

**BDP-Nuts脚手架开发手册**

1.0

文档管理

**更改历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **状态** | **日期** | **负责人** | **更改原因** |
| 1.0.0 | 起草 | 2018/03/20 | 沙辉 | 起草 |
| 1.0.1 | 修订 | 2018/03/23 | 沙辉 | 增加业务spring配置 |
| 1.0.2 | 修订 | 2018/03/26 | 沙辉 | 增加gdk相关的jar包；登录页面可配 |
| 1.0.3 | 修订 | 2018/03/30 | 沙辉 | 增加退出登录、编辑用户信息接口 |
| 1.0.4 | 修订 | 2018/04/27 | 沙辉 | 增加获取字典表的API |
| 1.0.5 | 修订 | 2018/7/6 | 沙辉 | 增加业务打包、安装基础脚本说明，增加自定义登录页说明 |
| 1.0.6 | 修订 | 2018/7/12 | 沙辉 | 增加操作日志、消息推送接口说明 |
| 1.0.7 | 修订 | 2018/08/09 | 沙辉 | 增加用户增删改监听注册功能 |
| 1.1.1 | 修订 | 2018/08/23 | 邹午阳 | 增加登录/登出回调 |

目录

[1 介绍 5](#_Toc519090281)

[1.1 文档目的 5](#_Toc519090282)

[1.2 适用范围 5](#_Toc519090283)

[1.3 术语和缩写 5](#_Toc519090284)

[1.4 参考资料 5](#_Toc519090285)

[2 脚手架组成 5](#_Toc519090286)

[3 快速开始 6](#_Toc519090287)

[3.1 开发环境要求 6](#_Toc519090288)

[3.2 本地搭建步骤 6](#_Toc519090289)

[4 目录结构 8](#_Toc519090290)

[5 技术栈 10](#_Toc519090291)

[6 接口说明 11](#_Toc519090292)

[6.1 菜单相关 11](#_Toc519090293)

[6.2 用户相关 11](#_Toc519090294)

[6.3 角色相关 12](#_Toc519090295)

[6.4 部门相关 13](#_Toc519090296)

[6.5 系统相关 14](#_Toc519090297)

[6.6 退出登录 14](#_Toc519090298)

[7 本地开发环境配置 14](#_Toc519090299)

[7.1 鉴权平台配置（BDP-AUTH） 15](#_Toc519090300)

[7.2 认证平台配置（BDP-CAS） 15](#_Toc519090301)

[7.3 当前系统配置 15](#_Toc519090302)

[7.4 数据源配置 15](#_Toc519090303)

[7.5 配置菜单 16](#_Toc519090304)

[8 打包 17](#_Toc519090305)

[9 安装 18](#_Toc519090306)

[9.1 基础运行环境安装 18](#_Toc519090307)

[9.2 业务系统安装 18](#_Toc519090308)

[9.2.1 文件说明 19](#_Toc519090309)

[9.2.2 安装配置脚本制作 19](#_Toc519090310)

[9.2.3 安装步骤 20](#_Toc519090311)

[9.2.4 安装检测 20](#_Toc519090312)

[10 常用功能示例 21](#_Toc519090313)

[10.1 上传下载 21](#_Toc519090314)

[10.2 增删改查 22](#_Toc519090315)

[10.3 国际化 23](#_Toc519090316)

[10.4 视图解析 23](#_Toc519090317)

[10.5 自定义登录页面（1.0.12+） 24](#_Toc519090318)

[10.5.1 登录流程 24](#_Toc519090319)

[10.5.2 开发步骤 24](#_Toc519090320)

[11 常用工具 26](#_Toc519090321)

[11.1 常用工具包 26](#_Toc519090322)

[11.2 druid监控 26](#_Toc519090323)

[11.3 代码执行时间监控日志 26](#_Toc519090324)

[12 常见问题 27](#_Toc519090325)

[12.1 打包问题 27](#_Toc519090326)

[12.2 安装问题 27](#_Toc519090327)

[12.3 界面问题 27](#_Toc519090328)

# 介绍

## 文档目的

本着以“技术求窄、服务求快、产品求精、资源积累”的思想，以降低开发者的门槛，快速上手为目的，根据WEB端技术指导规范，制定此开发脚手架（BDP-Nuts）,供公司产品线、各研发中心快速使用。

## 适用范围

本文档适用于公司内部以及各研发中心所有使用java开发的WEB项目；

## 术语和缩写

|  |  |
| --- | --- |
| 名称 | 描述 |
| BDP | (base develop platform) Web端统一开发平台 |
| BDP\_Nuts | 脚手架（开发基础环境） |
| BDP\_CAS | 统一认证平台（单点登录认证） |
| BDP\_AUTH | 统一鉴权平台（用户管理中心） |

## 参考资料

《WEB端技术指导规范.doc》

# 脚手架组成

**BDP\_CAS:** 统一认证平台（单点登录认证），CAS是耶鲁大学的一个开源的企业级单点登录系统，我们在此基础上内置了：密码，PKI，指纹，短信，IC卡等常用的登录方式；

**BDP\_AUTH:** 统一鉴权平台（用户管理中心），提供用户，角色，机构，岗位，职务，功能菜单，系统配置等基础功能的平台；

**BDP\_Nuts:** 脚手架（开发基础环境），脚手架只提供便捷的开发环境，包含对登录，功能权限的拦截，相关权限接口以client客户端jar包形式内置，如下图BDP-Client所示；脚手架中所有的页面都是demo页面，无任何功能页面，需要业务进行开发；脚手架做为BDP组件之一，和其他2个组件直接的关系如下图所示；



