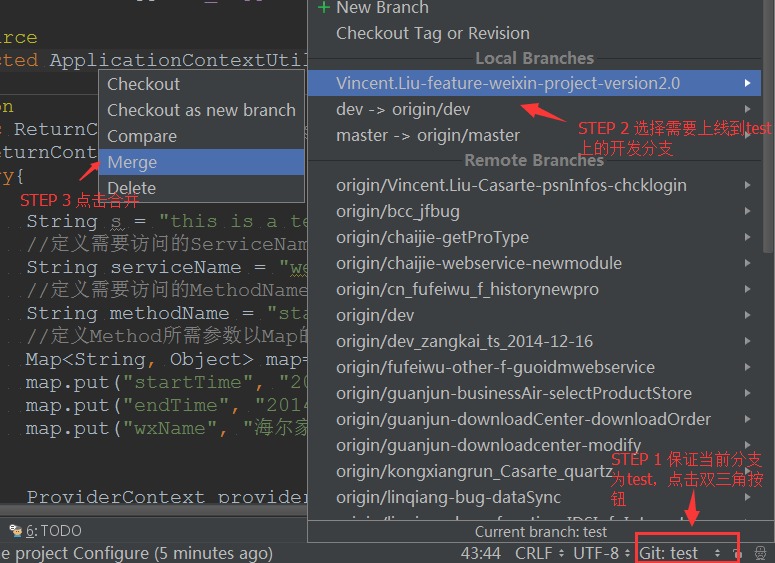
### 7.4 上线测试环境

#### 7.4.1 更新test分支

* 切换到test分支
* 更新test分支
  + 保证当前分支为test分支，查看右下角分支名称
  + 使用pull命令更新test分支



#### 7.4.2 合并开发分支代码到test



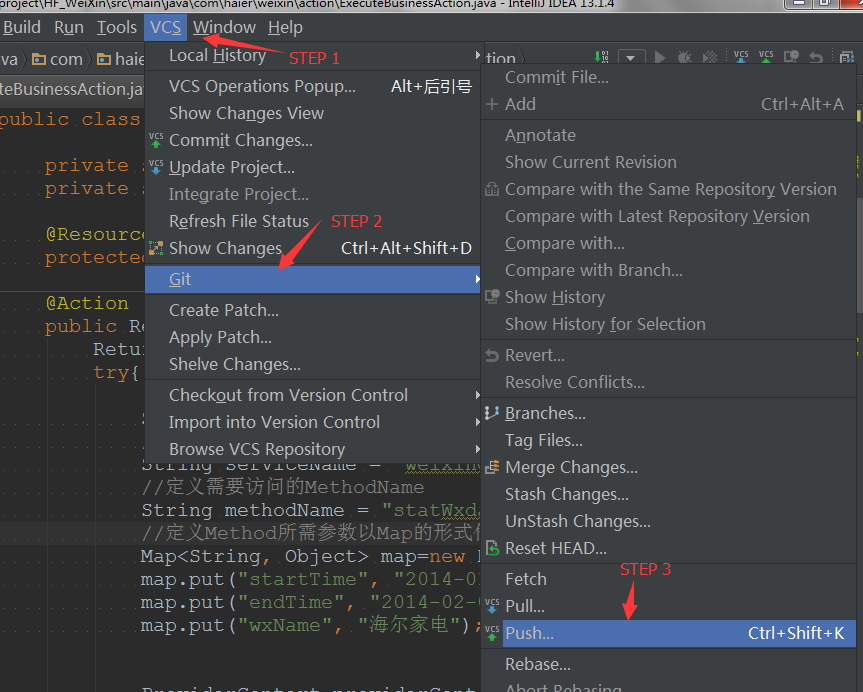
#### 7.4.3冲突解决

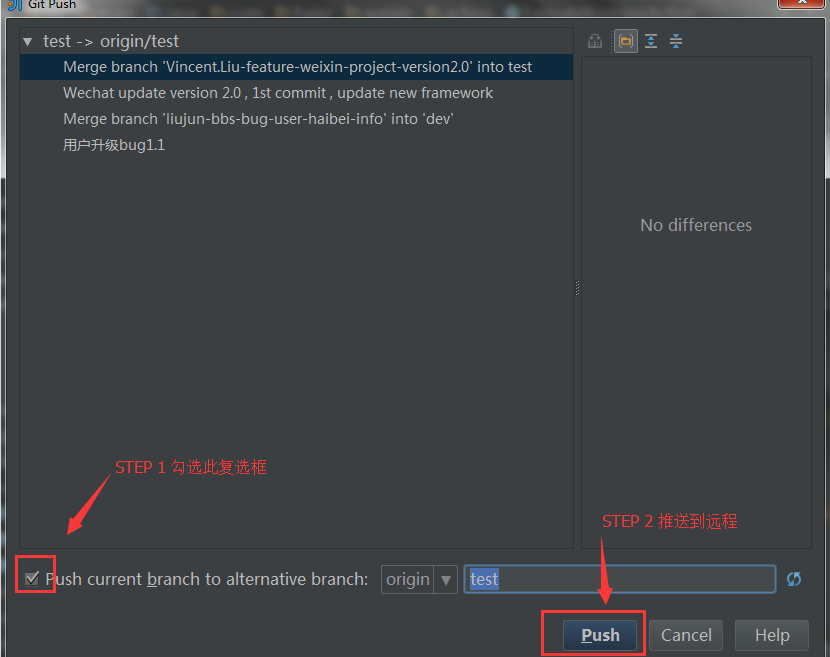
如果存在冲突的话，则需要进行冲突解决。

