灵动信息技术

系统框架使用说明

Version 0.1.0

作者:翟光涛

**文档信息**

|  |  |
| --- | --- |
| **项目名:** | 灵动信息技术 |
|  |  |
| **作者:** | 翟光涛 |
| **单位:** | Incito |
| **创建日期:** | 2014-03-03 |
| **上次更新日期:** | 2014-05-28 |
| **使用者:** | 灵动信息技术全体开发员工 |
|  |  |
| **标题:** | 系统框架使用说明 |
| **版本:** | 0.1.0 |
| **分类:** | 规范 |

**修订文档历史记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2014-03-03 | 0.1.0 | 创建文档，并初步完成框架 | 翟光涛 |
| 2014-04-28 | 0.1.0 | 按照惠东建议修订文档。 | 翟光涛 |
| 2014-05-22  2014-05-22 | 0.1.0  0.1.0 | 添加7，单元测试  添加8，系统日志 | 翟光涛  翟光涛 |
| 2014-05-22  2014-05-22 | 0.1.0  0.1.0 | 添加9，异常处理  添加10，缓存使用 | 翟光涛  翟光涛 |
| 2014-05-28 | 0.1.0 | 添加 11，rest接口测试 | 翟光涛 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **目录** |

[1. 概述 1](#_Toc388540481)

[2. 框架定制 1](#_Toc388540482)

[3. 开发环境搭建 1](#_Toc388540483)

[3.1 工具准备 1](#_Toc388540484)

[3.2 代码拉取 3](#_Toc388540485)

[3.3 运行项目 11](#_Toc388540486)

[4. 代码提交 13](#_Toc388540487)

[*4.1* 修改代码，Add to index 13](#_Toc388540488)

[*4.2* 提交到本地库 14](#_Toc388540489)

[*4.3* 提交到gerit审核代码 15](#_Toc388540490)

[*4.4* 审核代码 15](#_Toc388540491)

[*4.5* 同步-更新代码 18](#_Toc388540492)

[*4.6* 提交代码 19](#_Toc388540493)

[5. 项目结构说明 20](#_Toc388540494)

[5.1 项目构建思路 20](#_Toc388540495)

[5.2 项目目录说明 20](#_Toc388540496)

[5.3 模块项目包目录结构说明 20](#_Toc388540497)

[5.4 项目文件命名规则 21](#_Toc388540498)

[6. 项目权限使用说明 22](#_Toc388540499)

[6.1 思路 22](#_Toc388540500)

[6.2 代码使用 22](#_Toc388540501)

[7. 单元测试 23](#_Toc388540502)

[8. 系统日志 26](#_Toc388540503)

[9. 异常处理 27](#_Toc388540504)

[10. 缓存使用 27](#_Toc388540505)

[11. 常见问题 29](#_Toc388540506)

系统使用

# 概述

本文档介绍如何使用框架，进行开发。

# 框架定制

公司项目框架统一维护管理，各子公司项目开发时，先进行框架申请，获取仓库地址或代码包。

框架申请🡪审核🡪按需搭建框架🡪拉取代码🡪参考文档进行项目开发

# 开发环境搭建

## 工具准备

### 安装jdk

版本号：1.7.0.40

官网下载地址：<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

配置库地址：(建设中…)

### 安装开发工具eclipse

推荐使用 配置库共享的 eclipse.rar，已安装好了插件，解压缩即可。

如果不能获取请到这里下载：

<http://www.eclipse.org/downloads/>

版本：Eclipse Juno (4.2)

配置库地址：(建设中…)

### 安装eclipse插件

在eclipse中在线或离线 git插件，m2e插件,wtp web开发插件

如果使用配置库中的eclipse安装包，请忽略这一步。

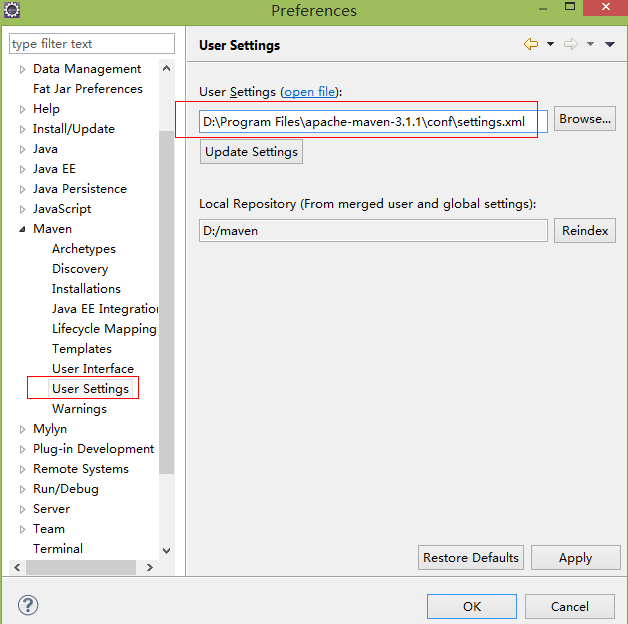
### 安装meavn

(parent项目下载之后做以下配置)

parent项目为基础框架项目

使用parent\maven\settings.xml 配置到eclipse中即可。

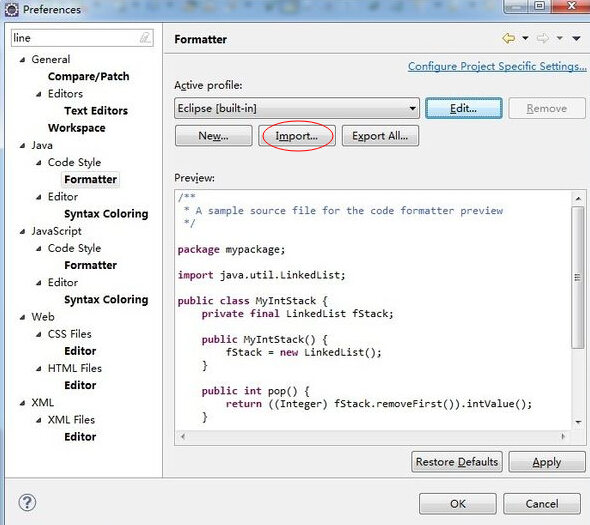
插件安装完毕后，配置eclipse maven设置，如下图



### 导入代码规范模版文件

(parent项目下载之后做以下配置)

模版配置文件：/smart-parent/eclipse/inctio.xml



该配置针对java代码格式化，页面代码避免使用格式化操作。

### 安装mysql

版本号：5.5.35

官网下载地址：<http://dev.mysql.com/downloads/windows/installer/>

配置库地址：(建设中…)

### 安装tomcat

版本号：tomcat-7.0.42

官网下载地址：<http://tomcat.apache.org/download-70.cgi>

配置库地址：(建设中…)

## 代码拉取

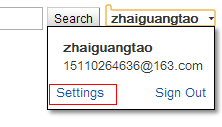
### 项目账号注册

帐号注册请联系所属项目组，项目主管。由项目主管分配帐号和权限。

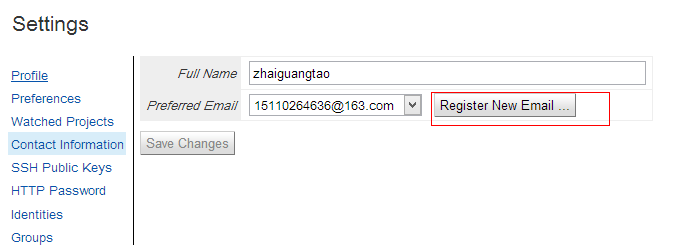
注：如果使用的是github ，请先去github官网注册帐号，将帐号告知主管，由主管分配权限。

拿到git仓库地址之后，直接走3.2.4.