脚手架采用业界熟知的SpringMVC做为MVC框架，持久层以Spring-JDBC，并内置若干工具类供业务方便使用。

# 快速开始

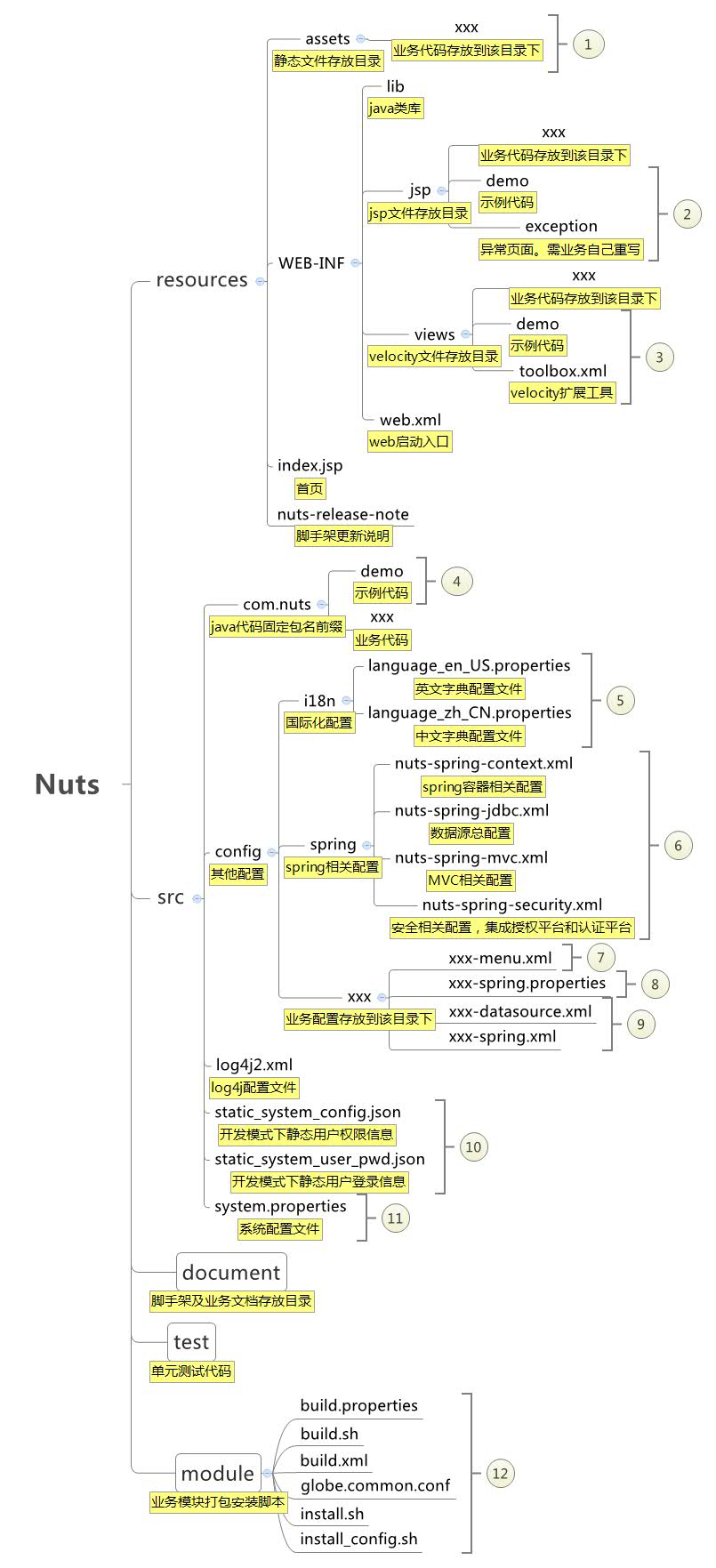
## 开发环境要求

* JDK1.8+
* Tomcat 8.0+
* Mysql 5.7+

## 本地搭建步骤

* 第一步：从发布网站下载我们的Nuts脚手架；
* 第二步：解压后得到项目相关文件：src，resources, document；
* 第三步：使用eclipse或者idea基于此目录结构构件web项目；
* 第四步：修改[system.properties](#_部署环境配置_1)文件，配置[数据源](#_数据源配置)
* 第五步：将project部署到tomcat中并启动tomcat；
* 第六步：使用浏览器访问IP:port/contextpath/即可进入首页；

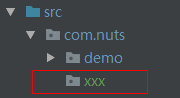
# 目录结构



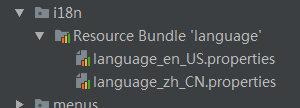
目录说明：

**为方便说明，下文出现的xxx代指业务模块名，开发时将xxx改为自己的模块名。**

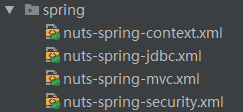
1. resources/assets存放静态资源，包括js、css、image、font以及第三方库等文件。具体规范参照前端规范。
2. resources/WEB-INF/jsp/为JSP视图解析的根目录。业务模块的jsp文件放在resources/WEB-INF/jsp/xxx/目录下（xxx表示业务模块名）。
3. resources/WEB-INF/views/为Velocity视图解析的根目录。业务模块的vm模板文件放在resources/WEB-INF/views/xxx/目录下（xxx表示业务模块名）。
4. src用于存放java代码，及插件包配置信息。java代码的包名统一以com.nuts开头，后跟项业务模块名，即com.nuts.业务模块名。如下所示，xxx表示业务模块名，demo为示例代码。



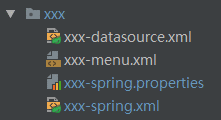
5、src/config/i18n/目录用于存放国际化配置信息。本框架中列举了两种语言的配置，如有其它语言的配置，请增加配置文件，并修改文件名为language\_{locale}.properties, locale表示本地当前语言代码。



6、src/config/spring目录用于存放所有的Spring配置信息。目前系统集成了spring-mvc，spring-jdbc，spring-security等模块。



7、业务配置统一放到config/xxx/目录下，其中xxx表示业务模块名，菜单、数据源配置必须放在该目录下。



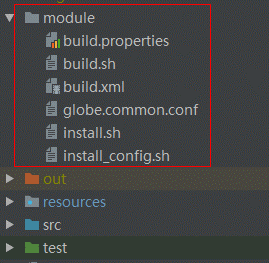
8、xxx-menu.xml是xxx业务菜单配置文件，文件名必须以-menu.xml结尾（具体配置请[《配置菜单》](#_配置菜单)）；xxx-datasource.xml是xxx业务数据源配置文件，文件名必须以-datasource.xml结尾。

9、xxx-spring.properties文件用于存放xxx业务常用配置项。xxx-spring.xml为业务spring相关配置。

10、static\_system\_config.json及static\_system\_user\_pwd.json存放的是假数据，用于开发环境中模拟获取用户登录信息及用户权限等。可通过system.properties中的ip.enable配置项来配置是否开启开发模式。ip.enable=true，表示发布模式，会从真实的鉴权平台获取用户权限数据；ip.enable=false表示开发模式，会从以上两个假数据的文件中获取用户相关数据。

11、system.properties用于存放系统常用的配置信息。所有常用的配置，尽量放在src/system.properties文件中，方便后期统一配置。具体配置请参照[《部署环境配置》](#_部署环境配置)。

12、module目录，是为业务提供的打包及安装相关的脚本及配置，如下图：



各文件作用如下：

1. build.properties 打包配置文件，如包名、版本号等
2. build.sh 在服务器上打包的执行程序
3. build.xml 在windows本地通过ant打包的脚本
4. globe.common.conf 安装前的配置文件，如脚手架对接BDP的配置，业务自己的配置
5. install.sh 在服务器上安装的执行程序
6. install\_config.sh 业务配置的相关的程序，在install.sh中会调用该程序

打包及安装说明见下文

# 技术栈

* **Spring/SpringMVC**

Spring作为整个项目的骨架，其他所有的框架都以此作为入口，Spring相关的学习资料，请通过网络进行学习。

* **Spring-JDBC**

Spring框架中的一个模块，对jdbc api做了封装，简化了jdbc的代码（比如，不用考虑获取连接与关闭连接）

* **druid**

Nuts平台使用的连接池为alibaba的druid，不是为了获得更好的性能，而是为了便于做相关的sql性能监控支持

* **Velocity**

提供Web系统的界面展示支持，实际上，velocity在Nuts平台中仅仅临时充当了html的功能，即：后续可以非常简单地切换为前后端分离的开发模式。因此您在开发过程中，只需要掌握最简单的vm语法即可。

* **slf4j**

要求使用此日志接口代替commons-logging(虽然Spring使用的还是commons-logging)，因为此日志接口拥有更好的动态参数支持。

* **log4j2**

slf4j是日志门面类，而具体的实现类为log4j，需要了解log4j的配置相关参数。

* **CAS**

CAS 是一个单点登录框架。单点登录，Single Sign On（SSO），通俗理解为一个应用登录了，其他被授权的应用不用再登录。

* **Spring-Security**

Spring Security是能够为J2EE项目提供综合性的安全访问控制解决方案的安全框架。它依赖于Servlet过滤器，这些过滤器拦截进入系统的请求，并且在应用程序处理该请求之前进行某些安全处理。

# 接口说明

框架基础接口全部整合在com.nuts.framework.PlatformAPI.java中，主要分为如下几种类型：

## 菜单相关

/\*\*  
 \* 验证当前用户是否有某个按钮的权限  
 \* @param optId 菜单编号,即\*-menu.xml中的menu节点的optId属性值  
 \* @param IOptMethod 按钮英文名称，即\*-menu.xml中的button节点的optMethod属性值  
 \* @return 是否有权限 true有权限 false 没权限  
 \*/  
**Boolean** checkUserOptPower(**String** optId, **String** IOptMethod)   
  
/\*\*  
 \* 获取当前用户的所有菜单.默认以system.properties中读取的ip.topoptid为顶级菜单，区分大小写  
 \* @return OptInfo列表  
 \*/  
List<? **extends** IOptInfo> getCurrentUserOpts()  
  
/\*\*  
 \* 获取某个菜单下的所有按钮信息  
 \* @param optId 菜单编号  
 \* @return OptMethod列表  
 \*/  
List<? **extends** IOptMethod> getOptMethodByOptID(**String** optId)   
  
/\*\*  
 \* 获取当前用户所有的按钮权限  
 \* @return 返回所有有权限的按钮编号，String的内容格式为 optId-optMethod,即“菜单编号-按钮英文名”  
 \*/  
Set<**String**> getUserAllOptPowers()

## 用户相关

/\*\*  
 \* 获取所有用户信息  
 \* @return UserInfo集合  
 \*/  
List<? **extends** IUserInfo> listAllUsers()  
  
/\*\*  
 \* 通过部门编号获取该部门中所有的用户信息  
 \* @param unitCode 部门编号  
 \* @return UserInfo集合  
 \*/  
List<IUserInfo> getSortedUnitUsers(**String** unitCode)   
  
/\*\*  
 \* 通过角色编号获取相关用户信息  
 \* @param roleCode 角色编号  
 \* @return UserInfo集合  
 \*/  
List<? **extends** IUserInfo> getUsersByRoleCode(**String** roleCode)   
  
/\*\*  
 \* 通过用户编号获取用户详情信息  
 \* @param userCode 用户编号  
 \* @return UserInfo详情  
 \*/  
IUserInfo getUserInfoByCode(**String** userCode)   
/\*\*  
 \* 通过用户登录名获取用户详情信息  
 \* @param loginName 用户登录名  
 \* @return UserInfo详情  
 \*/  
IUserInfo getUserInfoByLoginName(**String** loginName)   
  
