系统架构

v4.5.1

[环境 2](#_Toc511742762)

[服务器与开发 2](#_Toc511742763)

[框架 3](#_Toc511742764)

[开发流程 3](#_Toc511742765)

[环境搭建 3](#_Toc511742766)

[开发流程 5](#_Toc511742767)

[配置 7](#_Toc511742768)

[日志 7](#_Toc511742769)

[端配置 8](#_Toc511742770)

[核心配置 8](#_Toc511742771)

[新增配置 8](#_Toc511742772)

[JMX 9](#_Toc511742773)

[公共资源 9](#_Toc511742774)

[核心组件 9](#_Toc511742775)

[任务调度 9](#_Toc511742776)

[异步任务 10](#_Toc511742777)

[支付 10](#_Toc511742778)

[推送 10](#_Toc511742779)

[Redis 11](#_Toc511742780)

[SpringCache 11](#_Toc511742781)

[分页查询 11](#_Toc511742782)

[FastDFS 11](#_Toc511742783)

[权限 12](#_Toc511742784)

[其他 12](#_Toc511742785)

[UTIL 12](#_Toc511742786)

[convert 12](#_Toc511742787)

[date 12](#_Toc511742788)

[email 13](#_Toc511742789)

[json 13](#_Toc511742790)

[sms 13](#_Toc511742791)

[string 13](#_Toc511742792)

[view 14](#_Toc511742793)

[其他 14](#_Toc511742794)

[JS插件 14](#_Toc511742795)

[Ajax 14](#_Toc511742796)

[上传 14](#_Toc511742797)

[城市选择 15](#_Toc511742798)

[时间选择 15](#_Toc511742799)

[树Ztree 15](#_Toc511742800)

[DataTables 16](#_Toc511742801)

[编辑器 16](#_Toc511742802)

[弹窗 16](#_Toc511742803)

[部署 17](#_Toc511742804)

[普通部署 17](#_Toc511742805)

[Jenkins部署 18](#_Toc511742806)

[开发规范 20](#_Toc511742807)

[包规范 20](#_Toc511742808)

[命名规范 20](#_Toc511742809)

[SQL规范 21](#_Toc511742810)

[接口规范 21](#_Toc511742811)

[SVN 22](#_Toc511742812)

[注释 22](#_Toc511742813)

[返回码 23](#_Toc511742814)

[其他 24](#_Toc511742815)

[测试模块： 24](#_Toc511742816)

[SpringCloud 24](#_Toc511742817)

[项目打包 24](#_Toc511742818)

[更新 25](#_Toc511742819)

[图表 25](#_Toc511742820)

