infohold-logo.png

智控-平台-规范

JAVA代码示例

密 级：□ 普通 ■ 秘密 □ 机密 □ 绝密

文档编号：IH\_CP\_SPC\_001

| 版本号 | 版本说明/变更理由/变更内容 | 作者/日期 | 审批人/日期 | 备注 |
| --- | --- | --- | --- | --- |
| 0.1 | 初始创建/C | 姚家声/20150716 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

变更说明：C：Create，初始创建；A：Add，增加内容；M：Mod，修改；D：Del，删除

目录

[1. 概述 4](#_Toc425928554)

[1.1. 简介 4](#_Toc425928555)

[1.2. 目的 4](#_Toc425928556)

[1.3. 适用范围 4](#_Toc425928557)

[1.4. 文档读者 4](#_Toc425928558)

[1.5. 词汇表 4](#_Toc425928559)

[1.6. 参考资料 5](#_Toc425928560)

[2. 开发步骤 5](#_Toc425928561)

[2.1. 引入bdrp 5](#_Toc425928562)

[2.2. 包目录简介 6](#_Toc425928563)

[2.3. 创建系统配置信息 7](#_Toc425928564)

[2.4. model开发 7](#_Toc425928565)

[2.5. dao开发 8](#_Toc425928566)

[2.6. manager开发 9](#_Toc425928567)

[2.7. controller开发 9](#_Toc425928568)

[2.8. 单元测试 11](#_Toc425928569)

[3. 常见场景代码 13](#_Toc425928570)

[3.1. 一对一 13](#_Toc425928571)

[3.2. 一对多 13](#_Toc425928572)

[3.3. 多对多 13](#_Toc425928573)

[3.4. 复合主键 14](#_Toc425928574)

[3.5. 多表查询 14](#_Toc425928575)

[3.6. 树结构 14](#_Toc425928576)

[3.7. 执行原生sql 14](#_Toc425928577)

[3.8. 继承 14](#_Toc425928578)

# 概述

## 简介

## 目的

本文档供后端开发人员参考。

## 适用范围

本规范的适用范围包括产品研发前端平台的全部工作产品以及相关评审记录。

本规范只在本部门范围内有效，对其他部门没有参考作用。

## 文档读者

本规范的预期读者为智控美信相关领导和参与本部门的所有管理、设计、开发人员，未经智控美信书面允许，请勿扩散。

## 词汇表

* **Pascal case** － 所有单词第一个字母大写，其它字母小写。

例：Person, OrderDetail, OilTank。

* **Camel case －** 除了第一个单词，所有单词第一个字母大写，其他字母小写。

例：oilLevel, customerName

在后面的描述中我们使用 Pascal 代表第一种表示方式， Camel 代表第二种表示方式

## 参考资料

1. 智控-平台-规范-JAVA编码规范.docx

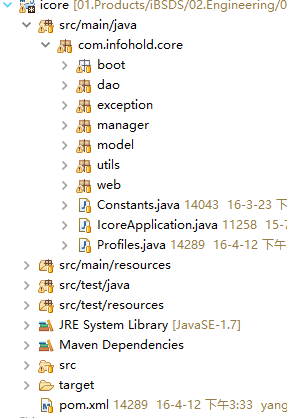
# 系统结构

## icore(SpringBoot)

使用icore核心进行编程（即springboot框架）,应该符合如下要求：

### 目录格式

目录格式以下图为参考



src/main/java：-------------------- 工程java源码目录

Com.infohold.core：-------- 核心包Core路径

boot：--------------------- Spring Boot相关代码

dao：---------------------- 抽象Dao接口定义目录

exception：-------------- 抽象exception类定义目录

manager：--------------- 抽象Manager接口定义目录

model：------------------ 数据库实体类对应目录，采用纯pojo类编写，所有属性必须提供get set方法

utils：--------------------- 系统基础工具类

web：-------------------- restful风格的web入口放置到模块文件夹下的web.rest目录下

Constants.java：------ 系统常量定义

IcoreApplication.java：系统启动类

Profiles.java：--------- 系统环境定义

src/main/resources：------------- 存放相关资源文件

src/test/java：---------------------- java测试类

src/ test / resources：------------ 存放测试相关资源文件

### 文件命名

* Java文件采用驼峰式写法，首字母大写.
* Rest目录下的类以controller为后缀，如果存在manager或者service层则定义其接口与实现，接口以manager为后缀，实现以managerImpl为后缀。
* 文件名或者缩写尽量有意义，保证可读性。

