# 微服务应用开发规范

目录

目录

[微服务应用开发规范 1](#_Toc498330483)

[目的 2](#_Toc498330484)

[概述 2](#_Toc498330485)

[微服务系统结构 2](#_Toc498330486)

[代码规范 3](#_Toc498330487)

[有关命名 3](#_Toc498330488)

[格式规范 5](#_Toc498330489)

[缩进 5](#_Toc498330490)

[换行 5](#_Toc498330491)

[对齐 5](#_Toc498330492)

[开发目录说明 6](#_Toc498330493)

[包名规则 6](#_Toc498330494)

[文件命名规则 6](#_Toc498330495)

[系统工具类说明 7](#_Toc498330496)

[字符串工具类 7](#_Toc498330497)

[时间工具类 7](#_Toc498330498)

[获取spring的bean的工具类 7](#_Toc498330499)

[Zip包的操作 7](#_Toc498330500)

[汉字转拼音 7](#_Toc498330501)

[操作Map的工具类 8](#_Toc498330502)

[Md5加密 8](#_Toc498330503)

[json字符串操作 8](#_Toc498330504)

[http请求工具类 8](#_Toc498330505)

[文件操作类 8](#_Toc498330506)

[UUID等编号的生成 9](#_Toc498330507)

[功能模块使用 9](#_Toc498330508)

[redis的使用 9](#_Toc498330509)

[多数据源配置 9](#_Toc498330510)

[获取系统配置文件值 10](#_Toc498330511)

[多线程的使用 11](#_Toc498330512)

[通知中心发送邮件 11](#_Toc498330513)

[源码生成 12](#_Toc498330514)

## 目的

规范公司微服务的开发架构

为了使软件开发过程顺畅，保证软件质量，利于后期维护等。

## 概述

项目以功能模块来划分分工

数据库不能随意修改，本地数据库须统一

有标准的Java代码风格，有良好习惯

时刻考虑代码的可复用性

每个人需要有每周项目进度报告

## 微服务系统结构



## 代码规范

### 有关命名

尽量保证能够通过变量名得知变量的含义

**3.1.1 包命名**

包命名采用全小写命名

通过唯一域名+组件来命名

**3.1.2 类命名**

类命名采用Pascal命名法

大写字母开头，各个单词首字母大写

**3.1.3 方法命名**

方法命名采用Camel命名法

小写字母开头，各个单词首字母大写

属性的getter和setter方法尽量使用自动生成，并放在程序的后面。

Boolean类型的属性的get方法应形如isProperty()

**3.1.4 变量命名**

采用Camel命名法

小写字母开头，各个单词首字母大写

特有大写缩写词汇保持大写如：SQL

变量名字不宜过长，可适当采用缩减英文元音字母来缩短长度

假如缩短后名字重复，可以保留其中一个的部分元音字母

**3.1.5 常量命名**

采用全大写命名法

所有字母均大写

**3.1.6 页面文件命名**

采用全小写命名法

所有的字母均小写，单词之间以下划线’\_’分隔

展示页面，按照名词+描述，如：news\_list.jsp

操作页面，按照名词+动词命名，如：news\_add.jsp

按照模块从大到小命名，如：news\_order\_add.jsp

**3.1.7 资源文件命名**

采用全小写命名法

按照前缀+模块+描述+状态命名，如：btn\_main\_login\_pressed.png

(不一定完全包含如上4个部分，但是要依照顺序命名)

**3.2 有关注释**

团队成员都应该形成良好的写注释的习惯，方便以后阅读，以及为了后期生成可读性良好的Java Doc

**3.2.1 程序文件头注释**

应该包含如下：

\* 文件描述

\* 作者

\* 版本

\* 创建日期时间

\* 修改日期时间

提前设置好文件的模板Template

如以下模板：

/\*\*

\* Description:

\* Author: KevinLee

\* Version: 1.0

\* Create Date Time: ${DATE} ${TIME}.