# 环境

## 服务器与开发

JDK1.8

MySQL 5.6+

Maven 3.3+

Redis 2.8+

CentOS 7

开发工具：IDEA 2017.3

## 框架

SpringBoot 2.0 RELEASE

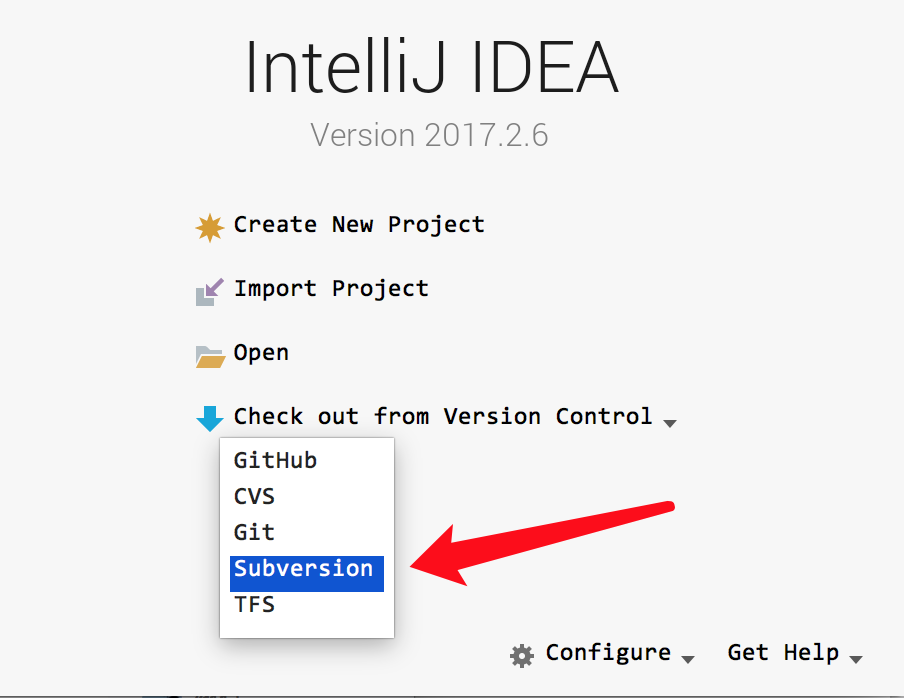
SpringCloud 2.0

Mybatis 3.4.5

# 开发流程

## 环境搭建

1、使用IntelliJ IDEA，打开软件，从SVN检出项目

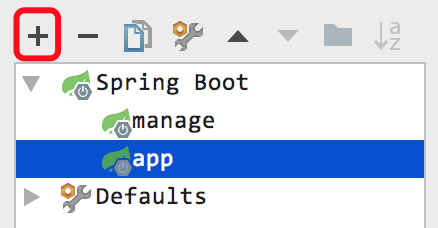


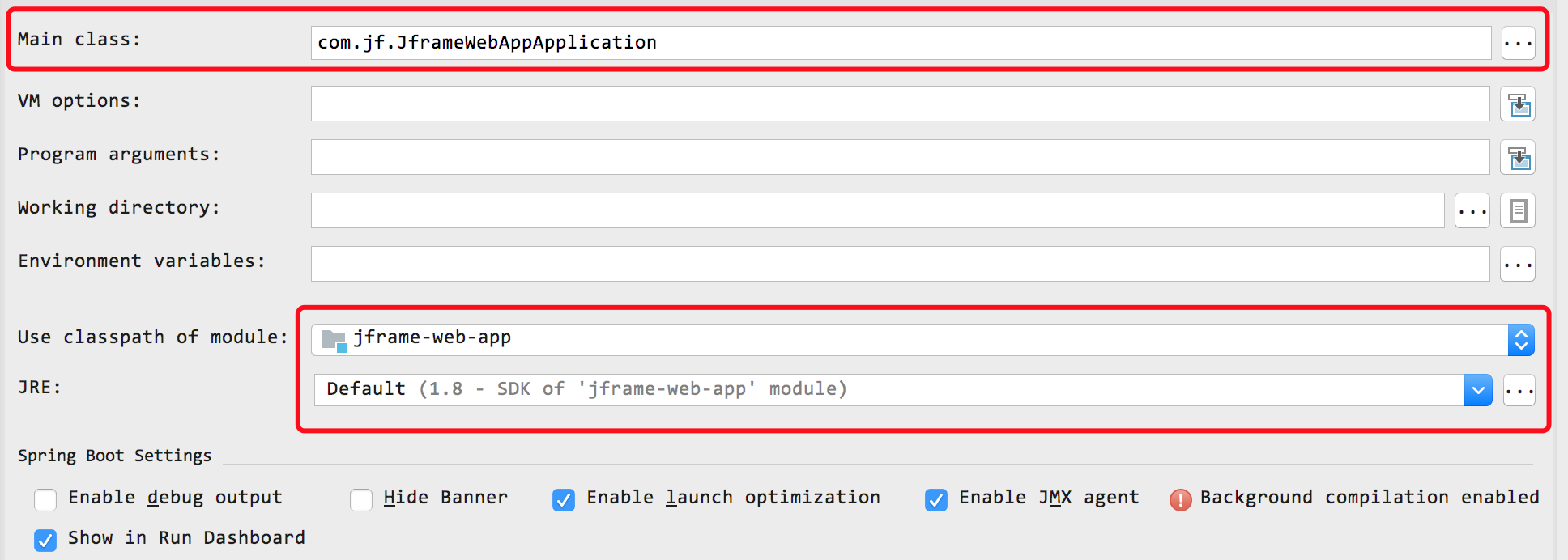
2、点击Open，打开项目所在路径（最外层文件夹），不建议使用Import Project。

3、点击最右侧的选项卡Maven Project，，,

再点击‘+’，选择 jframe/pom.xml文件，之后会自动关联模块，并下载Jar包。

4、 

**点击上图1处**，显示，点击‘+’，找到SpringBoot(如没有，则可能未安装专业版IDEA)。



配置如上，选择对应的引导类和模块，确认即可。

**点击2处**，选择JDK版本，最低**1.8**.

## 开发流程

1、数据库实体类和Mapper生成

jframe-utils/src/main/java/com/jf/db/GenerateBeansAndMybatisUtil



**建立文件夹**，**修改数据库连接池**，右键Run该类，即可在文件夹内生成对应的实体类和Mapper文件。

将对应的实体类和Mapper分别放置到**jframe-model**模块中。

实体类都需要继承BaseVo类。

mapper提供的接口：

findByCondition (BaseVo baseVo);

findById (Long id);

insert (User bean);

insertBatch (List<User> list);

update (User bean);

delete (Long id);