### 方法定义

* 控制方法体的长度，避免出现大方法，长方法。将长方法按照逻辑分成不同的方法去调用，提高代码复用率。
* 尽量写清楚注释，添加必要的注解，原则上java类中不允许出现编译错误与编译警告。
* 在需要使用泛型的地方必须使用泛型。如list、map等集合类。

## bdrp(基础开发库)

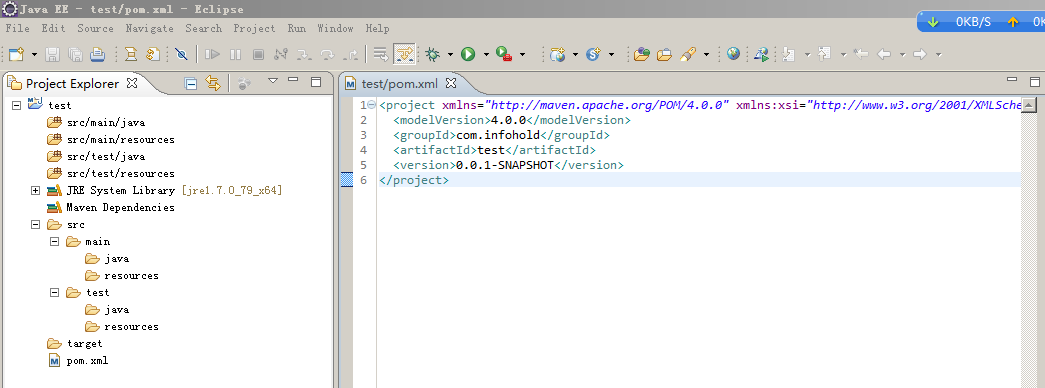
继承icore，包含系统基础功能模块，包括bdrp-common、bdrp-org、bdrp-authority、bdrp-scheduler、bdrp-ud、bdrp-log、bdrp-message等模块



# 开发步骤

## bdrp引入

新建空白项目如下：



按如下修改pom引入bdrp

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.infohold</groupId>

<artifactId>test</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<!-- icore定义的启动类，内部调用spring程序加载spring配置文件并启动tomcat -->

<start-class>com.infohold.core.IcoreApplication</start-class>

<java.version>1.7</java.version>

</properties>

<dependencies>

<dependency>

<groupId>com.infohold.bdrp</groupId>

<artifactId>bdrp-ud</artifactId>

<version>2.0.1-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

bdrp各子模块都依赖于icore，各子模块实现了不同的功能，新建项目只需依赖需要的bdrp模块即可，无须直接依赖icore。

## config配置

src/main/resources下新建系统配置文件application.properties

# Database and datasource settings

spring.jpa.database=MYSQL

spring.datasource.initialize=false

spring.datasource.sqlScriptEncoding=UTF-8

spring.datasource.max-active=20

spring.datasource.max-idle=8

spring.datasource.min-idle=8

spring.datasource.initial-size=10

spring.datasource.validation-query=select 1

spring.datasource.test-on-borrow=true

spring.datasource.url=jdbc:mysql://localhost:3306/SAMPLES?characterEncoding=utf8

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

# server settings

server.port=8088

# /info endpoint

info.app.name= Infohold test

info.app.version=${project.version}

# logging

logging.file=D:/tmp/logs/bootmore.log

## model开发

1.新建USER\_INFO表，建表语句如下：

mysql版：

create table USER\_INFO

(

ID VARCHAR(32) not null primary key,

NAME VARCHAR(80),

STATUS VARCHAR(1)