详细冲突解决合并方法见《GIT\_冲突解决\_v1.1.0.docx》

#### 7.4.4推送test到远程

* 保证当前分支为test
* 推送步骤





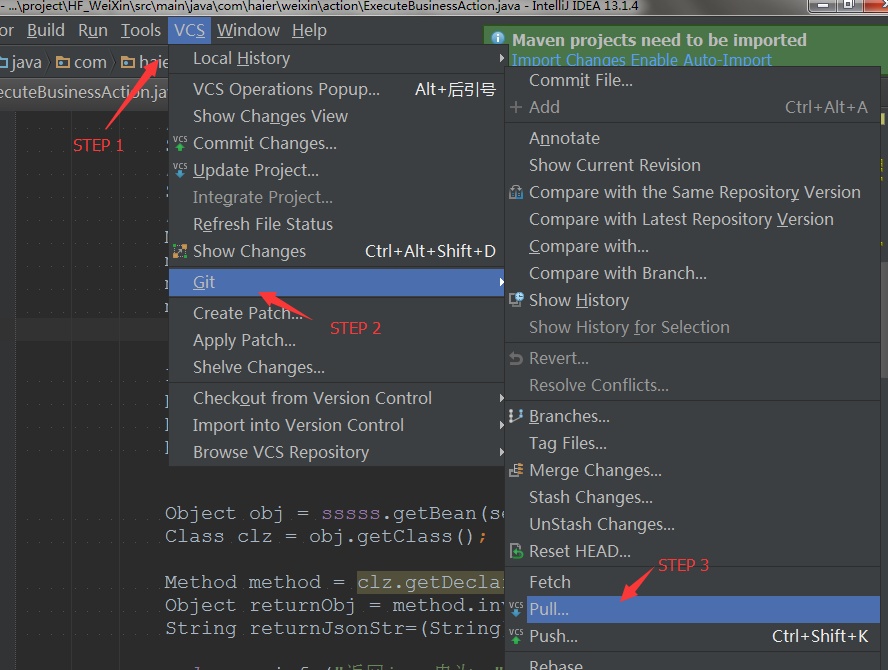
#### 7.4.5测试环境服务部署重启

详见Q&A中Jenkins重启测试环境应用

### 7.5 上线生成环境

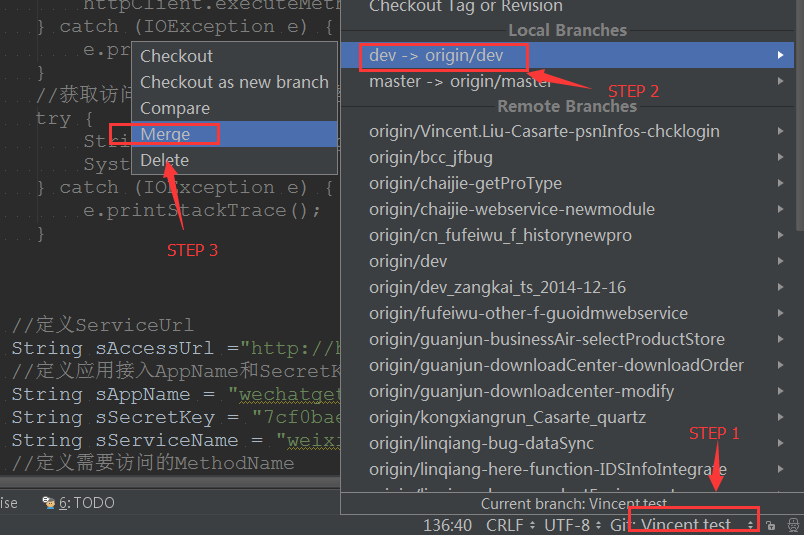
#### 7.5.1更新dev分支

* 切换到dev分支
* 更新dev
* 保证当前分支为dev，查看右下角分支名称
* 使用pull命令

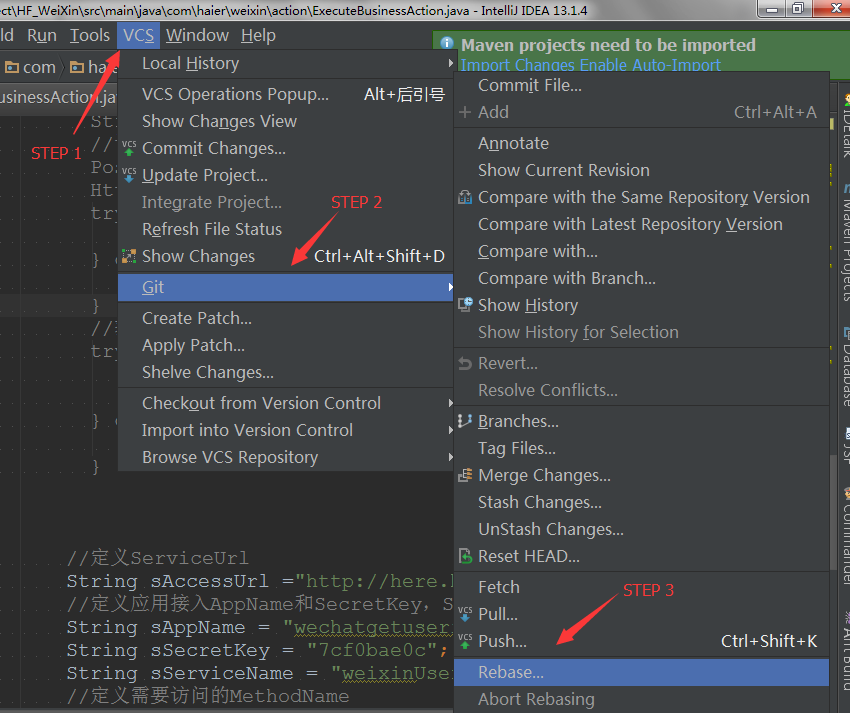