/\*\*  
 \* 获取当前用户设置  
 \* @param paramCode 配置项的键  
 \* @return 配置项的值  
 \*/  
**String** getUserSettingValue(**String** paramCode)   
  
/\*\*  
 \* 获取当前用户的所有用户设置信息  
 \* @return 返回用户配置的键-值对  
 \*/  
Map<**String**, **String**> getUserAllSettings()

/\*\*  
 \* 获取当前登录用户信息  
 \* @return 用户信息详情  
 \*/  
IUserInfo getCurrentLoginUser()

/\*\*  
 \* 修改用户密码  
 \* @oldPassword 旧密码  
 \* @newPassword 新密码  
 \* @exception **UserInfoException** 用户异常信息  
 \*/  
changeUserPassword(**String** oldPassword,**String** newPassword) **throws UserInfoException**

## 角色相关

/\*\*  
 \* 获取所有角色  
 \* @return RoleInfo集合  
 \*/  
Map<**String**, ? **extends** IRoleInfo> getRoleRepo()  
  
/\*\*  
 \* 通过用户编号获取用户的所有角色信息  
 \* @param userCode 用户编号  
 \* @return RoleInfo详情，即用户角色关联信息  
 \*/  
List<? **extends** IRoleInfo> getRolesByUserCode(**String** userCode)   
  
/\*\*  
 \* 通过部门编号获取角色信息  
 \* @param unitCode 部门编号  
 \* @return RoleInfo集合  
 \*/  
List<? **extends** IRoleInfo> listUnitRolesByUnitCode(**String** unitCode)   
  
/\*\*  
 \* 获取角色详情  
 \* @param roleCode 角色编号  
 \* @return RoleInfo详情  
 \*/  
IRoleInfo getRoleByRoleCode(**String** roleCode)   
  
/\*\*  
 \* 通过部门编号获取角色部门关联信息  
 \* @param unitCode 部门编号  
 \* @return RoleInfo集合  
 \*/  
List<IRoleInfo> getUnitRoleinfoList(**String** unitCode)

## 部门相关

/\*\*  
 \* 通过用户编号获取所在的部门信息  
 \* @param userCode  
 \* @return UnitInfo集合  
 \*/  
Set<IUnitInfo> getUserUnits(**String** userCode)   
  
/\*\*  
 \* 通过角色编号获取部门信息  
 \* @param roleCode 角色编号  
 \* @return UnitInfo集合  
 \*/  
List<? **extends** IUnitInfo> listRoleUnitByRoleCode(**String** roleCode)   
  
/\*\*  
 \* 通过部门编号获取部门信息  
 \* @param unitCode 部门编号  
 \* @return UnitInfo详情  
 \*/  
IUnitInfo getUnitInfoByCode(**String** unitCode)   
  
/\*\*  
 \* 获取子部门信息，不会递归获取子部门的子部门信息  
 \* @param unitCode 部门编号  
 \* @return UnitInfo集合  
 \*/  
List<IUnitInfo> getSubUnits(**String** unitCode)

/\*\*  
 \* 递归获取所有的子部门  
 \* @param unitCode 部门编号  
 \* @return UnitInfo集合  
 \*/  
List<IUnitInfo> getAllSubUnits(**String** unitCode)

## 系统相关

/\*\*  
 \* 获取System.properties文件属性值  
 \* @param key 键  
 \* @return 值  
 \*/  
**String** getSysConfigValue(**String** key)   
  
/\*\*  
 \* 通过前缀获取所有属性信息  
 \* @param prefix key的前缀  
 \* @return 多个值  
 \*/  
Map<**String**, **Object**> getSysConfigByPrefix(**String** prefix)

/\*\*  
 \* 获取字典表数据  
 \* @param sCatalog  
 \* @param sKey  
 \* @return  
 \*/  
**public static** IDataDictionary getDataPiece(**String** sCatalog, **String** sKey)

## 退出登录

退出登录可通过从客户端向本系统发http请求，如下：

<http://xx.xx.xx.xx:8080/xxx/logout>

退出登录后，页面后自动跳转到在system.properties中配置的local.home地址。

## 操作日志记录

脚手架内置的java接口依赖ActiveMQ，如业务需要自己的日志接口实现，可自定义日志记录接口的实现。

### 脚手架内置日志接口

业务调用日志接口后，脚手架中的日志客户端会将日志内容推送到ActiveMQ服务器，日志服务端与ActiveMQ对接获取日志详情，以实现日志客户端与日志服务端的解耦。因此，在使用此方式开发时，业务开发者需要在system.properties文件中配置ActiveMQ的相关配置，如下：

#消息是否发送到MQ,默认打印到控制台.如果为true 需先打开MQ相关配置  
**message.mq.enable**=true  
  
#ActiveMQ 相关配置，如需要使用MQ，请将此处打开;否则不要打开。  
**activemq.url**=tcp://172.16.115.210:61616  
**activemq.username**=admin  
**activemq.password**=admin  
**activemq.cachesize**=10

若**message.mq.enable**的配置项写成false，则只会将日志打印到控制台，不会往ActiveMQ推送。本地开发时，可以暂时将此配置改成false；部署时，改成true。

日志接口为：

LogAPI.writeLog(request, userId, moduleName, operateType, detail, caseId, meanId, remark, operateTime, operateUrl, condition)

详细开发说明如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 是否必填 | 描述 |
| request | HttpServletRequest | 是 | 请求，用于获取IP/Mac,用户ID，操作URL |
| userId | Long | 是 | 用户ID |
| moduleName | String | 是 | 模块名 |
| operateType | String | 是 | 操作类型，从OperateType中取 |
| detail | String | 是 | 日志详细内容 |
| caseId | String | 否 | 案件ID |
| meanId | String | 否 | 手段ID |
| ip | String | 否 | 客户机IP |
| mac | String | 否 | 客户机Mac |
| remark | String | 否 | 备注 |
| operateTime | Long | 否 | 操作时间 |
| operateUrl | String | 否 | 操作URL |
| condition | List<QueryConditionBean> | 否 | 查询条件 |

参数说明：

操作类型提供了常量类：操作类型优先从常量类中找，常量类中没有的操作自己到桌面的数据库中添加（表名bdp\_log\_operate），然后传入对应的operateType。

*/\*\*  
 \* 操作类型常量  
 \** ***@author*** *zouwuyang  
 \** ***@date*** *2018/6/14  
 \*/*public interface OperateType {  
 /\*添加\*/  
 String *INSERT* = "INSERT";  
 /\*修改\*/  
 String *UPDATE* = "UPDATE";  
 /\*查询\*/  
 String *QUERY* = "QUERY";  
 /\*删除\*/  
 String *DELETE* = "DELETE";  
 /\*登录\*/  
 String *LOGIN* = "LOGIN";  
 /\*退出\*/  
 String *LOGOUT* = "LOGOUT";  
 /\*处置\*/  
 String *TREAT* = "TREAT";  
 /\*布控\*/  
 String *MONITOR* = "MONITOR";  
 /\*停控\*/  
 String *DISMONITOR* = "DISMONITOR";  
 /\*导入\*/  
 String *IMPORT* = "IMPORT";  
 /\*导出\*/  
 String *EXPORT* = "EXPORT";  
 /\*中断\*/  
 String *INTERRUPT* = "INTERRUPT";  
 /\*撤销\*/  
 String *CANCEL* = "CANCEL";  
 /\*执行\*/  
 String *EXEC* = "EXEC";  
 /\*审批\*/  
 String *APPROVAL* = "APPROVAL";  
 /\*发表评论\*/  
 String *COMMENT* = "COMMENT";  
 /\*结果反馈\*/  
 String *FEEDBACK* = "FEEDBACK";  
 /\*盖章\*/  
 String *STAMP* = "STAMP";  
 /\*浏览\*/  
 String *BROWSE* = "BROWSE";  
 /\*重新执行\*/  
 String *REDO* = "REDO";  
 /\*统计\*/  
 String *COUNT* = "COUNT";  
 /\*立项\*/  
 String *PROJECT* = "PROJECT";  
 /\*备份\*/  
 String *BACKUP* = "BACKUP";  
 /\*启用\*/  
 String *ENABLED* = "ENABLED";  
 /\*停用\*/  
 String *DISABLED* = "DISABLED";  
 /\*上传\*/  
 String *UPLOAD* = "UPLOAD";  
 /\*下载\*/  
 String *DOWNLOAD* = "DOWNLOAD";  
 /\*刷新\*/  
 String *REFRESH* = "REFRESH";  
 /\*登录失败\*/  
 String *LOGIN\_FAIL* = "LOGIN\_FAIL";  
 /\*使用业务功能\*/  
 String *USE\_FUNCTION* = "USE\_FUNCTION";  
 /\*查询命中白名单\*/  
 String *QUERY\_MATCH\_WHITELIST* = "QUERY\_MATCH\_WHITELIST";  
 /\*迁移\*/  
 String *MOVE* = "MOVE";  
 /\*备案\*/  
 String *ARCHIVECASE* = "ARCHIVECASE";  
 /\*在控\*/  
 String *CONTROL* = "CONTROL";  
 /\*移交\*/  
 String *HANDOVER* = "HANDOVER";  
 /\*合并\*/  
 String *MERGE* = "MERGE";  
 /\*结案\*/  
 String *OVERCASE* = "OVERCASE";  
 /\*批量查询\*/  
 String *BATCH\_QUERY* = "BATCH\_QUERY";  
 /\*发布\*/  
 String *RELEASE* = "RELEASE";  
 /\*共享\*/  
 String *SHARE* = "SHARE";  
}