分配完gerit帐户后登陆gerit完善账号信息 [http://192.168.11.100:8090](http://192.168.11.100:8090/)



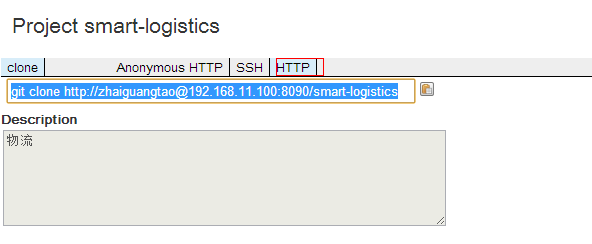
### 完善帐户信息，验证邮箱



你会收到一封邮件，打开邮件中的验证链接，验证通过邮箱地址就会显示在gerit中

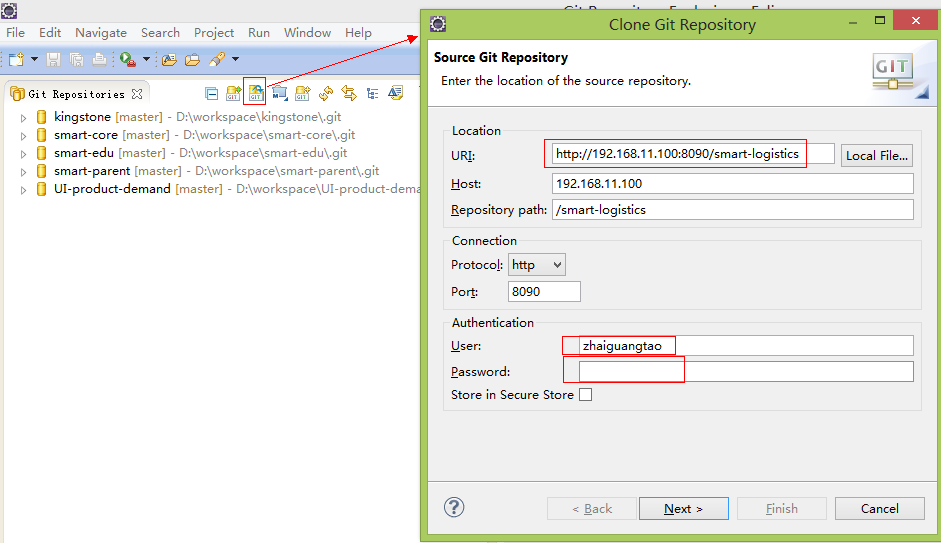
### 获取项目地址



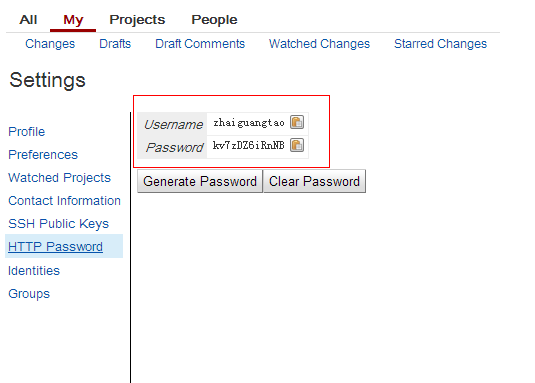


<http://192.168.11.100:8090/smart-logistics>

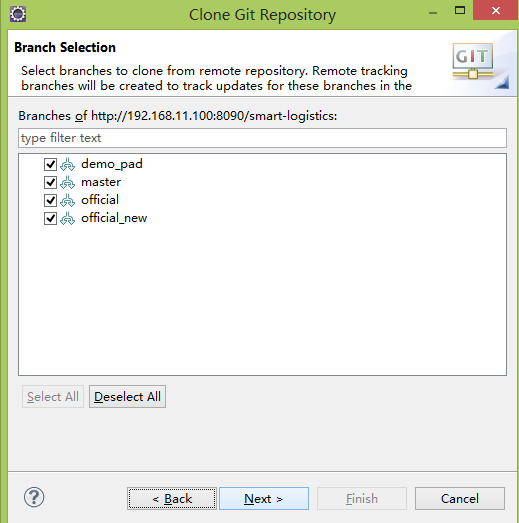
### 在eclipse中clone 项目



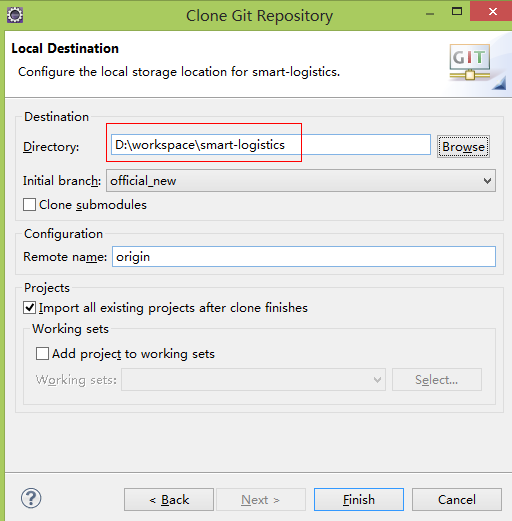
用户名和密码在下图中寻找：



根据项目需要下载指定的版本



next

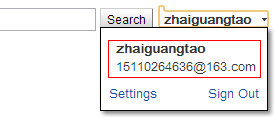


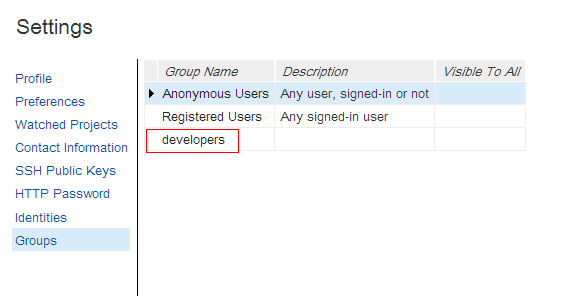
Finish

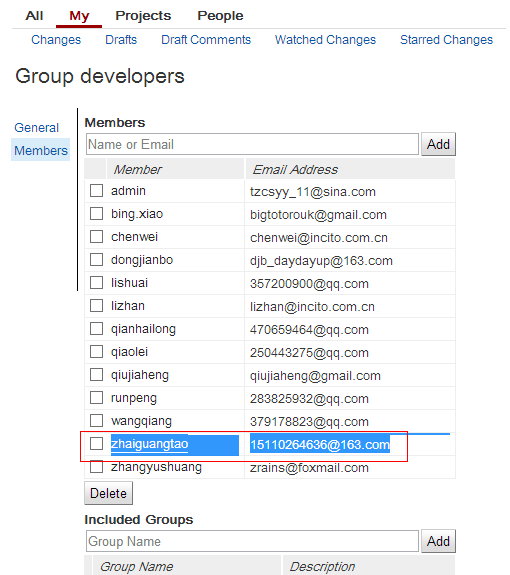
静静的等待项目下载完成

如果提示：read 超时，请检查 邮箱验证是否通过，账号是否添加到该项目组中。

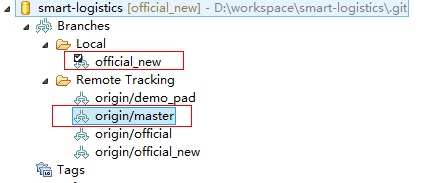
检查以下三项





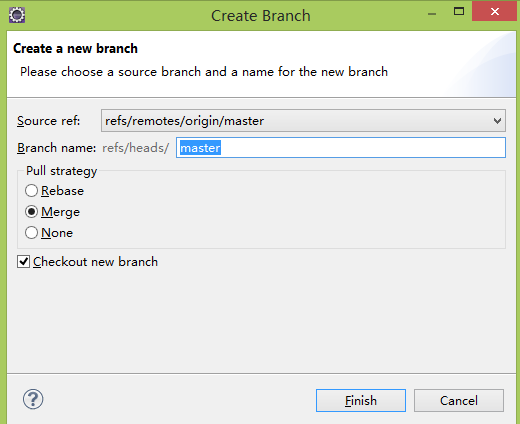


将master分支 拉到本地来

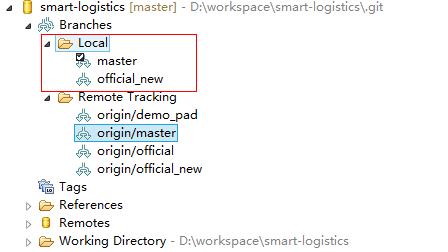


如图下载之后 本地只有一个分支，需要将master分支拉到本地来。

操作方法：对准master右键 create branch…

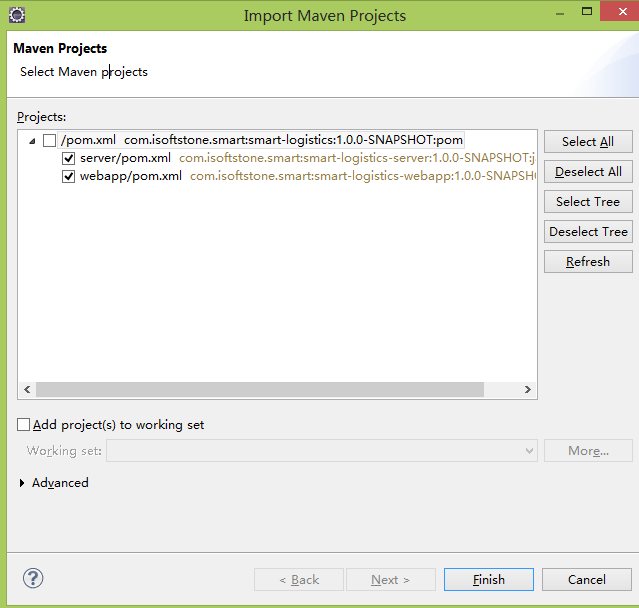