#### 7.5.2进行代码合并或衍合

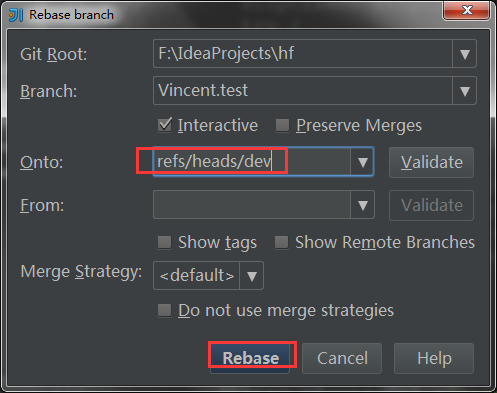
* 切换到自己分支
* 代码合并（merge）
* 保证当前分支为自己的开发分支
* 选用本地最新dev合并到自己的开发分支
* 原理见《git-bri-manual-v1.0.0.docx》



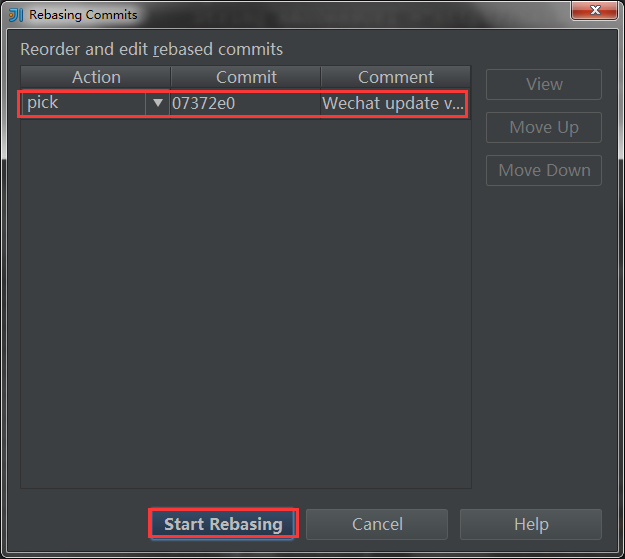
* 代码衍合（rebase）
* 如果已经进行代码合并（merge）则无需进行此步骤
* 保证当前分支为自己的开发分支
* 原理见《git-bri-manual-v1.0.0.docx》