\* Update Date Time:

\*/

**3.2.2 方法头注释**

一般在写完一个方法后使用快捷键生成一个块注释，IDE会自动帮我们写入一些信息。

应该包含如下信息：

\* 方法描述 Description:

\* 参数信息 @param

\* 返回信息 @return

\* 异常信息 @Exception

\* 参考信息（可选）@see also //指定一个类或者方法（通过类后面加#选择方法）

\* 笔记信息（可选）Note:

如以下模板：

/\*\*

\* Description: 返回一个“Hello”字符串

\* @param str 一个字符串

\* @return 返回一个字符串

\* @throws Exception 抛出一个异常

\*/

public String sayHello(String str) throws Exception{

str="Hello";

return str;

}

注意：方法里面不要使用块注释

**3.2.3 关键点注释**

应该包含如下信息：

\* 一些程序关键的地方

\* 一些程序不易读的地方

\* 编写代码过程中遇到问题的地方

\* 需要提示读者的地方

注释应该写得少而易懂

若修改了文件，可以加上修改人的信息，和修改日期。

## 格式规范

### 缩进

应注意使用format来格式化代码，使用Tab键来缩进代码，相当于4个空格。

### 换行

{}花括号应该另起一行，左花括号与方法名、类名在同一行。(除了数组初始化时的花括号)

if、while等语句，假如体内只有一句代码也不要省略{}，为了方便以后的增删

字符串过长考虑拆分成多行

if(str == null) {

}

### 对齐

{}括号等应该对齐

类和方法的块注释必须紧贴类和方法

单独起行的//注释必须对齐被注释语句

## 开发目录说明

### 包名规则

com

frame 大的模块名称

sys 子模块名称

dao 查询数据库的接口类

oracle mybatis的oracle的xml

service 处理业务逻辑的模块，这里为接口类

impl 处理业务逻辑的模块，这里为实现类

pojo 对应的数据实体，可包括扩展字段

controller 对外提供的接口

user 用户信息的相关模块

dao 查询数据库的接口类

oracle mybatis的oracle的xml

service 处理业务逻辑的模块，这里为接口类

impl 处理业务逻辑的模块，这里为实现类

pojo 对应的数据实体，可包括扩展字段

controller 对外提供的接口

### 文件命名规则

com

frame 大的模块名称

sys 子模块名称

dao 查询数据库的接口

SysDictDao.java

oracle mybatis的oracle的xml

SysDictDao.xml

service 处理业务逻辑的模块，这里为接口类

SysDictService.java

impl 处理业务逻辑的模块，这里为实现类

SysDictServiceImpl.java

pojo 对应的数据实体，可包括扩展字段

SysDict.java

controller 对外提供的接口

SysDictController.java

注意：

SysDict代表功能模块，这里一般是取表名

## 系统工具类说明

系统工具类，统一采用Frame开头

源码参考：cjhx-system项目

svn地址：https://10.201.200.27:8443/svn/msmp/trunk/src/cjhx-core/comm/cjhx-system

包路径为：com.system.comm.utils

### 字符串工具类

FrameStringUtil.java

详细内容见文件。

### 时间工具类

FrameTimeUtil.java

详细内容见文件。

### 获取spring的bean的工具类

FrameSpringBeanUtil.java

详细内容见文件。

### Zip包的操作

FrameZipUtil.java

详细内容见文件。

### 汉字转拼音

FramePinyinUtil.java

详细内容见文件。

### 操作Map的工具类

FrameMapUtil.java

详细内容见文件。

### Md5加密

FrameMd5Util.java

详细内容见文件。

### json字符串操作

FrameJsonUtil.java

详细内容见文件。

### http请求工具类

FrameHttpUtil.java

详细内容见文件。

### 文件操作类

FrameFileUtil.java

详细内容见文件。

### UUID等编号的生成

FrameNoUtil.java

详细内容见文件。

## 功能模块使用

### redis的使用

1. 配置redis：

进入spring微服务的Monitor平台，进入配置中心，增加如下的配置

|  |
| --- |
| #redis 配置 hosts多个用;分隔  redis.hosts=10.201.224.175:6379;  #暂无  redis.password=  #暂无  redis.maxIdle=500  #暂无  redis.maxTotal=1000  #暂无  redis.maxWaitMillis=100000  #暂无  redis.keyPrefix=momapi- |