结果如下：



### 检出meavn项目

选中右键，列表最下方 

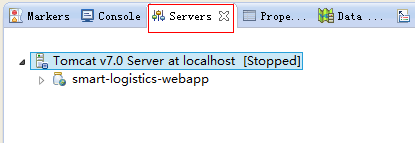


Finish

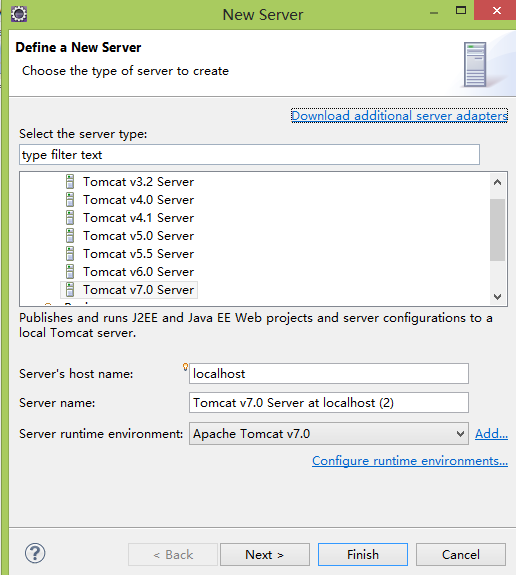
## 运行项目

### 配置tomcat

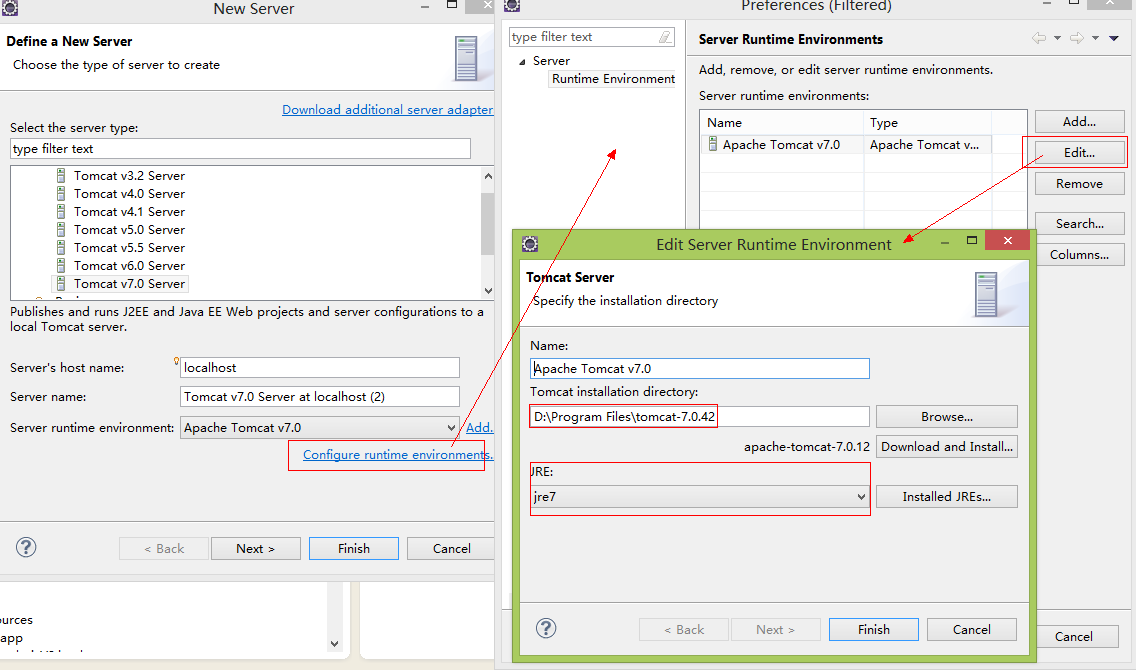
在servers视图中new一个tomcat



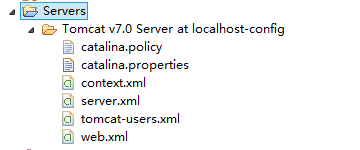
右键new server



配置tomcat位置，jre位置



Tomcat创建好了之后会多一个项目



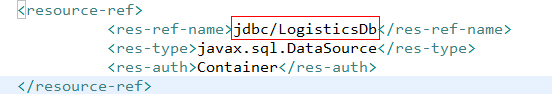
### 导入数据库脚本

建立数据库，通过sql脚本 或者数据库复制工具 导入数据库工具。在次不描述。

### 配置数据源

查看webapp项目的web.xml文件 位置如下：

\smart-logistics-webapp\src\main\webapp\WEB-INF\web.xml



在server.xml 文件中添加 数据源配置

在<GlobalNamingResources> 标记下添加：

|  |
| --- |
| <Resource  name="jdbc/LogisticsDb"  type="javax.sql.DataSource"  auth="Container"  driverClassName="com.mysql.jdbc.Driver"  maxActive="15"  maxIdle="7"  password="passw0rd"  username="root" url="jdbc:mysql://192.168.11.135:3306/smartdb?autoReconnect=true&amp;useUnicode=true&amp;characterEncoding=utf8"  validationQuery="Select 1"/> |