提供通用的xml代码片段：

COMMON.DATE、COMMON.PAGE、COMMON.USER

baseCondition、simpleColumn、allColumn

*对于SQL规范会在最后的规范中说明。*

2、**Service**

jframe-core/../service

如果Service需要事务，请在类前加上@Transactional

注入Mapper，使用@Resource或@Autowire

3、在jframe-web-manage开发页面

如果是页面，直接在类注解**@Controller**，方法上@RequestMapping，返回String；

如果全是JSON数据，使用**@RestController**，并且方法只需要注解@GetMapping或@PostMapping，返回ResMsg类型。

页面模板在**resources/templates**，以**ftl**结尾，按模块划分。

head标签加入**<#include "include.ftl"/>**引入全局JS/CSS。

页面布局统一，**查询栏，工具栏，列表，分页**。

4、在jframe-web-app开发APP接口

一般使用**@RestController+ @GetMapping**或**@PostMapping**注解，返回**ResMsg**。

操作**Token**，直接在需要的方法参数前加上：**@Token**，目前是**token和id**进行绑定，用于用户识别。

# 配置

## 日志

jframe-core/src/main/resources/logback-boot.xml

分别配置：**LOG\_HOME**和**LOG\_HOME\_PRO**为**测试**和**生产环境**的日志地址

日志名称通过环境变量获取（具体查看部署脚本），未指定则使用***jframe***为日志文件名称。请确保日志**目录存在**，并且**可写**。

## 端配置

jframe-web-test/src/main/resources/application.yml

具体可参考注释，注意以下几点：

1. **server.port**和**spring.application.name**唯一

2. **app.pro**为**生产环境**需要调用的配置

3. **app.db.enabled**为组件开关（如不启用可以不写）

## 核心配置

jframe-core/src/main/resources/application-dev.yml

该环境是测试环境，如需生产环境，在端配置指定：**spring.profiles.active: pro**

具体配置依次如下：

日志（logging）、Tomcat服务(server)、缓存(cache)、session、mvc、

上传限制、资源目录、json、Freemarker、数据源(druid)及监控、

异步任务、邮件、缓存(redis)、JMS(activemq)、fastfdfs、Eureka注册、

熔断器(feign)、监控(Actuator)

**系统配置：**appkey、自定义静态目录、日志目录、dfs nginx地址、支付 宝、微信、上传等。

## 新增配置

在application-dev.yml新增属性，要求如下：

1. TAB必须为两个空格

2. 明确层级关系

3. 建议全部小写，多个单词用下划线分割

在jframe-core/src/main/java/com/jf/system/conf/SysConfig.java 新增字段（如果新增的属性为三级，需要将第二级定义为静态内部类，再定义字段），并生产get/set方法。

## JMX

启用JMX监控：rmi.enabled=true

run启动脚本，在java命令后追加：

-Djava.rmi.server.hostname=192.168.1.199

## 公共资源

jframe/public

- static 静态文件

- common 公共的css、全局JS函数

- library JS插件库

- templates freemarker模板

# 核心组件

## 任务调度

jframe-core/src/main/java/com/jf/system/schedule/ScheduleConfig.java

新建任务类，如日任务：DayJob，使用注解：**@EnableScheduling @Component**

在具体的方法上加上注解：**@Scheduled**(cron = "0 0 23 \* \* ?")

可在类中直接注入**Service**

以上为Spring自带的，如果使用**Quartz**，在jobs包中建立类继承**QuartzJobBean**，然后在**QuartzConfig.java**创建@Bean，如同*JobDetail()*

## 异步任务

jframe-utils/src/main/java/com/jf/sms/SMService.java

异步方法所在类需要注解：**@Component**

在需要异步的方法上加上注解：**@Async**

## 支付

jframe-core/src/main/java/com/jf/system/third/alipay

**1. 支付宝**

下单：AliPayUtil.alipay(String body, String subject, Double price, String orderNum, String url)

转账：transfer(String orderNum, Double price, String account, String realname)

检查到账状态：check(String orderNum, String thirdNum)

Service调用：**AliPayService**.alipay("充值", "账户充值", money, orderNum)

**2. 微信同 WxPayService**

## 推送

jframe-core/src/main/java/com/jf/system/third/jpush/JPushService.java

注入**JPushService**，调用方法

sendPush(**final int** devive, **final** String alert, **final** Map<String, String> extras, **final** String... id)

devive 设备类型

alert 推送内容

extras 额外参数

id 设备id

## Redis

注入：**RedisTemplate**

调用接口方法

如：**template.** **opsForValue().set**(String key, String value)

## SpringCache

**Service**层中，在查询方法上加入注解：

@Cacheable(value = **"user"**, key = **"'findUserById'+#id"**)

在增删改方法中加入注解：（更新会直接删除老数据）

@CacheEvict(value = **"user"**, key = **"'findUserById'+#user.id"**)

## 分页查询

**pagehelper**

condition.setPage(**true**);  
List<User> list = **userMapper**.findByCondition(condition);  
**return new** PageInfo(list);

默认不分页，需要设置**page = true**

## FastDFS

注入：**FastFileStorageClient**

调用：

StorePath storePath = **storageClient**.uploadFile(file.getBytes(), suffix);  
String filePath = **config**.getFdfsNginx() + storePath.getFullPath();

## 权限

对于后台权限，事先在后台 系统管理->模块管理 配置好菜单和按钮功能后，

在具体方法中加入注解：**@AuthPassport**，默认验证登录和权限，可配置login=true/false , right=true/false，login表示登录，right表示操作权限。

@RequestMapping(**"/userList"**)  
@AuthPassport(right = **false**)  
**public** String userList() {