参数中查询条件bean

*/\*\*  
 \* 查询类日志 查询条件  
 \** ***@author*** *zouwuyang  
 \** ***@date*** *2018/6/14  
 \*/*public class QueryConditionBean implements Serializable {  
  
 private static final long *serialVersionUID* = 316106841681794454L;  
  
 */\*\*  
 \* 查询的协议列表，精确到大协议，使用字典定义  
 \* 选填 长度0-4000  
 \*/* private String protocal;  
  
 */\*\*  
 \* 查询的数据源，存储字典定义  
 \* 选填 长度0-4000  
 \*/* private String datasource;  
  
 */\*\*  
 \* 关键字  
 \* 选填 长度0-1024  
 \*/* private String keyWord;  
  
 */\*\*  
 \* 关键字类型  
 \* 选填 长度0-1024  
 \*/* private String keyWordType;  
  
 */\*\*  
 \* 开始时间 毫秒数*

*\* 选填  
 \*/* private Long startTime;  
  
 */\*\*  
 \* 结束时间 毫秒数*

*\* 选填  
 \*/* private Long endTime;

### 业务自定义接口实现

如业务不想使用内置的ActiveMQ的日志实现方式，也通过如下步骤实现日志接口：

1、实现LogWriter接口，save方法里实现记录日志逻辑

public interface LogWriter {

void save(LogTransferBean logBean);

}

2、调用LogAPI的注册方法，将自己的实现类的实例注册进去

public static void resgisterLogWriter(LogWriter logWriter);

3、按照6.7.1的接口调用方式，调用日志接口

## 消息推送

消息推送依赖ActiveMQ，脚手架内置消息推送客户端，业务调用客户端接口时，客户端发将消息推送到ActiveMQ，消息服务端与ActiceMQ对接获取客户端推送的消息数据。因此，在使用此方式开发时，业务开发者需要在system.properties文件中配置ActiveMQ的相关配置，如下：

#ActiveMQ 相关配置，如需要使用MQ，请将此处打开;否则不要打开。  
**activemq.url**=tcp://172.16.115.210:61616  
**activemq.username**=admin  
**activemq.password**=admin  
**activemq.cachesize**=10

客户端接口类：NotificationCenter

1、发送预警消息接口

/\*\*  
 \* 发送预警消息  
 \* @param sender 发送人userCode  
 \* @param receiver 接口人userCode(多个人以逗号隔开)  
 \* @param msgSubject 消息主题  
 \* @param msgContent 消息内容  
 \* @param optUrl 业务回跳URL  
 \* @return "OK" 表示成功，其他的为错误信息  
 \*/  
**public String** sendWarnMessage(**String** sender, **String** receiver, **String** msgSubject, **String** msgContent, **String** optUrl)

参数：sender String 必填 发送系统名称

receiver String 必填 接收人ID，多个以逗号隔开

msgSubject String 必填(长度不超过100) 消息主题/标题

msgContent String 必填 消息内容

optUrl String 选填(可为null) 消息回跳到业务的url

返回值：成功返回”OK”,成功代表成功发送，不代表成功接收。发送方不管接收成功与否。

失败返回”失败信息”

2、发送应用推送接口

/\*\*  
 \* 发送应用推送  
 \* @param sender 发送人userCode  
 \* @param receiver 接口人userCode(多个人以逗号隔开)  
 \* @param msgSubject 消息主题  
 \* @param msgContent 消息内容  
 \* @param optUrl 业务回跳URL  
 \* @return "OK" 表示成功，其他的为错误信息  
 \*/  
**public String** sendAppMessage(**String** sender, **String** receiver, **String** msgSubject, **String** msgContent, **String** optUrl)

参数：sender String 必填 发送系统名称

receiver String 必填 接收人ID，多个以逗号隔开

msgSubject String 必填(长度不超过100) 消息主题/标题

msgContent String 必填 消息内容

optUrl String 选填(可为null) 消息回跳到业务的url

返回值：成功返回”OK”,成功代表成功发送，不代表成功接收。发送方不管接收成功与否。

失败返回”失败信息”

单元测试示例

/\*\*  
 \* 消息客户端接口单元测试  
 \*/  
@RunWith(**SpringJUnit4ClassRunner**.**class**)  
@WebAppConfiguration  
@ContextConfiguration(  
classes = {**ActiveMQConfig**.**class**, **SpringConfig**.**class**, **MessageClientConfig**.**class**})  
**public class MessageTest** {  
 @Autowired  
 **private** NotificationCenter notificationCenter;  
  
 @Before  
 **public void** init(){  
 **MockServletContext** sc = **new** MockServletContext("");  
 sc.addInitParameter(**ContextLoader**.CONFIG\_LOCATION\_PARAM, "/config/spring/nuts-spring-context.xml");  
 ServletContextListener listener = **new** ContextLoaderListener();  
 **ServletContextEvent** event = **new** ServletContextEvent(sc);  
 listener.contextInitialized(event);  
 }  
  
 @Test  
 **public void** testSendConsole(){  
 **System**.out.println(notificationCenter.sendAppMessage("1", "-1", "test", "testContent", **null**));  
 }  
}

# 本地开发环境配置

系统发布的配置项尽量提取到system.properties文件中，以便于后期做统一配置。system. properties配置信息主要分为以下几类：

## 鉴权平台配置（BDP-AUTH）

鉴权平台提供用户、角色、机构、岗位、职务、功能菜单、系统配置等基础功能。本框架与鉴权平台集成后，可以通过PlatformAPI接口获得用户、菜单、部门、权限等相关数据。因框架通过http通过请求获取鉴权接口数据，因此需要配置鉴权平台的访问地址，如下所示：

#是否从鉴权平台获取数据，如果为false，会从static\_system\_user\_pwd.json文件中获取数据.开发环境可以将此项置为false。发布时一定要置为true  
**ip.enable**=true

#鉴权平台访问地址  
**ip.home** = http://172.16.5.106:8080/bdp-auth

## 认证平台配置（BDP-CAS）

CAS是耶鲁大学的一个开源的企业级单点登录系统，本框架在此基础上内置了：密码、PKI、指纹、短信、IC卡等常用的登录方式；集成该功能后，可以实现系统的安全验证功能。集成该功能需要配置如下信息：

#CAS访问路径前缀  
**cas.home**=http://172.16.5.107:8082/centit-cas  
  
#工程唯一标识，保证注册到CAS上的工程标识唯一  
**app.key** = demo  
  
#当前tomcat的访问局域网地址，不要填localhost或127.0.0.1。当前系统作为CAS的客户端，登录过程中CAS会与当前系统通信  
**local.home** = http://10.0.200.44:8080/demo  
  
#是否使用cas的登录页面。1.0.12版本以上的脚手架开始支持该配置项，详见自定义登录页说明  
**cas.login.support**=true  
  
#系统登录页面。如果cas.login.support为true，该项不生效；否则，配本地的登录页面请求路径  
#local.login.url=/login/page

## 当前系统配置

当前系统配置主要指基于本框架开发的业务配置项。空框架中包含了ip.topoptid配置项，该配置项在获取菜单时起作用，当调用PlatformAPI获取菜单接口时，将会以以下的配置作为顶级菜单返回有权限的菜单数据。