在context.xml中添加

|  |
| --- |
| <ResourceLink global=*"jdbc/LogisticsDb"* name=*"jdbc/LogisticsDb"* type=*"javax.sql.DataSource"*/> |

### 运行项目

运行webapp项目

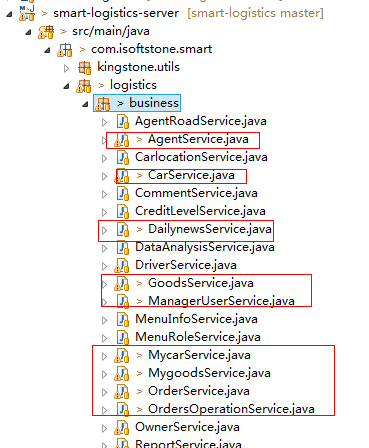


右键run as 🡪run as on server 至此tomcat启动，启动成功后会自动打开项目首页

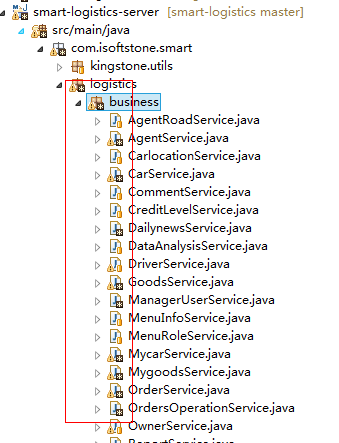
# 代码提交

## 修改代码，Add to index

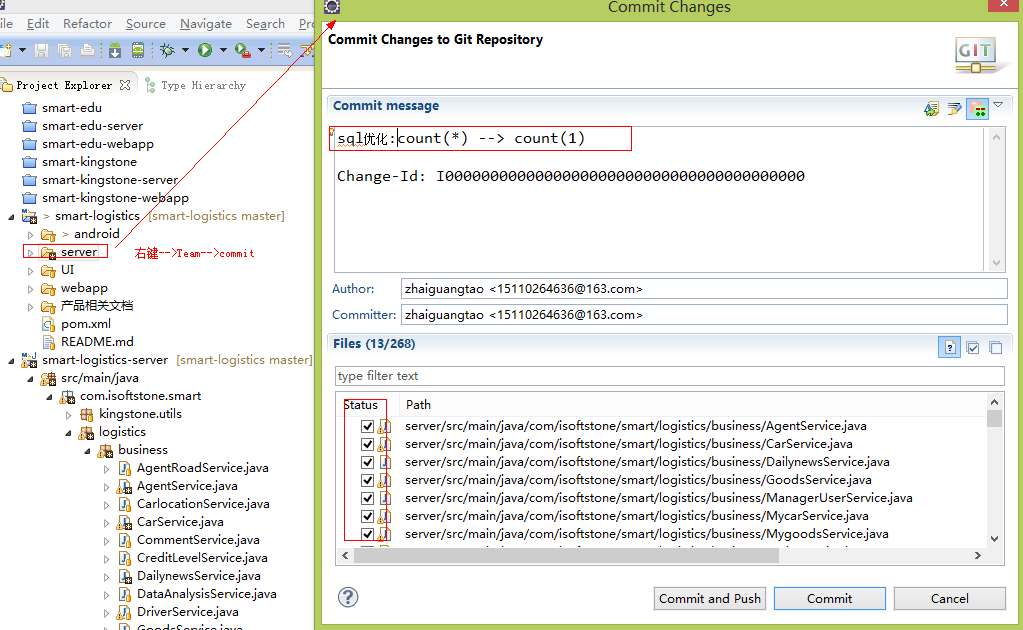
修改后的代码标识如下：选中修改的代码右键Team🡪Add to index



操作后的标记如下;