}

## 其他

如 **ActiveMQ、WebSocket、Flume、Redisson**，在项目中未使用到，具体实现可在core/com/jf/system中查看示例。

# UTIL

## convert

Convert.doubleAdd(**double** d1, **double** d2) 浮点数相加

Convert.doublesAdd(**double**... doubles) 不定参数相加【请确保最少有两个参数】

Convert.doubleFormat(**double** d) double保留两位小数点

MoneyUtil.transMoney(Double money) 万元金额转换

number2CN(BigDecimal numberOfMoney) 把输入的金额转换为汉语中人民币的大写

## date

DateUtil.getCurrentTime(Integer type) 获取当前时间

DateUtil.dateToStr(Date date) date转string

DateUtil.strToDate(String dateString) string转date

DateUtil.getYearAndMonth(**boolean** flag) 获取年月

DateUtil.between(String d1, String d2, String date) 检查某时间是否在时间段内 格式：yyyy-MM-dd

DateUtil.getMonthFirstDay() 获取当月第一天

DateUtil.getMonthLastDay() 获取当月的最后一天

DateUtil.diffDays(Date startDate, Date endDate) 计算两个时间之间相差的天数

DateUtil.remainDateToString(Date startDate, Date endDate) 计算剩余时间

DateUtil.add(Date date, **int** type, **int** value) 日期加减

RelativeDateFormat.format(Date date) 获取相对时间

## email

注入EmailService

调用sendTemplateMail (Map<String, Object> root, String toEmail, String subject, String templateName)

root 参数，键值对

toEmail 收件人

subject 主题

templateName 模板名称

## json

JSONUtils.toJSONString(List<T> list) 将List对象序列化为JSON文本

JSONUtils.toJSONString(Object object) 将对象序列化为JSON文本

JSONUtils.toJSONString(JSONArray jsonArray) 将对象转换为JSON对象数组

JSONUtils.toJSONObject(Object object) 将对象转换为JSON对象

## sms

注入SMService

调用send(String content, String phone)

content 内容

phone 手机号

模板发送sendSms(Map<String, String> root, String templateName, String... phones)

root 参数，键值对

templateName 模板名称

phones 手机号

## string

StringUtil.getOrderCode() 获取订单编号

StringUtil.getSmsCode(**int** size) 获取手机验证码

StringUtil.strToArray(String answer) 将字符串转换为一个新的字符数组

StringUtil.MD5Encode(String origin) 对密码进行加密

StringUtil.randomFilename(String file) 文件随机命名

StringUtil.getColumnNum(String num) 阿拉伯数字转中文

StringUtil.getTokenId() 构造SESSIONID

## view

新建类，继承AbstractXlsView

参考jframe-web-manage/src/main/java/com/jf/controller/view/UserExcel.java

## 其他

**db**、**file**、**http**、**system**、**page**更多请参考具体类

# JS插件

## Ajax

Ajax.ajax({  
 **url**: **'/admin/user/userEdit'**,  
 **params**: $form.serialize(),  
 success: **function** (data) {  
 **if** (data.**code** == 0) {  
 showMsg(data.msg, 1, **function** () {  
 });  
 } **else** {  
 showMsg(data.msg, 2);  
 }  
 }  
});

## 上传

<**a class="img-area-btn" id="avatar"**></**a**>

**$**(**"#avatar"**).Uploader({  
 **limit**: 1, 限制数量  
 **default**: **'${user**.**avatar}'**, 默认图片  
 **type**: 1 1.图片 2.文件  
});

## 城市选择

<**input type="text" id="city-picker" name="address" value="${user**.**address }" class="form-control" placeholder="住址"**>

cityPicker(**"#city-picker"**, **"distrdict"**, **"address"**);

参数：id；类型（city|district|street）；返回值类型（address|code）

## 时间选择

<**input type="text" class="form-control input-sm" id="startDate" name="startDate" placeholder="开始时间"**>

datePicker(**'#startDate,#endDate'**, **"yyyy-mm-dd"**, 2);

参数：id（多个）；格式化（*4 3 2 1*）；格式化（*yyyy yyyy-mm yyyy-mm-dd yyyy-mm-dd hh:ii*）

## 树Ztree

<**ul id="tree" class="ztree"**></**ul**>

**var *zindex***;  
**var *tree*** = **""**;  
**var *setting*** = {  
 **check**: {  
 **chkboxType**: {**"Y"**: **"p"**, **"N"**: **"ps"**},// 勾选checkbox对于父子节点的关联关系  
 **chkStyle**: **"checkbox"**,  
 **enable**: **true** // 是否复选框  
 },  
 **data**: {  
 **simpleData**: {  
 **enable**: **true** }  
 }  
};

***tree*** = **$**.**fn**.zTree.init(**$**(**"#tree"**), ***setting***, data);  
***tree***.expandAll(**true**);

## DataTables

<**table id="table" class="table table-striped table-bordered table-hover" width="100%"**></**table**>

*具体JS参考：user/user\_list.ftl*

## 编辑器

<**textarea name="content" style="width**: 100%;**height**: 500**px**;**"**>**${**article.content **}**</**textarea**>  
<**div class="word-count"**>字数统计：<**span id="wc"**></**span**></**div**>

**var** editor;  
**KindEditor**.ready(**function** (K) {  
 editor = K.create(**'textarea[name="content"]'**, {  
 **resizeType**: 1,  
 **allowPreviewEmoticons**: **false**,  
 **allowImageUpload**: **true**,  
 **uploadJson**: **'/uploadwithKE'**,  
 afterBlur: **function** () {  
 **this**.**sync**();  
 },  
 afterChange: **function** () {  
 K(**'#wc'**).html(**this**.count(**'text'**));  
 }  
 });  
});