#顶级菜单编号

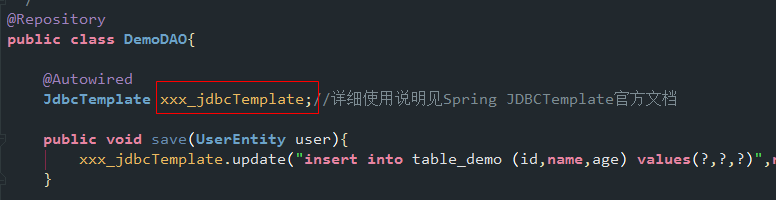
**ip.topoptid** = demo

## 数据源配置

数据源配置主要包括url、username、password等配置。如果需要多数据源，可参考xxx-datasource.xml在config/xxx/自己模块配置目录下新建 xxxx-datasource.xml 配置文件；为防止多数据源的情况下数据源冲突，Bean的id必须以模块名做为前缀。如下(下图中的xxx表示模块名)：

<!-- 数据源配置 -->  
<**bean** id="xxx\_datasource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">  
 <**property** name="url" value="jdbc:mysql://172.16.2.170:3306/xxx?useUnicode=true&amp;characterEncoding=UTF-8" />  
 <**property** name="username" value="xxxx" />  
 <**property** name="password" value="xxxx" />  
  
 <!--配置初始化大小、最小、最大-->  
 <**property** name="initialSize" value="2" />  
 <**property** name="minIdle" value="20" />  
 <**property** name="maxActive" value="100" />  
  
 <!--配置获取连接等待超时的时间-->  
 <**property** name="maxWait" value="60000" />  
  
 <!--配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒-->  
 <**property** name="timeBetweenEvictionRunsMillis" value="60000" />  
  
 <!--配置一个连接在池中最小生存的时间，单位是毫秒-->  
 <**property** name="minEvictableIdleTimeMillis" value="300000" />  
 <**property** name="validationQuery" value="SELECT 'x'" />  
 <**property** name="testWhileIdle" value="true" />  
 <**property** name="testOnBorrow" value="false" />  
 <**property** name="testOnReturn" value="false" />  
  
 <!--配置监控统计拦截的filters，去掉后监控界面sql无法统计-->  
 <**property** name="filters" value="stat" />  
</**bean**>  
  
<!-- JDBCTemplate -->  
<**bean** id="xxx\_jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">  
 <**constructor-arg** name="dataSource" ref="xxx\_datasource"/>  
</**bean**>

java DAO中使用方式如下：



## 配置菜单

本框架提供自动注册菜单的功能，各业务需将菜单信息配置在config/xxx/目录下，系统启动时扫描该目录下的配置文件，并将菜单及按钮信息注册到BDP-AUTH授权平台中。系统启动后，需到授权平台给用户分配菜单及按钮权限。菜单具体配置信息如下：

<!--  
 ##属性说明  
 menu 菜单配置项  
 optId 菜单编号  
 optName 菜单中文名称  
 optUrl 菜单路径  
 isInToolbar 是否显示,填Y或N。【可为空，默认显示】

oderInd 排序字段

button 按钮配置项，如果没有，可为空

optName 按钮中文名称  
 optMethod 英文名称，可用于判断是否有该按钮的权限  
 optDesc 按钮描述信息，【可为空】  
 optUrl ~~请求路径。该路径与菜单的optUrl组合成一个完整的url~~页面路径，目录不用配置该项  
 optReq 请求方式。填C、U、R、D。分别表示POST、PUT、GET、DELETE。【可为空，默认不区分】

isPublic 是否是公共按钮

icon 图标名称

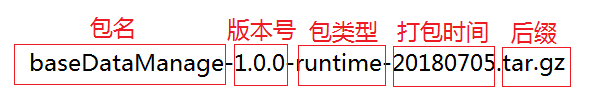
-->  
  
<**config**>  
 <!--父级系统菜单编号，可为空。如果不为空，会把本系统的所有菜单挂到该菜单下  
 （适用场景：将本系统的菜单全部挂载到其他某个系统中）；如果为空，会将menus里的最外层菜单作为顶级菜单。-->  
 <**parent**>xxxx</**parent**>  
  
 <**menus**>  
 <!--当前系统的顶级菜单，optid必须要与system.properties中的ip.topoptid配置项内容保持一致。否则在当前系统中调用获取菜单接口时，取不到数据-->  
 <**menu** optId="demo0" optName="演示菜单" optUrl="/mofang/user/" isInToolbar="Y" oderInd="1">  
 <**menu** optId="demo1" optName="演示子菜单1" optUrl="/meeting/my/" isInToolbar="Y" oderInd="1" icon="hedadsfdf">  
 <**button** optName="预定" optMethod="add" optDesc="预定会议室" optUrl="/meeting/room" optReq="C"/>  
 <**button** optName="释放" optMethod="delete" optDesc="取消预定" optUrl="/meeting/room" optReq="D"/>  
 <**button** optName="修改" optMethod="update" optDesc="修改预定的会议室" optUrl="/meeting/room" optReq="U"/>  
 <**button** optName="查询" optMethod="query" optDesc="查询列表" optUrl="/meeting/room" optReq="R"/>  
 </**menu**>  
 <**menu** optId="demo2" optName="演示子菜单2" optUrl="/notice/center/" isInToolbar="Y" oderInd="2">  
 </**menu**>  
 <**menu** optId="demo3" optName="演示子菜单3" optUrl="/my/apply/" isInToolbar="N" oderInd="3">  
 </**menu**>  
 </**menu**>  
 </**menus**>  
</**config**>

注:

1. 本程序仅支持新增菜单和修改菜单、按钮非主键的操作，如需删除菜单或修改菜单optId、按钮optMethod，需要手动到鉴权平台修改。
2. 为防止多次重复刷菜单，菜单配置文件解析后会被重命名成xxx-menu.xml\_bak文件。如需重复执行，请注意修改classpath:/config/xxx/下的文件名（其中xxx表示模块名）。
3. <parent>节点可为空。如果有值，必须存在于鉴权平台已有菜单。否则，所有菜单均不会显示在鉴权平台的菜单列表中。
4. <parent>节点如果有值，系统会自动给菜单和按钮的optUrl加上ip、端口、工程名等前缀，这些值会从system.properties文件中的local.home中的配置项中读取。

# 打包

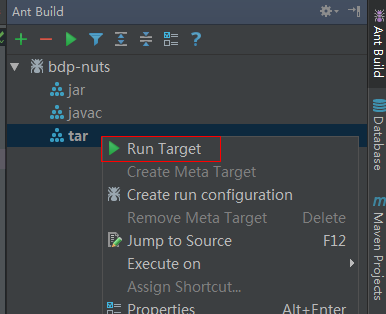
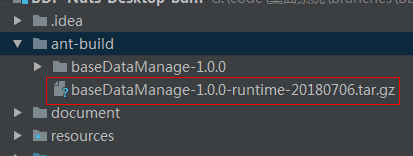
脚手架提供基础的打包程序，业务开发者可在此基础上制作自己业务需要的打包程序（如无特殊打包需求，只需修改build.properties中的配置即可，其他文件保持原样）。打包相关文件有build.properties,build.sh,build.xml。通过打包程序生成的安装包包名结构如下：



1. build.properties文件为打包的配置文件，可配置包名、版本号等信息。如无特殊需求，仅需修改该文件中的package.name和package.version即可。

##打包配置，服务器上，通过执行sh build.sh执行打包，本地通过Ant，执行build.xml中的tar命令打包  
  
#包名  
**package.name**=xxx  
#版本号  
**package.version**=1.0.0  
#打包临时目录，不用修改  
**package.build**=ant-build  
#系统类型，不用修改  
**os.type**=allsystem

1. build.xml为本地，可以在idea或eclipse中通过Aut Build插件的run target执行”tar”命令来打安装包，打包完成后，在ant-build目录下，可以看到以.tar.gz结尾的业务安装包。如：