## 提交到本地库



Commit

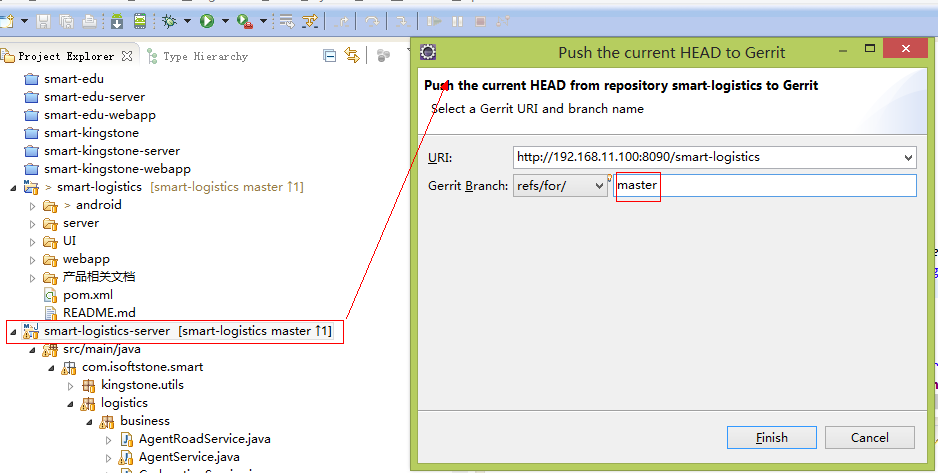
这时候代码提交到了本地库。

注意：代码必填务必写清楚代码修改注释。

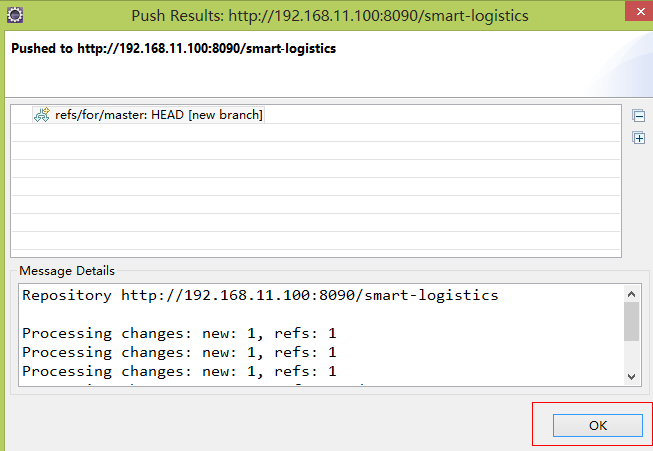
## 提交到gerit审核代码

然后将本地库代码提交到gerit审核

Team🡪Remote🡪Push to gerit

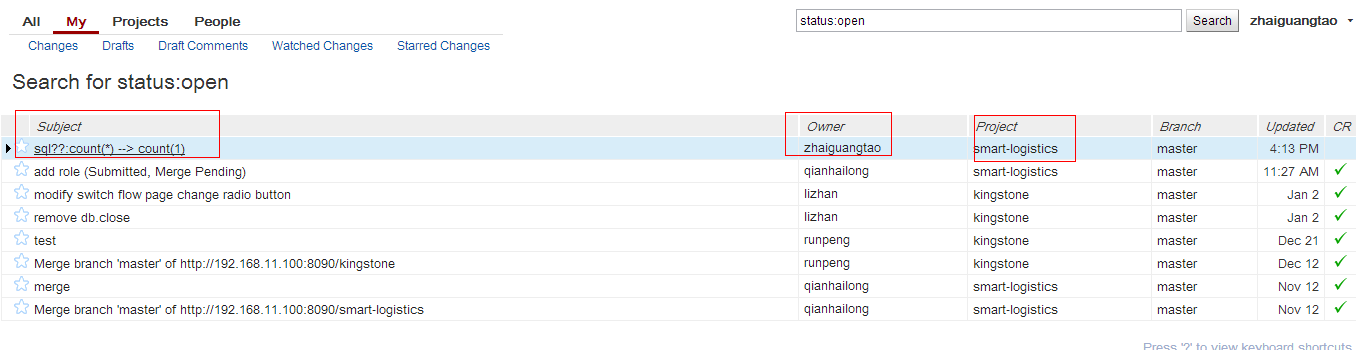


这时候代码提交到了gerit

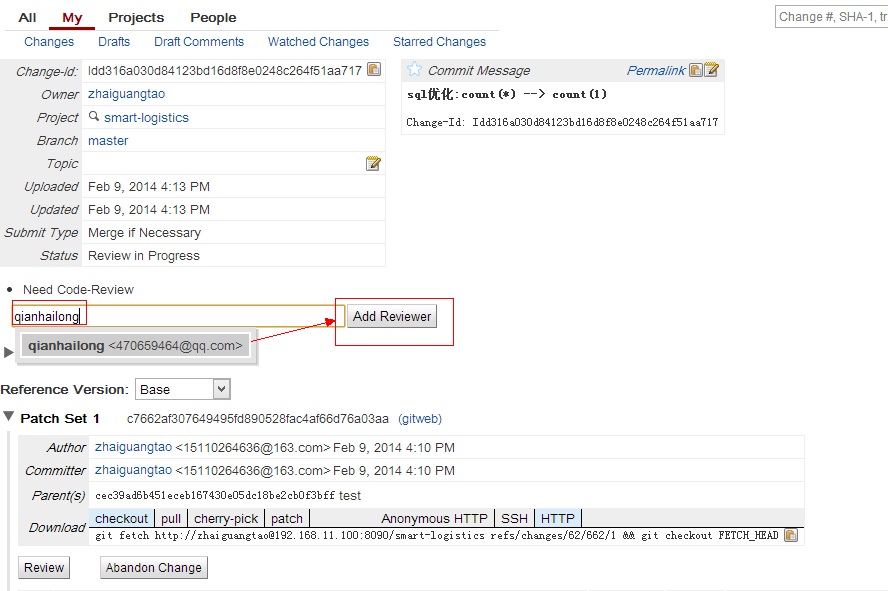


## 审核代码

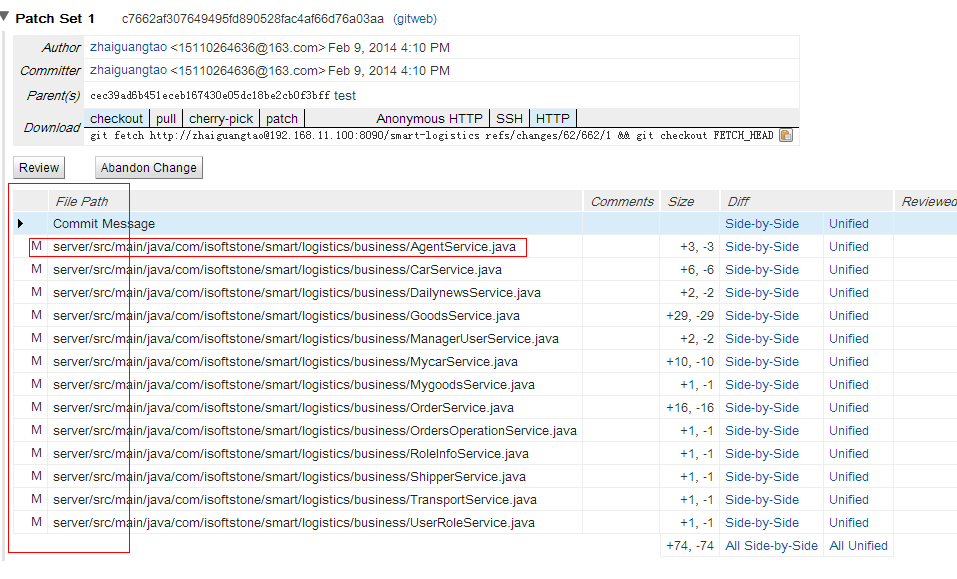
在gerit上查看提交的代码审核记录



指定代码审核者

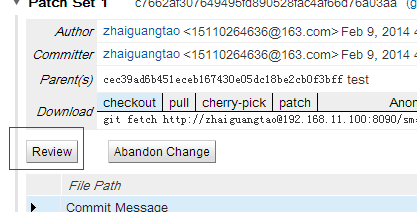


文件提交列表如下：点击记录查看文件修改信息

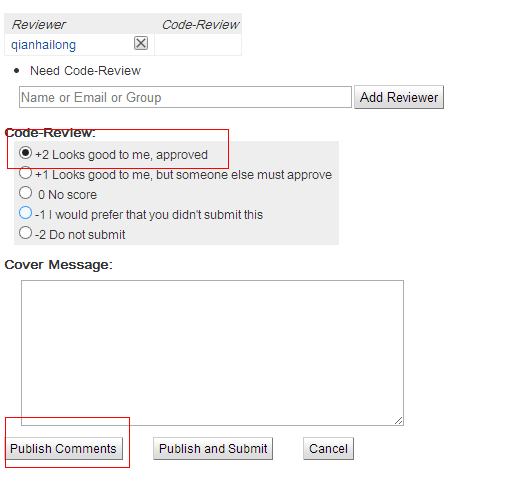




点击主记录页



选中+2 可以直接通过，（待完善）



提交。

## 同步-更新代码

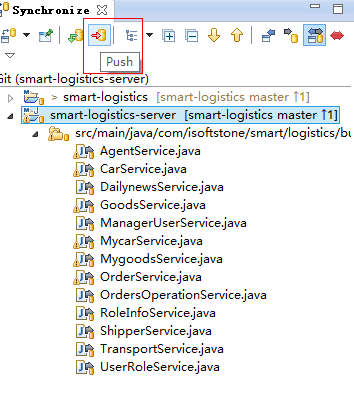
代码审核之后 才可以提交本地仓库代码到服务端仓库

选中项目 右键 同步项目

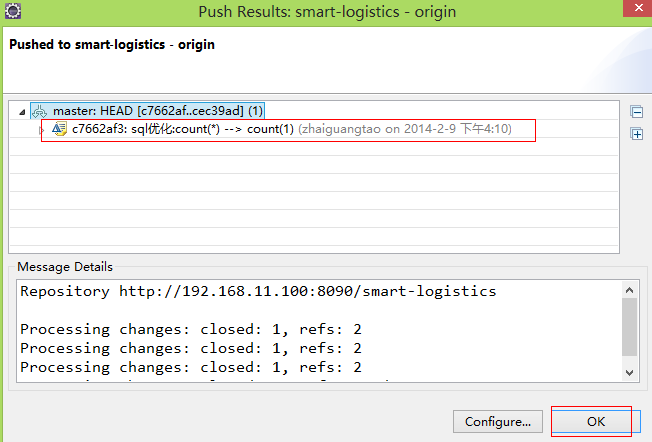
同步中如果有更新，需要先更新再push。

如果有冲突，需要解冲突，然后再走提交本地库🡪提交gerit审核🡪同步项目🡪更新🡪提交

## 提交代码



下图中的注释信息 来自于4.2的填写。



至此提交完成。

# 项目结构说明

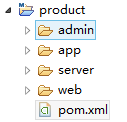
## 项目构建思路

项目基于产品的思路进行构建，目前每个产品依赖parent项目。

在业务平台建立之后每个产品除了依赖parent项目，还可以选择是否依赖业务平台。

## 项目目录说明

项目构建目录如下：



product 为产品名称

admin 后台项目模块的目录

app 移动端接口项目模块的目录

web 前台项目模块的目录

server 服务端，统一业务处理项目模块的目录

pom.xml 统一配置产品的版本信息以及模块的文件

## 模块项目包目录结构说明

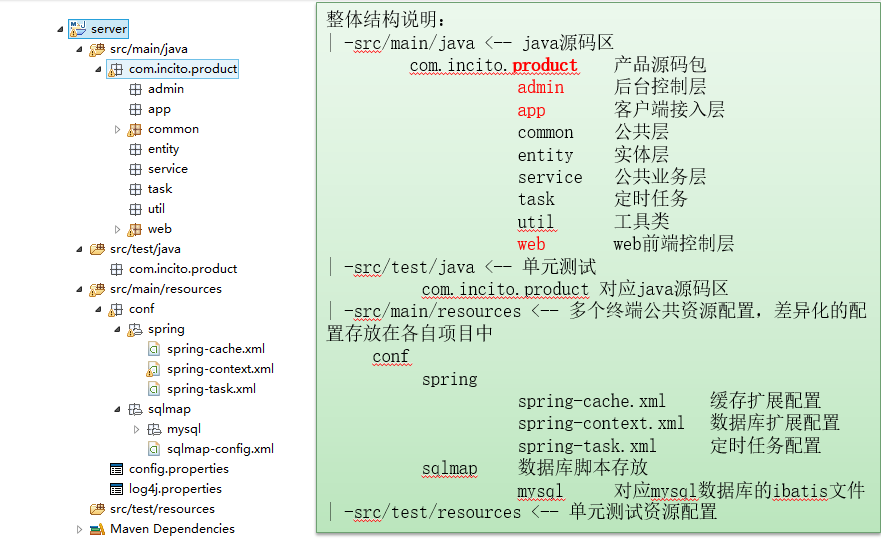
业务项目结构介绍：

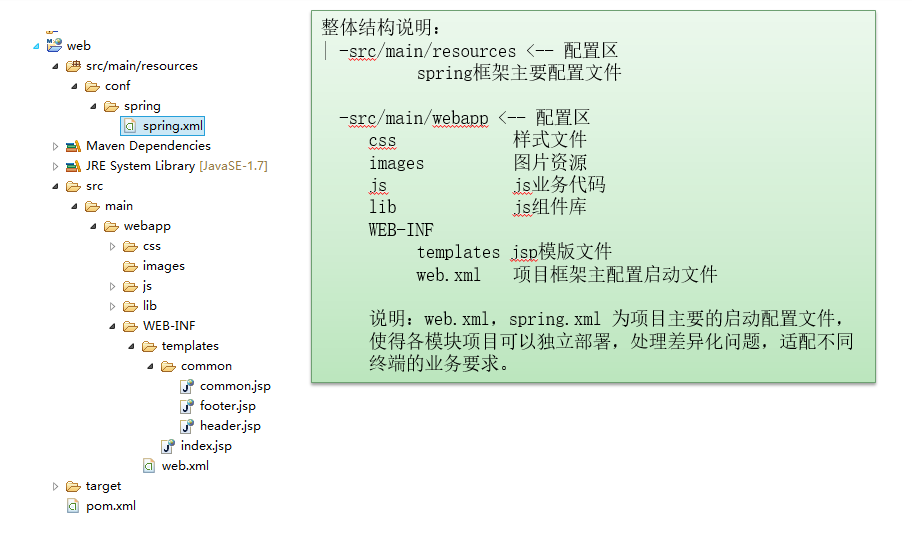