## 弹窗

*openLayerUrl*(title, width, height, url, maxmin) *打开layer url*

*openLayerContent*(title, width, height, $obj) *打开HTML窗口【对象】*

*openLayerHTML*(html, width, height, func) *打开HTML窗口【HTML】*

*openLayerLoading*() *加载动画*

具体参考***global.js***或者直接参考layui：http://layer.layui.com/

*\*大部分插件都是二次封装，方便调用，可在jframe/public/static/common/js/global.js查看。*

*\*其他插件见jframe/public/static/library/plugins*

# 部署

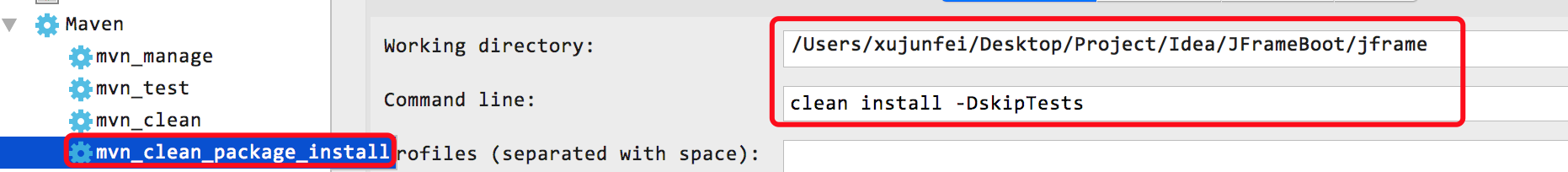
服务器目录结构如下：

**/data/wwwroot/{module\_name}** 模块

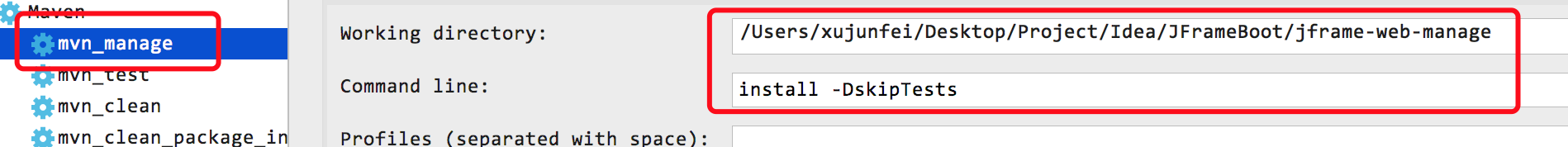
**/data/wwwlogs** 日志

## 普通部署

**clean install/package -DskipTests**



上面是编译全部模块，如果要编译**单个**：



编译完成后，会在各个端下的target目录生成如下：



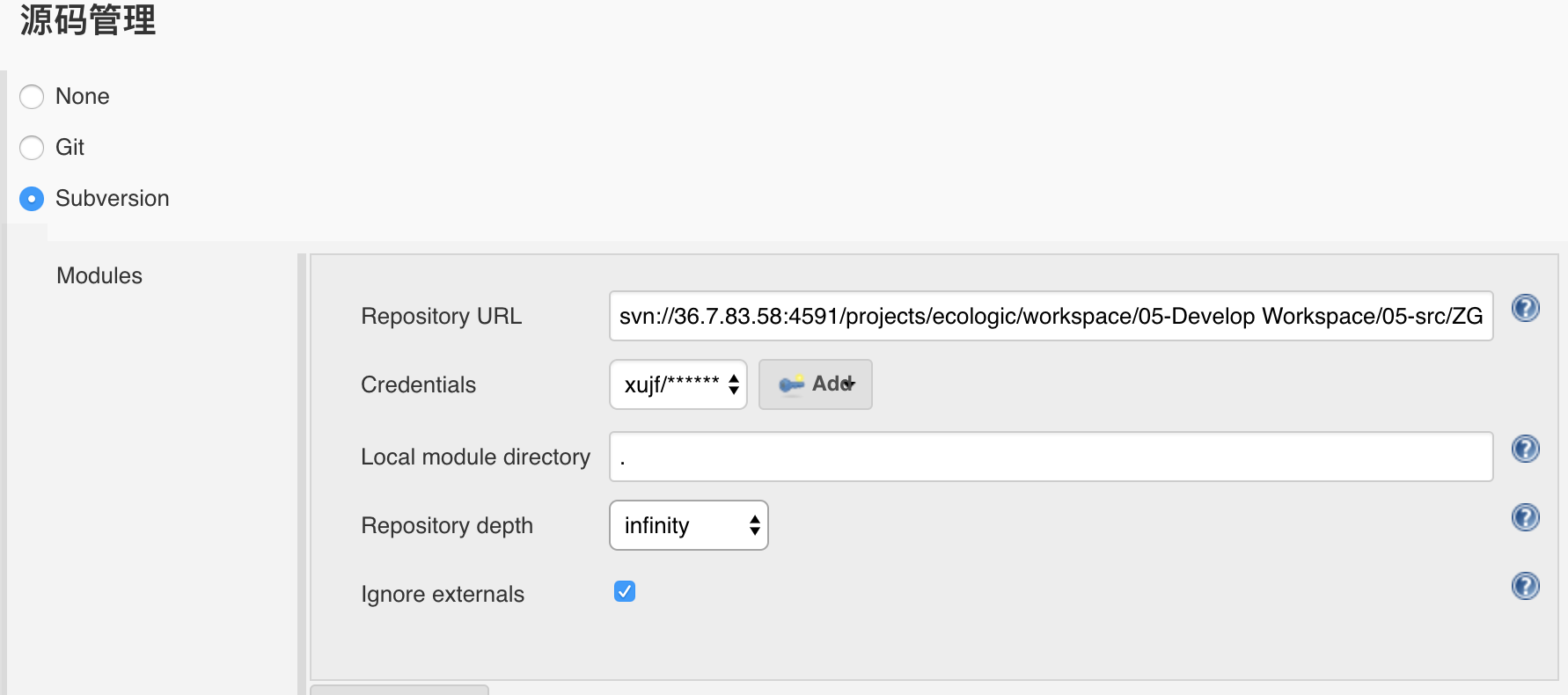
将包上传到服务器的**/data/wwwroot/test**目录下，并将**jframe/build/run**脚本上传到/data/wwwroot/目录，通过**./run test [port]**启动项目。