2. 在项目中引入RedisConfig.java文件即可

### 多数据源配置

1. 修改加载数据源文件

编辑DataSourceConfig.java文件，增加读取数据源到DataSource

|  |
| --- |
| **public** DataSource dataSourceMom() **throws** Exception {  Properties props = **new** Properties();  props.put("driverClassName", env.getProperty("jdbc2.driverClassName"));  props.put("url", env.getProperty("jdbc2.url"));  props.put("username", env.getProperty("jdbc2.username"));  props.put("password", env.getProperty("jdbc2.password"));  **return** DruidDataSourceFactory.*createDataSource*(props);  } |

2. 增加添加到多数据源集合中

|  |
| --- |
| @Bean  @Primary  **public** DynamicDataSource dataSource() **throws** Exception {  DataSource dataSourceMom = dataSourceMom();  DataSource dataSourceAts = dataSourceAts();  DataSource dataSourceStage = dataSourceStage();  Map<Object, Object> targetDataSources = **new** HashMap<Object, Object>();  targetDataSources.put("dataSourceMom", dataSourceMom);  targetDataSources.put("dataSourceAts", dataSourceAts);  targetDataSources.put("dataSourceStage", dataSourceStage);    DynamicDataSource dataSource = **new** DynamicDataSource();  dataSource.setTargetDataSources(targetDataSources);// 该方法是AbstractRoutingDataSource的方法  dataSource.setDefaultTargetDataSource(dataSourceMom);// 默认的datasource设置为myTestDbDataSource  **return** dataSource;  } |

### 获取系统配置文件值

获取配置文件中key对应的value

|  |
| --- |
| String model = *get*(Env.***PROJECT\_MODEL***); |

Env枚举的内容

|  |
| --- |
| **public** **enum** Env {  ***PROJECT\_MODEL*** ("project.model", "项目模式[dev开发、test测试、release正式]"),  ***PRES\_JXFXBG\_PATH***("pres.jxfxbg.path", "绩效分析报告地址"),  。。。  。。。 |

### 多线程的使用

主要使用的spring提供的

2. 在项目中引入ThreadPoolConfig.java文件即可

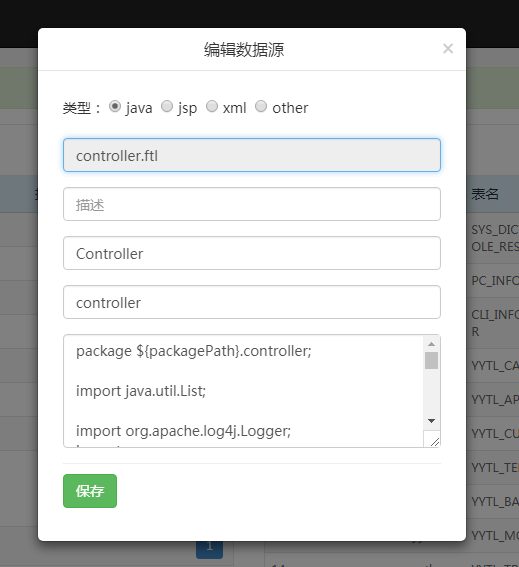
|  |
| --- |
| ThreadPoolTaskExecutor threadPool = **new** ThreadPoolTaskExecutor();  //核心线程数  threadPool.setCorePoolSize(10);  //最大线程数  threadPool.setMaxPoolSize(50);  //队列最大长度 >=mainExecutor.maxSize  threadPool.setQueueCapacity(5000);  //线程池维护线程所允许的空闲时间  threadPool.setKeepAliveSeconds(300);  //线程池对拒绝任务(无线程可用)的处理策略  threadPool.setRejectedExecutionHandler(**new** RejectedExecutionHandler() {  @Override  **public** **void** rejectedExecution(Runnable r, ThreadPoolExecutor executor) {  LOGGER.error("Begin exception handler-----------");  //执行失败任务  **new** Thread(r, "exception by pool").start();  //打印线程池的对象  LOGGER.error("The pool RejectedExecutionHandler = "+executor.toString());  }  }); |

## 源码生成

1. 进入Monitor系统的项目管理



2. 设置生成源码的对应模板



3. 选择对应表名，再点击完成，之后点击下载源码，将下载的源码解压粘贴到固定目录即可。