admin,app,web 为服务接入层,对应各模块项目的请求，针对不同终端请求进行处理和适配，共享service层，实现业务统一处理；

entity 实体层，对应数据结构；

util，common，task为辅助性层级；

同样 各模块项目与sever项目的关系是：业务逻辑与视图展现层的分离。





## 项目文件命名规则

此处主要强调几处主要规范，其他地方参考开发规范；

控制层(包名：web，admin，app)格式：XxxCtrl.java

Service(包名：service)：XxxService.java

Ibatis sql 配置文件(sqlmap目录)：xxx\_sql.xml

# 项目权限使用说明

## 思路

1，针对不同终端，通过配置不同的filter分别进行权限处理。

2，通过请求隔离(在spring.xml 中配置)，实现各终端项目控制层独立服务。

3，框架目前通过filter实现项目访问权限管理，后续将联合 业务平台中的 菜单，权限，角色，组织机构等实现更为完善的权限配置管理。

说明：目前代码为基础版，可以解决多种方式（接口请求方式，Ajax请求方式，页面调整方式）的请求权限处理。

## 代码使用

1. 配置需要拦截的url

在示例项目中web项目模块的web.xml中

|  |
| --- |
| <!-- 权限过滤filter 开始 -->  <filter>  <filter-name>SecurityFilter</filter-name>  <filter-class>com.incito.product.common.WebSecurityFilter</filter-class>  <init-param>  <param-name>urls</param-name>  <param-value>/xxx</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>SecurityFilter</filter-name>  <url-pattern>/xxx1/\*</url-pattern>  <url-pattern>/xxx2/\*</url-pattern>  </filter-mapping>  <!-- url-patter 需要过滤的请求规则 ，多个配置 url-patter 列表-->  <!-- 权限过滤filter 结束 --> |

说明：WebSecurityFilter针对web项目进行权限处理，app，admin项目对应的filter是AppSecurityFilter，AdminSecurityFilter.