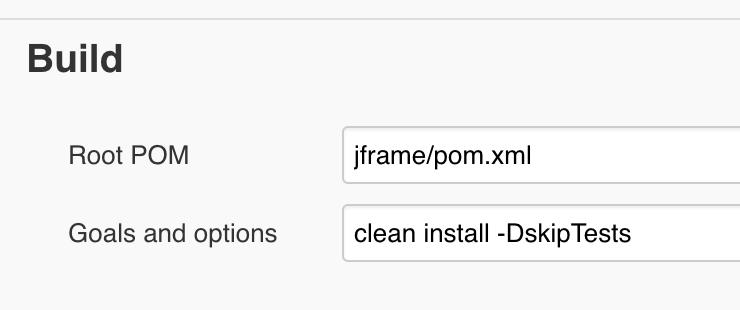
注意：run命令中需要制定**JAVAPATH**(Java的路径)，才能正确执行，不加端口号，默认使用配置中的端口。默认使用**pro生产环境**。

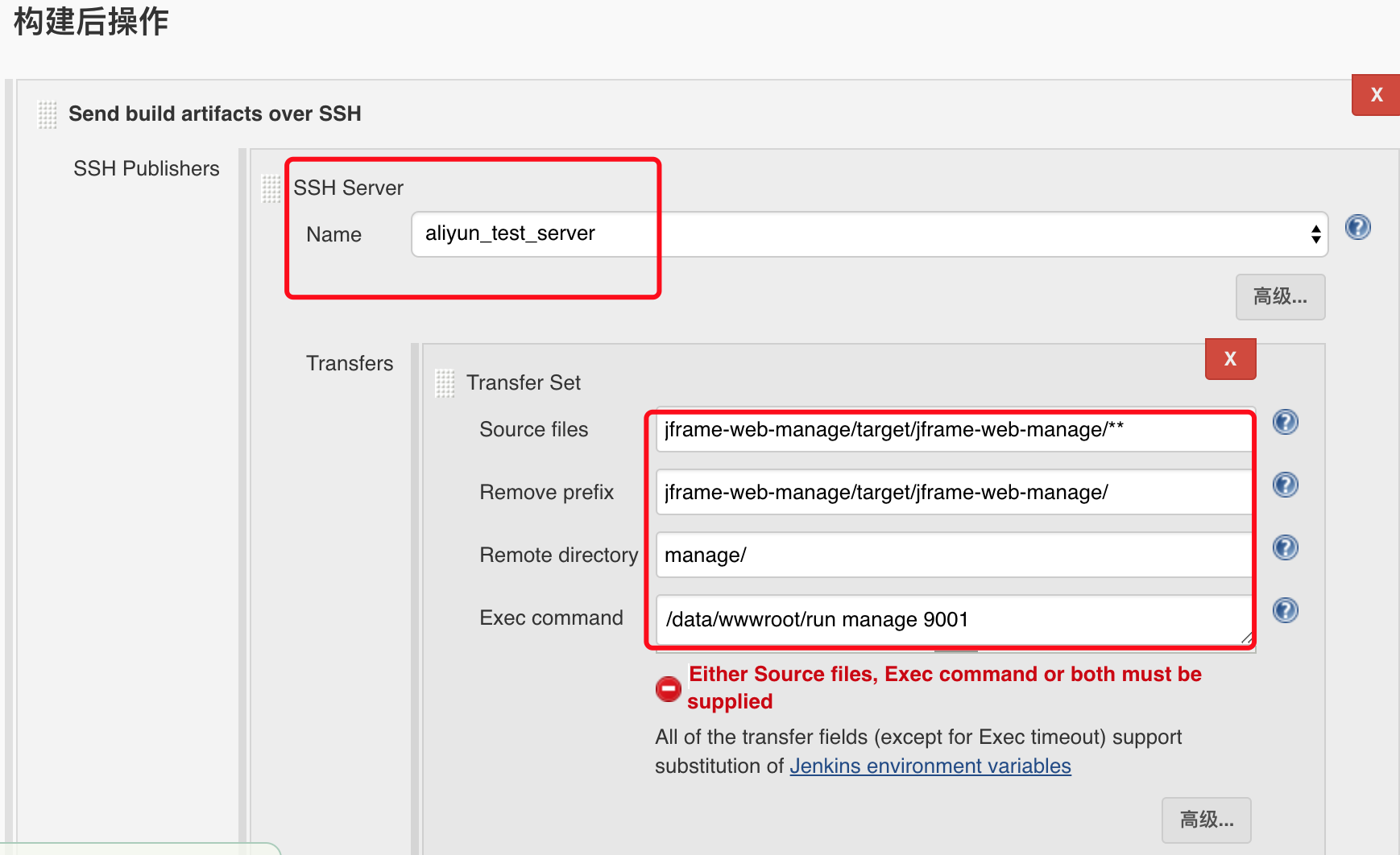
## Jenkins部署

安装**Jenkins**，并启动后，将本项目配置好，在manage后台右上角，点击“自动化部署”，然后输入Jenkins的**IP**和**项目名称**，点击刷新按钮，查看状态，点击部署按钮可自动部署（如服务可用）。

以下是项目的配置：







# 开发规范

## 包规范

**包名**：全部小写，尽量一个英文单词，如：user order article

service通常以业务类型进行分包，如，

用户模块：user；订单模块：order；充值模块：charge

**controller**通常以模块进行分包，如，订单模块：order

## 命名规范

**1、Java**：

**类名**，首字母大写，多个英文时，遵循驼峰形式

如：AppController OrderItem UserService

**常量**，命名全部大写，多个使用‘\_’ 隔开

如：SESSION\_NAME

**枚举**，命名全部大写

如：SUCCESS YES

**代码格式化，**快捷键：Ctrl+Alt+L

**2、数据库**：全部英文小写，多个英文使用‘\_’ 隔开，表使用‘t\_’开头

如：**表名** t\_user t\_order

**字段名**user\_name user\_id order\_num

**3、HTML**：全部英文小写，多个英文使用‘\_’ 隔开

如：user\_list.html user\_edit.ftl index.jsp