1. build.sh为服务器打包脚本，可用于jekens打包

# 安装

## 基础运行环境安装

业务系统依赖的tomcat、jdk等，详见如下文档：

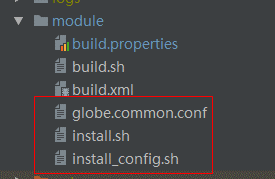


南京内网包获取路径：<http://172.16.115.185:81/build/BDP/03_BDP-Nuts/dependencies/>

## 业务系统安装

本文档仅用于告知开发者如何制作基于脚手架开发的业务系统安装脚本，提供给测试及工程的《开发维护手册》需业务开发者自行提供。

### 文件说明



安装脚本有三个文件，其中：

globe.common.conf为安装前的配置文件

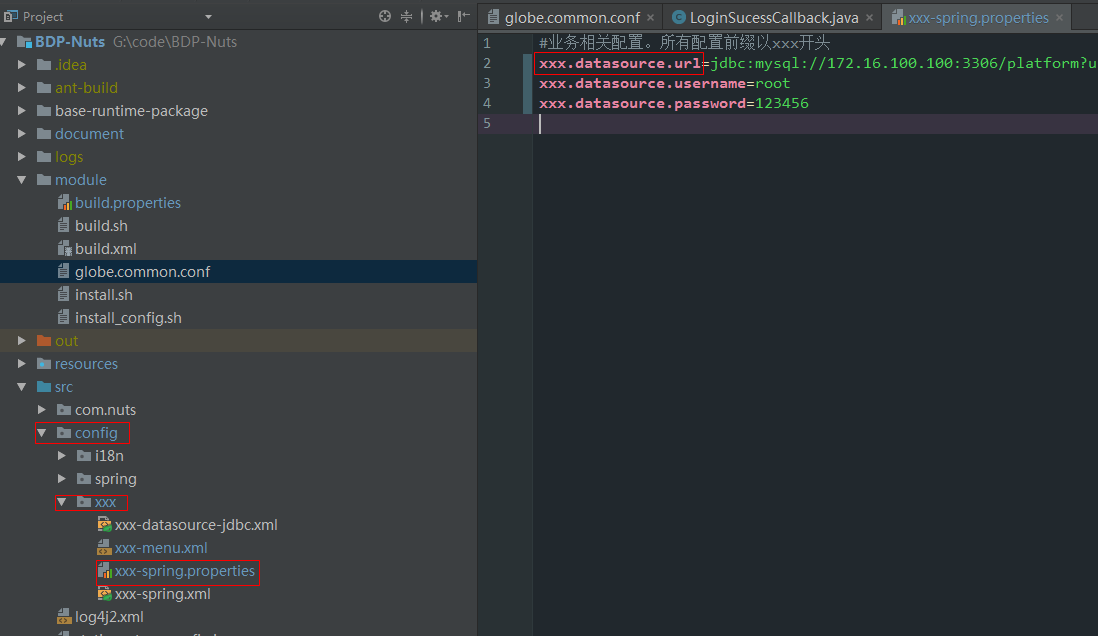
install.sh为安装执行脚本

install\_config.sh为业务系统的配置脚本

一般情况下，业务只需在install\_config.sh文件中添加自己的配置脚本，并在globe.common.conf中加上相关配置项即可，install.sh不需要修改。如有特殊需求，业务开发者也可根据自己的需要修改install.sh。

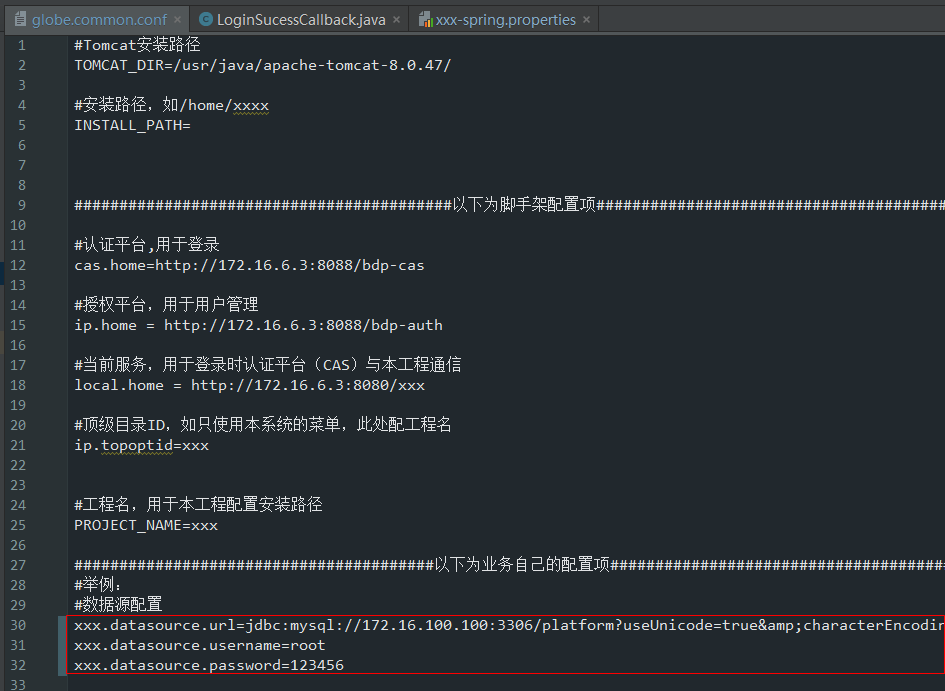
### 安装配置脚本制作

安装配置脚本制作只需两步，以修改数据源为例，在安装前需要修改config/xxx/xxx-spring.properties文件中的几个配置项的值：

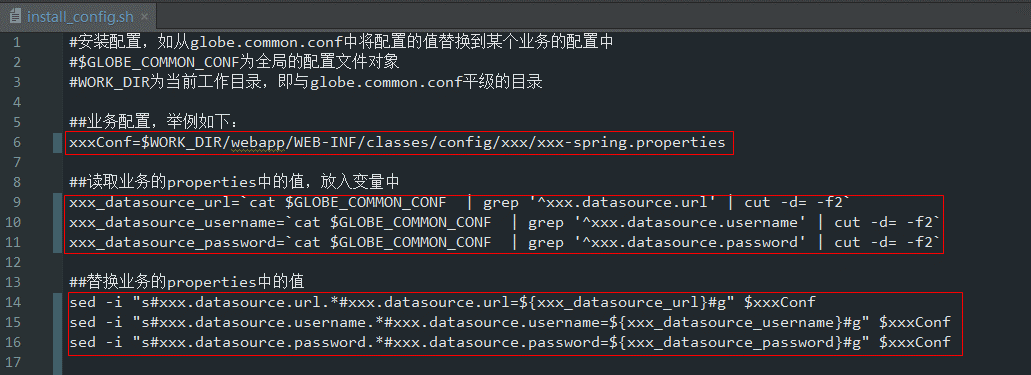


需要开发者在安装配置中做如下修改：

**第一步：**将相关配置项放入globe.common.conf文件中，添加业务配置项后的globe.common.conf文件如下：



**第二步：**编辑install\_config.sh，加入如下脚本（脚本内容不限制，业务开发者可根据自己的需求来实现配置的读取与替换）



以上两步即可完成安装脚本的制作，在安装执行sh install.sh时，会自动加载install\_config.sh脚本

### 安装步骤

安装业务系统之前，确保基础运行环境中已正确安装tomcat和jdk

1. 将打包的tar包放入业务的安装目录下，如/home/xxx，此处已基础数据的安装包举例
2. 执行“tar -xzvf 包名”解压，当前目录中会生成解压后的目录
3. 进入解压后的目录，编辑globe.common.conf文件，修改配置项
4. 执行“sh install.sh”命令安装，安装完成后会自动重启tomcat

### 安装检测

正确安装完成后，tomcat的conf/Catalina/localhost/目录下会生成业务的tomcat部署文件



path中的xxx表示工程名，docBase为工程部署的实际路径

如tomcat启动失败，请开发者根据tomcat中的logs/catalina.out中的日志来排查启动问题。

# 常用功能示例

本框架已集成了如下功能，并内置了几个常用功能的简单demo，系统启动后会自动进入到demo页面。

## 上传下载

本框架demo中提供了两种上传和两种下载的方法，开发时可根据实际场景选择不同的方式开发。

以一种上传方式为例，前端表单如下：

<**form** action="**<%=**contextPath**%>**/file/upload" enctype="multipart/form-data" method="post">  
<**table**>  
 <**tr**>  
 <**td**>文件描述:</**td**>  
 <**td**><**input** type="text" name="description"></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>请选择文件:</**td**>  
 <**td**><**input** type="file" name="files"></**td**>  
 <**td**><**input** type="file" name="files"></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**><**input** type="submit" value="上传"></**td**>  
 </**tr**>  
</**table**>  
</**form**>