<param-value>/xxx</param-value> 用于配置url过滤中的白名单，多资源以|隔开

需要过滤的请求规则 ，多个配置 url-patter 列表

1. 权限处理代码

实现是否登录判断：

目前通过session实现，集群模式下登录信息需要存放到第三方缓存，或者依赖web容器实现session复制。

如果需要关联角色信息，根据用户标识，查询角色权限code，在过滤器中通过判断角色code，进行权限判断。

1. 无权限情况下的提示处理

1，针对客户端接口请求：返回对应的json格式和状态标识。

2，针对页面Ajax请求，返回518状态标识，通过页面jquey Ajax 全局状态捕获进行提示操作。

3，针对页面URL请求：根据权限判断跳转到相应的提示页面，或绑定标识，页面解析标识并做对应的信息提示。

# 单元测试

单元测试技术：Spring test + junit4

添加jar包依赖：

|  |
| --- |
| <!-- 加入spring测试依赖包 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.0.0.RELEASE</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>20030825.184428</version>  <type>jar</type>  <scope>test</scope>  </dependency> |

单元测试配置：/src/test/resources/test-context.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns:util=*"http://www.springframework.org/schema/util"*  xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd"*  default-autowire=*"byName"*>    <!--配置webc层 springmvc的范围-->  <context:component-scan base-package=*"com.incito.\*"* >  <!--请求隔离：排除 admin后台 和 app 接口服务 只加载 web前端-->  <context:exclude-filter type=*"regex"* expression=*"com.incito.logistics.app.\*"*/>  <context:exclude-filter type=*"regex"* expression=*"com.incito.logistics.admin.\*"*/>  <context:exclude-filter type=*"regex"* expression=*"com.incito.logistics.web.\*"*/>  </context:component-scan>    <!-- 配置spring-context -->  <context:property-placeholder location=*"classpath:db.properties"* file-encoding=*"utf8"* />    <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*  destroy-method=*"close"*>  <property name=*"driverClassName"* value=*"${db.driver}"* />  <property name=*"url"* value=*"${db.url}"* />  <property name=*"username"* value=*"${db.username}"* />  <property name=*"password"* value=*"${db.password}"* />  </bean>  <bean id=*"sqlMapClient"* class=*"org.springframework.orm.ibatis.SqlMapClientFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"*/>  <property name=*"configLocation"* value=*"classpath:conf/sqlmap/sqlmap-config.xml"*/>  <property name=*"mappingLocations"* value=*"classpath\*:conf/sqlmap/mysql/\*\_sql.xml"*/>  </bean>    <bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>    <!-- 开启注解事务 -->  <tx:annotation-driven transaction-manager=*"transactionManager"*/>  </beans> |

针对普通类测试

|  |
| --- |
| **package** com.incito.base.util;  **import** org.junit.Assert;  **import** org.junit.Test;  /\*\*  \* IDUtil单元测试  \* **@author** zhaigt  \*  \*/  **public** **class** IDUtilTest {  @Test  **public** **void** testGetID() {  String id = IDUtil.*getID*();  Assert.*assertNotNull*(id);  }  } |

针对service层测试

|  |
| --- |
| **package** com.incito.logistics.service;  **import** org.junit.Test;  **import** org.junit.runner.RunWith;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.test.context.ContextConfiguration;  **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  **import** org.springframework.test.context.transaction.TransactionConfiguration;  /\*\*  \* service层 单元测试  \* **@author** zhaigt  \*/  @ContextConfiguration("/test-context.xml")  @RunWith(SpringJUnit4ClassRunner.**class**)  @TransactionConfiguration(defaultRollback=**false**)// 设置 自动提交事物，默认是true 自动对测试数据回滚  **public** **class** AgentServiceTest {  @Autowired  **private** AgentService agentService;    @Test  **public** **void** testCountAgent() {  **int** count = agentService.countAgent("AG0000095", **null**);  System.*out*.println(count);  }  @Test  **public** **void** testCountAllAgents() {  String result = agentService.countAllAgents();  System.*out*.println(result);  }  } |

# 系统日志

日志处理目的：

1. 替代system.out，用于 测试阶段 跟踪调试。
2. 格式化 错误 堆栈信息，方便定位查看
3. 统一处理错误信息，多节点日志 统一收集 入库，后台：错误日志分析 导出 --- 定时反馈开发人员修改, 便于 系统完善.

程序日志处理，分以下四类：

info：消息级别， 很少使用

BaseLog.i(Class clazz, String message);

例如: BaseLog.i(HsUtil.class, "init:参数初始化成功！");

debug：调试级别，用于输出调试信息，替代system.out.println();

BaseLog.d(Class clazz, String message);

例如: BaseLog.d(HsUtil.class, "callWS:参数格式不正确");

warn：警告级别，用于提醒

BaseLog.w(Class clazz, String message);

例如: BaseLog.e(HsUtil.class, "callWS:磁盘空间不足");

error：错误级别，

用于try catch 中的异常信息处理

BaseLog.e(Class clazz, String message, Exception e)

例如: BaseLog.e(HsUtil.class, "callWS:未传入服务参数串", e);

用于参数缺失，等错误信息的输出

BaseLog.e(Class clazz, String message)

例如: BaseLog.e(HsUtil.class, "callWS:未传入服务参数串");

# 异常处理

1. 控制层 必须 处理异常， 根据业务情况 给予友好提示.
2. Service 层异常需要向外抛出异常。
3. 容器设置全局异常处理方式，处理那些没有抛到最外层没有被处理的异常。

# 缓存使用

项目使用spring cache标准来实现：

spring @Cacheable @CachePut @CacheEvict的使用说明：

例如：

|  |
| --- |
| @Cacheable(value = "employee", key = "'cacheType\_'+#surname" ,condition = "#age < 25")  public Person findEmployeeBySurname(String firstName, String surname, int age) {  return new Person(firstName, surname, age);  } |

参数说明：

@Cacheable 针对方法配置，能够根据方法的请求参数对其结果进行缓存

value 必选 指明了缓存将被存到哪个缓存块 如果使用EHCache，就是ehcache.xml中声明的cache的name

key 可选 默认按照方法的所有参数进行组合；可使用SPEL表达式 调用方法参数

condition 可选 根据条件匹配是否缓存结果, 可使用SPEL表达式

@CachePut 和 @Cacheable 不同的是，它每次都会触发真实方法的调用

@CacheEvict 支持如下几个参数：

value：缓存位置名称，不能为空，同上

key：缓存的key，默认为空，同上

condition：触发条件，只有满足条件的情况才会清除缓存，默认为空，支持SpEL

allEntries：true表示方法调用后将立即清除value中的全部缓存，默认为false，

beforeInvocation：是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存

例如：

//清除掉指定key的缓存

@CacheEvict(value="andCache",key="#user.userId + 'findById'")

public void modifyUserRole(SystemUser user) {

System.out.println("hello andCache delete"+user.getUserId());

}

//清除掉全部缓存

@CacheEvict(value="andCache",allEntries=true)

public final void setReservedUsers(String[] reservedUsers) {

System.out.println("hello andCache deleteall");