**4、JavaScript**：文件全部英文小写，多个英文使用‘\_’或‘-’ 隔开

方法和变量也遵循驼峰形式，但首字母是小写

如：**文件**：app-index.js global.js

**方法**：getVer() showMenu()

## SQL规范

关键字全部**大写**

如：**SELECT** id,name **FROM** t\_user

**UPDATE** t\_user **SET** name = '' **WHERE** ...

对于mapper，需对生成的代码进行精简和优化，如有默认值的不需要新增，不需要更新的字段删除。

格式化，结构清晰；对于id、时间、类型、状态等设置**默认值**

## 接口规范

1、APP接口统一以/app开头，返回类型为：

无数据：ResMsg(0, **SUCCESS**)

有数据：ResMsg(0, **SUCCESS, data**)

地址：*/app/user/userList*，如下：

**user**是根据类名，如UserController来命名的，或是模块名

**userList**为请求名称，首字母小写，后**遵循驼峰形式**

2、分页：**PageInfo**类，直接注入ResMsg对象的data属性中。

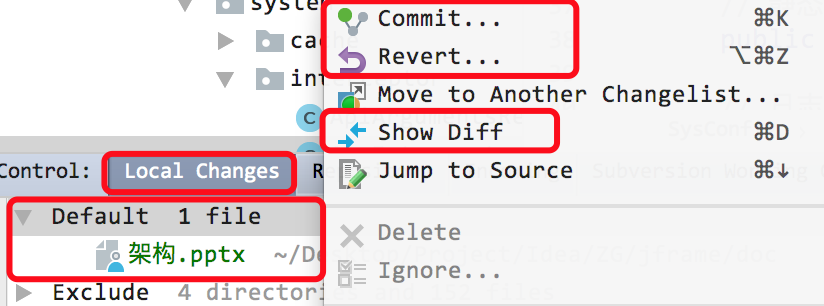
3、**restful**风格（建议）： /app/user/10001 等同于：/app/user?userId=10001。

4、app请求需要在**header**中注入：Req-Type=APP token=？

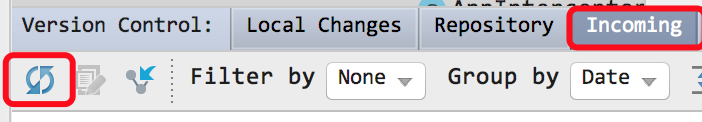
5、可通过IDEA自带的插件进行接口测试：**Tools** -> **REST Client**，将测试的结果录入到API文档中供APP开发人员参考。