后端controller处理如下：

/\*\*  
 \* 文件上传功能。  
 \* 方式一：通过二进制保存文件到服务器  
 \* 上传文件会自动绑定到MultipartFile中，将上传的文件通过二进制保存到web服务器上去  
 \*/  
@RequestMapping(value = "/upload", method = RequestMethod.POST)  
@ResponseBody  
**public Response** upload(HttpServletRequest request, @RequestParam("description") **String** description, @RequestParam("files") MultipartFile[] files){  
 //上传文件路径  
 **String** path = request.getServletContext().getRealPath("/WEB-INF/upload/");  
 **if**(files != **null** && files.length > 0){  
 **for**(MultipartFile multipartFile : files){  
 //获取上传文件的原名  
 **String** originalFilename = multipartFile.getOriginalFilename();  
 //目标存储文件  
 **File** destFile = **new** File(path, originalFilename);  
 //判断路径是否存在，如果不存在就创建一个  
 **if**(!destFile.getParentFile().exists()){  
 destFile.getParentFile().mkdirs();  
 }  
 **if**(!destFile.exists()){  
 //将上传文件保存到一个目标文件当中  
 **try**{  
 multipartFile.transferTo(**new** File(path + **File**.separator + originalFilename));  
 }**catch**(**IOException** e){  
 //文件存储失败  
 e.printStackTrace();  
 }  
 }**else**{  
 //已存在同名文件  
 }  
 }  
 }**else**{  
 //上传的文件为空  
 }  
 **return Response**.ok();  
}

可参照java代码com.nuts.demo.upload.\*，jsp代码/WEB-INF/jsp/demo/uploadFile.jsp

## 增删改查

本框架提供了简单的增删改查的demo，如下：

以新增操作为例，前端表单以post方式提交数据：

<**form** action="**<%=**contextPath**%>**/users/" method="post">  
 <**input** type="text" name="id" placeholder="ID" value=""/>  
 <**input** type="text" name="username" placeholder="用户名" value=""/>  
 <**input** type="text" name="age" placeholder="年龄" value=""/>  
 <**button** type="submit">提交</**button**>  
</**form**>

Controller层以post的方式接收表单数据：

@Controller  
@RequestMapping("/users/")  
**public class CurdController**{  
  
 Logger logger = **LoggerFactory**.getLogger(**this**.getClass());  
  
 @Autowired  
 **DemoServiceImpl** demoService;  
  
 @PostMapping  
 **public ModelAndView** add(@ModelAttribute **UserEntity** user){  
 logger.info("进入了add方法："+ JSON.toJSONString(user));  
 demoService.save(user);  
 **ModelAndView** modelAndView = **new** ModelAndView("demos/curd");  
 **return** modelAndView;  
 }

}

Service层处理

@Service  
**public class DemoServiceImpl implements** IDemoService{  
  
 @Autowired  
 **private DemoDAO** demoDAO;  
  
 /\*\*  
 \* 添加  
 \* @param user 用户信息  
 \*/  
 **public void** save(**UserEntity** user){  
 //验证参数合法性  
  
 //执行业务逻辑  
  
 //调用DAO  
 demoDAO.save(user);  
  
 //处理返回结果  
 }

}

DAO层执行操作数据

@Repository  
**public class DemoDAO**{  
  
 @Autowired  
 **JdbcTemplate** jdbcTemplate;//详细使用说明见Spring JDBCTemplate官方文档  
  
 **public void** save(**UserEntity** user){  
 jdbcTemplate.update("insert into table\_demo (id,name,age) values(?,?,?)",**new** Object[]{user.getId(),user.getUsername(),user.getAge()});  
 }

}

删除、修改、查询可参照java代码com.nuts.demo.curd.\*，jsp代码/WEB-INF/jsp/demo/curd.jsp

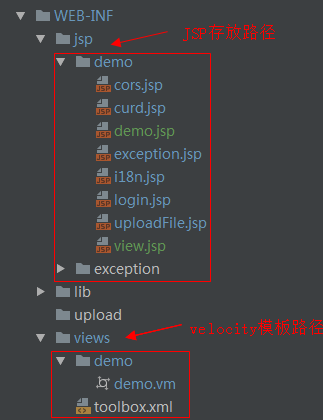
## 国际化

本框架提供了简单的国际化的demo

可参照java代码com.nuts. demo.i18n.\*，jsp代码/WEB-INF/jsp/demo/i18n.jsp

## 视图解析

本框架提供两种视图解析方式：Veloctity、JSP，默认以Velocity解析器优先，如匹配不到velocity模板，才会解析JSP。其中，velocity模板存放目录必须在/WEB-INF/views/下；JSP文件必须在/WEB-INF/jsp/目录下，否则会找不到相应的文件。如下图：



Controller中的示例代码如下，如果存在同名视图，系统只会匹配.vm文件：

@GetMapping("vm")  
**public String** vm(){  
 logger.info("进入了VM方法");  
 **return** "demo/demo";  
}

## 自定义登录页面（1.0.12+）

1.0.12版本开始，脚手架支持自定义登录页面样式功能。业务可根据自己的需要如如下开发规范来修改页面背景样式，指定需要的登录方式。

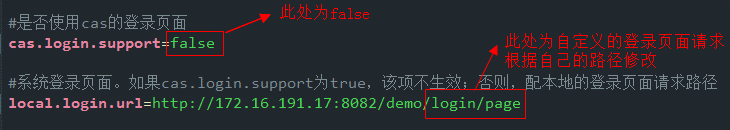
### 登录流程



业务模块定义好CAS的相关配置，在自定义登录页引入cas的js文件，传入相关参数，CAS服务会根据参数渲染相应的登录方式，之后通过iframe的形式嵌入到业务登录页面，业务调整相关样式，渲染完整登录页。

### 开发步骤

第一步：修改system.properties



1. cas.login.support 改成false（为true时表示使用cas默认的登录页）
2. local.login.url填写自定义的登录页面地址，可以指向其他系统的地址，如指向非同源系统的登录页面，需要前面加IP:PORT/CONTEXTPATH

第二步：编写自定义登录页面（其中标红的为system.properties中cas.home的配置值）

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">  
 <title>登录</title>  
 <script type="text/javascript" src="http://172.16.191.17:8088/bdp-cas/js/bdpLogin.js"></script>  
</head>  
<body>  
 <div class="content">  
 <h1>\*\*\*\*\*平台</h1>  
 </div>  
 <div id="loginPlaceHolder">  
  
 </div>  
</body>  
  
<script type="text/javascript">  
  
 // 调用CAS的渲染区块登录页方法  
 $("#loginPlaceHolder").casLogin({  
 /\*cas服务地址\*/  
 casHome:"http://172.16.191.17:8088/bdp-cas",  
 /\* 业务支持的登录方式,可选的值有："password", "finger", "pki", "pkiAgent", "icCard", "sms", "face" \*/  
 supportAuthTypes: ["password", "finger", "pki"],  
 /\* 当前默认登录方式 \*/  
 currentAuthType: "password",  
 /\* 模块名称（要保证唯一性） \*/  
 moduleName: "iquery",  
 /\* 渲染成功回调 \*/  
 success: **function**() {  
 alert("成功");  
 },  
 /\* 渲染失败回调 \*/  
 error: **function**(errorMsg) {  
 alert(errorMsg);  
 }  
 });  
</script>  
</html>

实现后的效果如下： loginPlaceHolder的div中将嵌入**登录块**



**自定义登录的后台实现，可参考脚手架jar包中的com.nuts.demo.login.LoginController类**

### 用户增删改监听

该功能用于在Auth中增删改用户信息时，通知业务做相应的用户相关逻辑处理。

开发步骤：

第一步：写一个实现类，实现IUserChangeCallback接口

**public class PlatformUserCallBackImpl implements** IUserChangeCallback{  
 @Override  
 **public void** onUpdate(**String** userid){  
 **System**.out.println("执行了update方法,userid为："+userid);  
 }  
  
 @Override  
 **public void** onDelete(**String** userid){  
 **System**.out.println("执行了delete方法,userid为："+userid);  
 }  
  
 @Override  
 **public void** onAdd(**String** userid){  
 **System**.out.println("执行了add方法,userid为："+userid);  
 }  
}

第二步：在适当的时机（如：系统启动完成后），调用注册接口

**CallbackRegisterAPI**.register(**new** PlatformUserCallBackImpl());

## 登录登出回调（1.1.1+）

### 登录成功回调

步骤一、继承com.nuts.framework.config.LoginSucessCallback,实现whenLoginSuccess方法

public class TestLoginSuccessCallBack extends LoginSucessCallback {  
  
 @Override  
 protected void whenLoginSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {  
 System.*out*.println("登录成功");  
 }  
}

步骤二、在配置文件中创建bean

<bean class="com.xxx.TestLoginSuccessCallBack" />

### 登录失败回调

步骤一、继承com.nuts.framework.config.LoginFailureCallback，实现whenLoginFailure方法

public class TestLoignFailure extends LoginFailureCallback {  
  
 @Override  
 protected void whenLoginFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception) {  
 System.*out*.println("登录失败");  
 }  
}