* 选用最新的dev作为衍合基础分支（名称固定为refs/heads/dev）
* 点击rebase进行衍合前准备



* 列表显示当前分支所有的commit信息
* 点击start rebasing 开始衍合

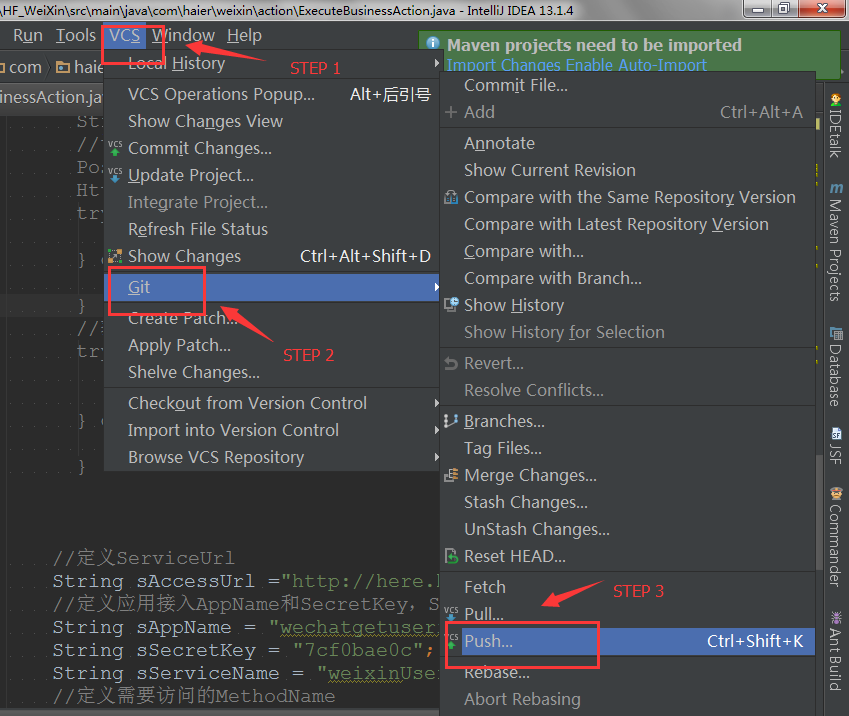


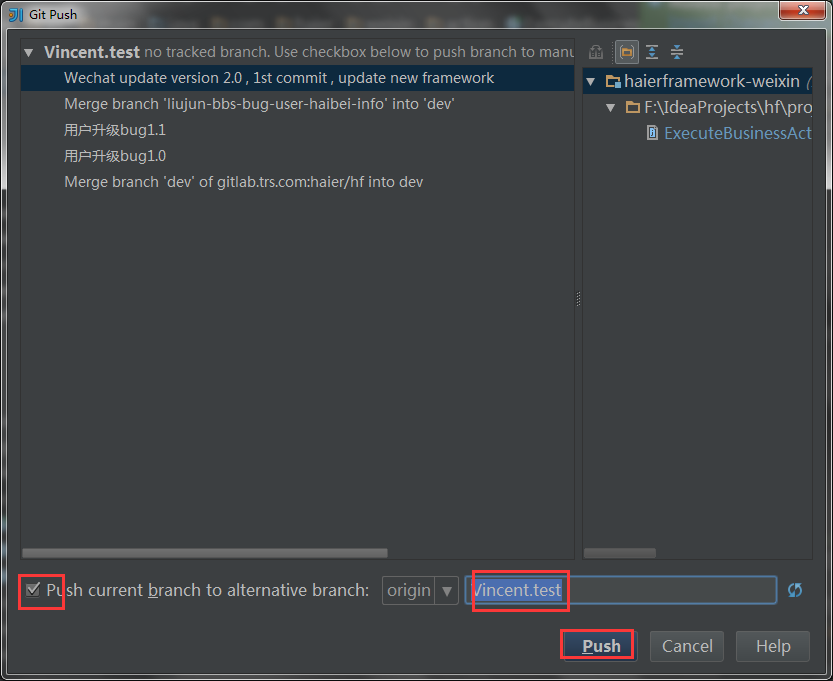
* 解决冲突

如果代码存在冲突，需解决冲突后才能完成合并或者衍合，方法将在Q&A中进行详细说明。

#### 7.5.3推送开发分支到远程

* 保证当前分支为自己的开发分支
* 推送步骤





#### 7.5.3提交合并请求

Gitlab上提交合并请求

#### 7.5.4被请求人合并Dev

#### 7.5.5合并master并重启上线