## SVN

一般开发人员只负责某一块，不涉及其他人员的模块，如果涉及到，最先提交的等待，后一个需要提交的先更新，后提交，第一个人再更新（相同类会直接覆盖）。



以上为本地提交常用的功能；更新版本（右键->Update）↓↓↓



对于一些**配置文件**、**JUnit测试类、IDE自动生成的配置文件**，请不要提交。

如将日志的路径改为自己电脑的路径，该文件不应提交，将其移至排除目录（新建目录Exclude）。

## 注释

任何**类、方法、常量**都需要编写注释

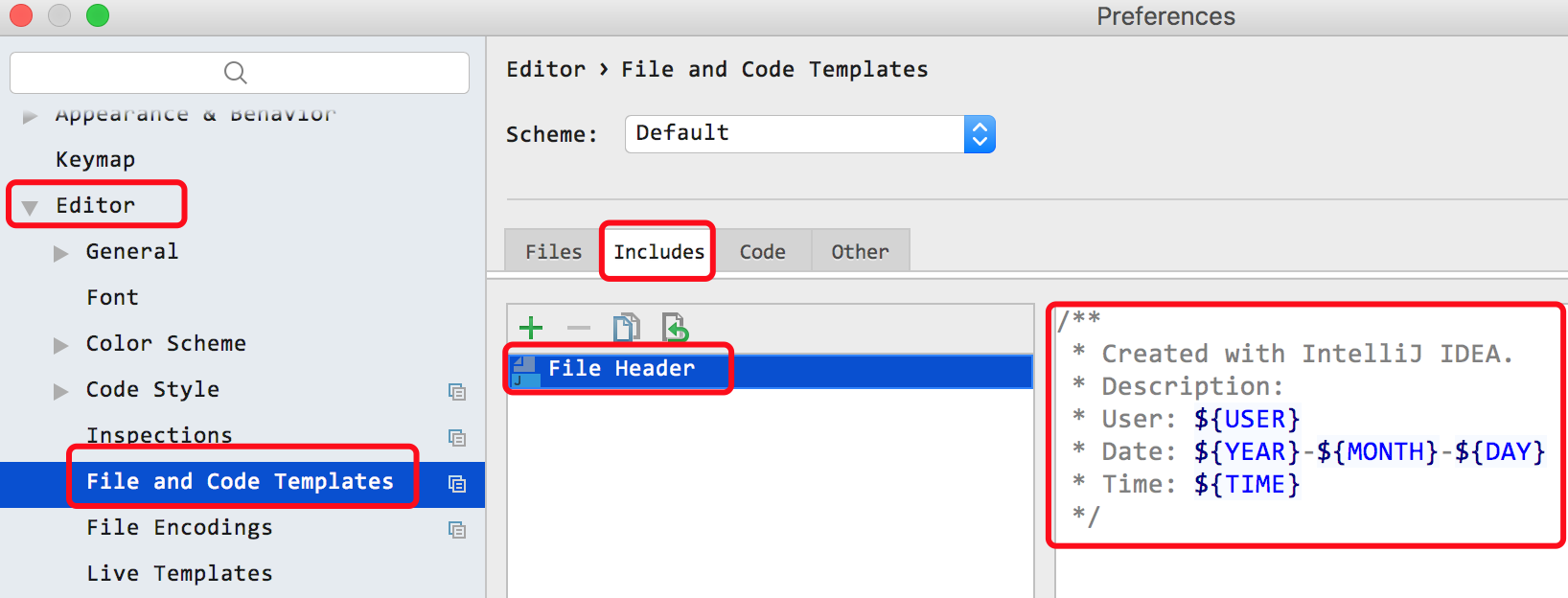
模板（可在IDE里设置）：

/\*\*  
 \* Created with IntelliJ IDEA.  
 \* Description: 说明  
 \* User: xujunfei  
 \* Date: 2017-11-28  
 \* Time: 15:52  
 \*/

**public class** ClassName {

/\*\*  
 \* 删除用户  
 \*  
 \* **@param *userId*** \* **@return** \*/  
**public int** deleteUser(Long userId) {

// APPID  
**public static** String *appid*;



## 返回码

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 成功 | 100 | 无权限 |
| >0 | 失败 | 99 | 未登录 |
| -1 | 系统异常 | 98 | APPKey不正确 |
|  |  | 97 | 非法访问 |

# 其他

## 测试模块：

jframe-web-test

提供：以上所有**Java组件**和**JS插件(包括页面)**的测试DEMO

## SpringCloud

由于项目中并未用到，但提供DEMO

具体如下：开启**cloud-server**端，启动**jframe-service-order[服务者]**端，浏览器输入：ip:10000/打开eureka控制台，如发现服务则表示注册服务已成功启动；

随后启动**jframe-web-test模块[消费者]**，注意，需要将**app.eureka.enabled**设置为true（监控同理），rest接口调用在src/main/java/com/jf/restapi/**OrderRestService**，而后请求*/order*接口即可访问服务。

## 项目打包

具体打包形式见**jframe/build/package.xml**，**format**标签可以设置打包成文件夹或者tar包形式。

<**format**>dir/tar.gz</**format**>

## 更新

可在Github查看架构更新

<https://github.com/jacksonrick/JFrameBoot>

# 图表