步骤二、在配置文件中创建bean

<bean class="com.xxx.TestLoignFailure" />

### 登出回调

步骤一、继承com.nuts.framework. NutsLogoutHandle，实现whenLogout方法

public class TestLogoutHandle extends NutsLogoutHandle {  
 @Override  
 protected void whenLogout(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Authentication authentication) {  
 System.*out*.println("logout");  
 }  
}

步骤二、在配置文件中创建bean

<bean class="com.xxx.TestLogoutHandle" />

# 常用工具

## 常用工具包

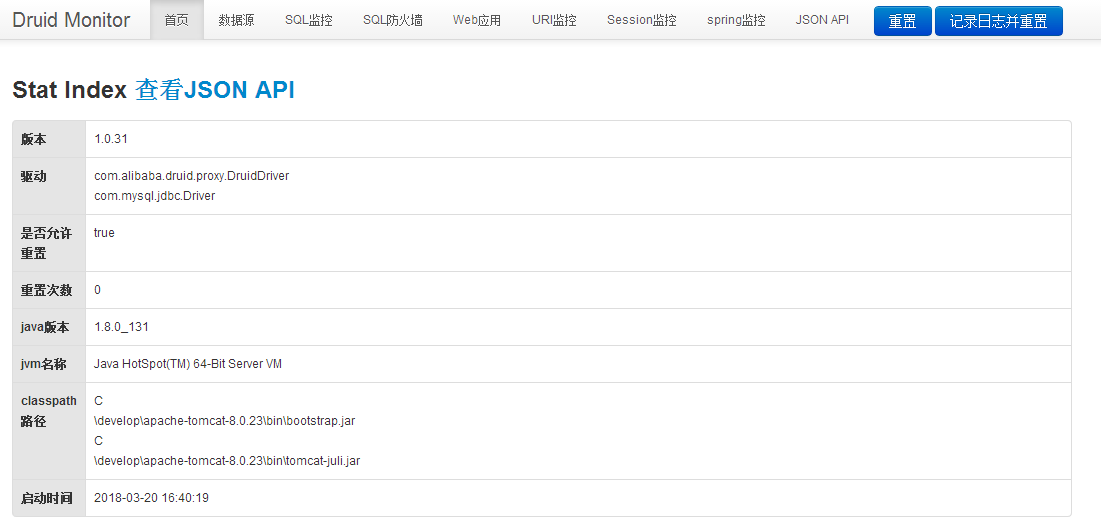
本框架中目前提供以下工具包：

**\*** Hibernate-validator 后台Bean验证工具包  
**\*** jackson/fastjson JSON解析库. 推荐使用jackson  
**\*** Apache commons-\* Apache通用工具包  
**\*** guava 类似Apache Commons工具集. guava 包含了   
**\*** jsoup HTML解析库  
**\*** dom4j XML解析库  
**\*** poi-\* Excel读写库  
**\*** quartz 定时任务库   
**\*** pinyin4j 拼音解析库  
**\*** joda-time 日期和时间库

## druid监控

本框架已集成druid连接池监控页面，可通过访问如下url查看监控页面：

<http://xx.xx.xx.xx:8080/xxxx/druid/index.html>



## 代码执行时间监控日志

为方便定位问题，可开启开发模式来定位一些问题，不需要重启tomcat。重启tomcat后会自动恢复非开发模式。

打开方式：访问 IP:PORT/工程名/config?key=debug&value=true 即可。打开开发模式后，可以看到执行日志。目前开发模式提供如下功能：

1、排查哪些service执行的时间长

可通过 IP:PORT/工程名/config?key=min\_execute\_time&value=600000 设置最小执行时间。执行时间超过value值时候，才会打印日志（如不设置，默认为1分钟）。

2、完善中…

# 常见问题

## 打包问题

暂无

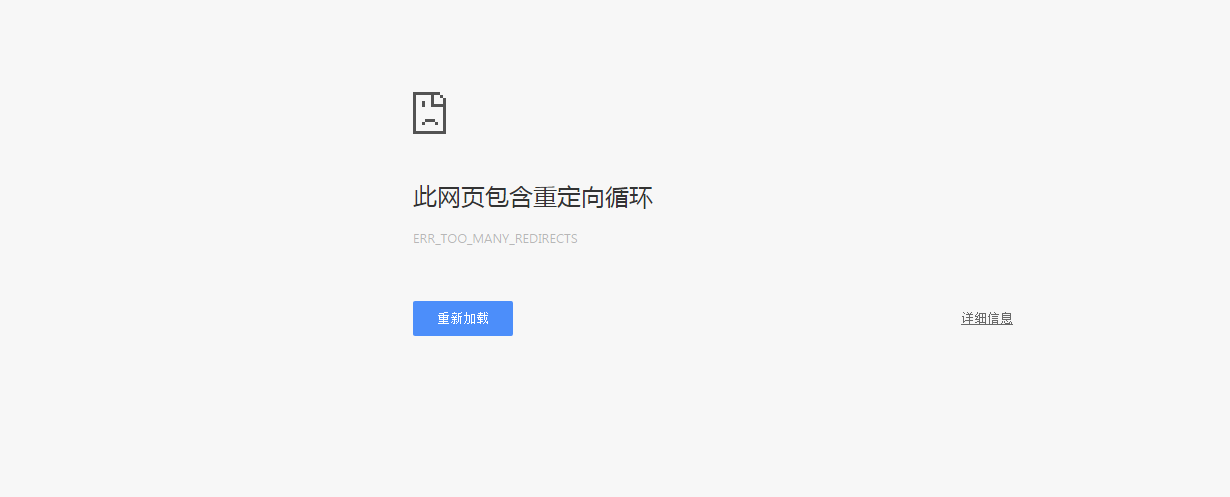
## 安装问题

暂无

## 界面问题

1、页面中出现”此网页包含重定向循环”的问题





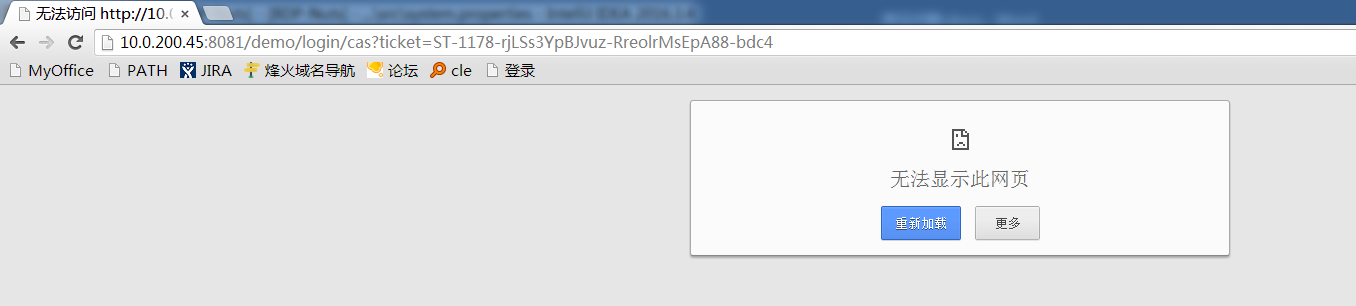
可能原因1：登录页面的url没有配置在nuts-spring-security.xml的允许放行的列表中。

2、登录页面一直显示加载中，F12显示加载js时failed



原因：cas服务配置的不正确，请在system.properties中修改cas.home的配置项

3、url中带有?ticket的参数，页面显示“无法显示此网页”



原因：system.properties中配置的local.home错误。应配置带“IP：端口号/工程名”的本机服务地址。

4、输入用户名密码登录后，页面一直显示“正在验证身份，请稍后…”，F12中cas?ticket的请求一直pending状态。



原因：system.peoperties中配置的ip.home不正确。

1. 脚手架jar包升级到1.0.12以后，session过期时，Ajax请求没有自动跳转到登录页面，F12的console中有报错。刷新页面才会跳转到登录页面。

原因：需要设置一下使用的Ajax，可在项目引用的Jquery.js最后，加上如下代码：

//session过期时，退出登录  
**var** reloadFlag = **false**;  
$.ajaxSetup({  
 complete:**function**(xhr,status){  
 **if**(401 == xhr.status && !reloadFlag){  
 reloadFlag = **true**;//防止一个页面刷新多次  
 window.location.href="logout";  
 }  
 }  
})

1. 登录完成后，进入首页报404或500，url显示如<http://172.16.5.107/thunder/>

ActiveMQ可能没有启动，导致写日志时报错

1. 登录完成后进入首页报500，url中带有ticket

可能是Auth启动有问题，或配置的Auth不正确