}

一般来说，我们的更新操作只需要刷新缓存中某一个值，所以定义缓存的key值的方式就很重要，最好是能够唯一，

因为这样可以准确的清除掉特定的缓存，而不会影响到其它缓存值 ，

比如这里针对用户的操作，使用(userId+方法名称)的方式设定key值 ，当然，你也可以找到更适合自己的方式去设定。

配置 cache 的支持

需要添加 cache的命名空间申明：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:cache=*"http://www.springframework.org/schema/cache"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*  *http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache-3.1.xsd"*>    <!-- 启用缓存注解功能，这个是必须的，否则注解不会生效，另外，该注解一定要声明在spring主配置文件中才会生效 -->  <cache:annotation-driven cache-manager=*"cacheManager"*/>    <!-- spring自己的换管理器，这里定义了两个缓存位置名称 ，既注解中的value -->  <bean id=*"cacheManager"* class=*"org.springframework.cache.support.SimpleCacheManager"*>  <property name=*"caches"*>  <set>  <!-- 缓存块 default与andCache 对应value="andCache" ,缓存产品可以更换为 memcached-->  <bean class=*"org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean"*  p:name=*"default"* />  <bean class=*"org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean"*  p:name=*"cache1"* />  </set>  </property>  </bean>    </beans> |

spring cache 的原理：

即它是基于动态生成的 proxy 代理机制来对方法的调用进行切面，

这里关键点是对象的引用问题，如果对象的方法是内部调用（即 this 引用）而不是外部引用，

则会导致 proxy 失效，那么我们的切面就失效，

也就是说上面定义的各种注释包括 @Cacheable、@CachePut 和 @CacheEvict 都会失效

# Rest接口测试

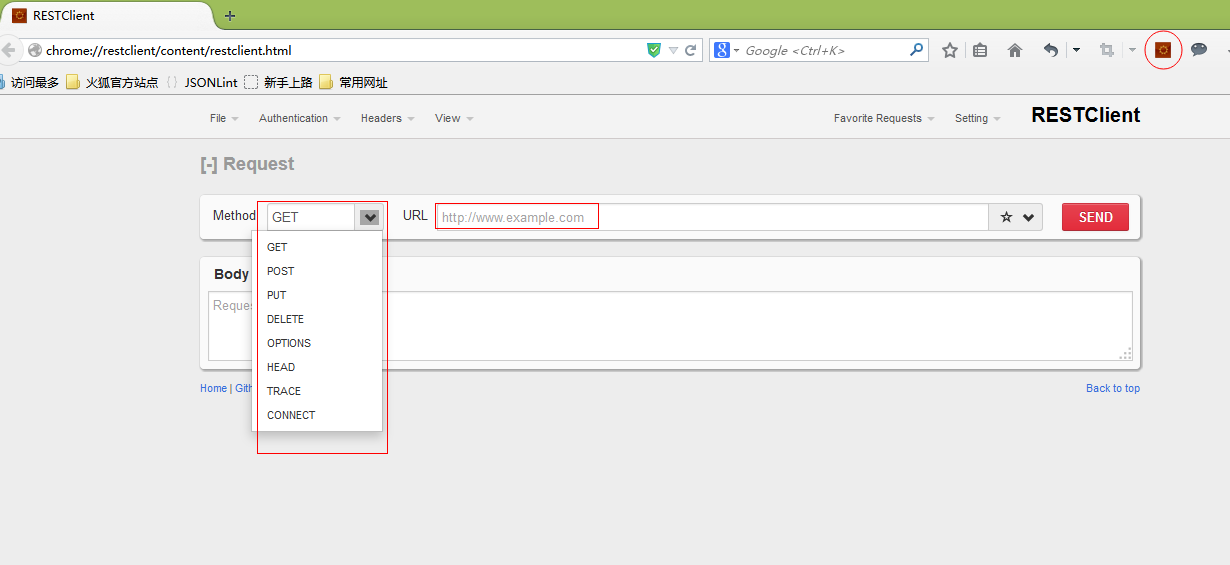
通过firefox插件RESTClient来进行Restful接口测试。

1. 安装插件

打开firefox 附加组件管理器，搜索 RESTClient ，在搜索结果列表点击安装即可。

1. 插件使用说明

插件主界面，选择对应的请求方式，输入URL即可。

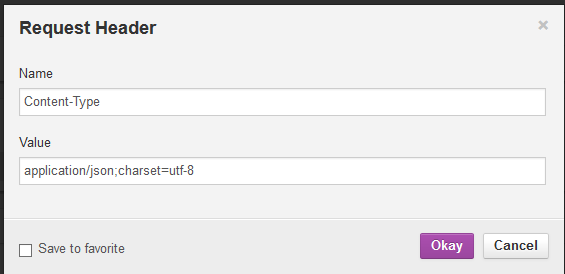


需要注意的是 POST方式下，需要设置Headers

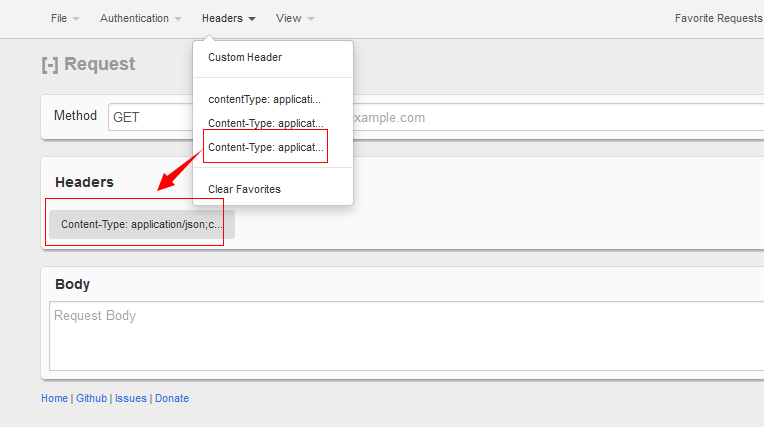
点击头部导航中的Headers，在以下框中填写：

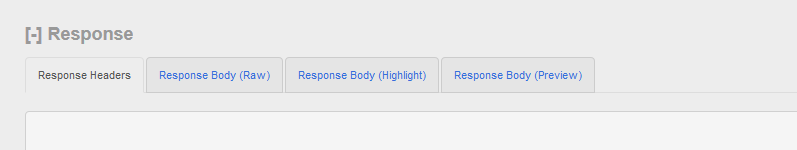
Content-Type

application/json;charset=utf-8



选中 配置好的设置值。





在底部的 响应结果区域可以对响应结果进行分析。

另一款 测试工具：基于google的插件

<http://www.getpostman.com/>

访问上面网址，按照提示，按照插件。

# 常见问题

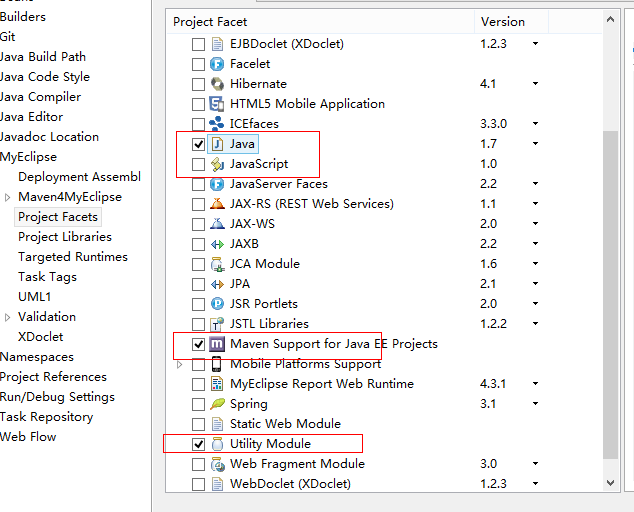
1. pom.xml报错

查看报错信息，如果新down下来的项目就报错请尝试更新meavn依赖，如果还报错，请检查对应的依赖项目是否下载。

如果是依赖的jar包报错，请检查下载的jar包是否完整，在网络不稳定的情况下，下载的jar包可能会不完整，请在meavn仓库中删除对应的jar目录，更新meavn依赖，重新下载。

1. 项目下载后不能自动构建成java project类型 或者 web project类型

检查项目的属性 facets



1. 项目有红色！类型的报错

更新maven依赖。

检查build path设置

1. 不能提交代码

能拉取代码不能push代码的情况，请检查如下配置