);

2.Model示例，在src\main\java\com\infohold\bdrp\samples\model目录下新建model，如下：

**package** com.infohold.bdrp.samples.model;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Table;

**import** com.infohold.core.model.BaseIdModel;

@Entity

@Table(name="USER\_INFO")

**public** **class** User **extends** BaseIdModel{

**private** **static** **final** **long** ***serialVersionUID*** = -4141898108921268666L;

**private** String name;

**private** String myState;//此命名用于演示Column注解

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Column(name="STATUS")

**public** String getMyState() {

**return** myState;

}

**public** **void** setMyState(String myState) {

**this**.myState = myState;

}

}

注意：

* 包名必须基于com.infohold，否则icore扫描不到实体。
* 如果带id属性则必须继承BaseIdModel。
* 符合驼峰规则的实体类名和属性名会自动转为带下杠的表名和字段名，比如myUser->MY\_USER，myName->MY\_NAME，myage->MYAGE。
* 如果数据库系统是大小写敏感的，建表语句里表名和字段名必须用大写（比如linux上的mysql），icore内部已经指定实体映射出的sql里的表名和字段名均为大写。

## dao开发

1. 在spring中声明UserDao对象，在src\main\resources\config目录下创建

applicationContext-user.xml，代码如下：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-4.0.xsd"*

default-lazy-init=*"false"*>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 案例 User START \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<bean id=*"userDao"* class=*"com.infohold.core.dao.impl.GenericDaoJpaImpl"*>

<constructor-arg value=*"com.infohold.bdrp.samples.model.User"* />

</bean>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 案例 User END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

</beans>

注意：

* icore框架要求统一使用GenericDao对表做增删改查操作，GenericDao为icore下dao包里定义的泛型接口，内部定义了增删改查基本方法（具体参见icore的javadoc），目前的实现类只有GenericDaoJpaImpl实现类，此实现类定义为通过构造器注入实体，注入实体后此GenericDao对象内部操作都是对注入的实体类型进行操作。

## manager开发

1. 在spring中声明UserManager对象，将如下红色代码添加到applicationContext-user.xml中，

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-4.0.xsd"*

default-lazy-init=*"false"*>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 案例 User START \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<bean id=*"userDao"* class=*"com.infohold.core.dao.impl.GenericDaoJpaImpl"*>

<constructor-arg value=*"com.infohold.bdrp.samples.model.User"* />

</bean>

<bean id=*"userManager"* class=*"com.infohold.core.manager.impl.GenericManagerImpl"*>

<constructor-arg ref=*"userDao"* />

</bean>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 案例 User END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

</beans>

注意

* 接口类命名以xxxxManager.java格式，实现类用xxxxManagerImpl.java格式，一般以接口方式进行定义，并将实现类定义于此目录下impl包中；
* Controller层对表操作用GenericManager，不要用GenericDao（GenericDao不受事务控制，icore事务配置目前为只匹配类名和包名都含有Manager的类）；
* GenericManagerImpl内部所有操作都转发给了GenericDao，GenericManager相比GenericDao仅多了事务控制；
* 无须自定义manager，目前icore事务仅控制在Manager层，未来会控制在controller层，开发时就把controller当作service来写即可；
* 泛型manager层原理同dao，通过构造器注入dao；

## controller开发

1. 在src\main\java\com\infohold\bdrp\samples\web\rest目录下创建UserRestControl.java，如下：

**package** com.infohold.bdrp.samples.web.rest;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.ModelAttribute;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.infohold.bdrp.samples.model.User;

**import** com.infohold.core.dao.Page;

**import** com.infohold.core.manager.GenericManager;

**import** com.infohold.core.web.MediaTypes;

**import** com.infohold.core.web.rest.BaseRestController;

/\*\*

\* 用户管理

\* **@author** Administrator

\*

\*/

@RestController

@RequestMapping("/user")

**public** **class** UserRestControl **extends** BaseRestController{

@Autowired

**private** GenericManager<User, String> userManager;

/\*\* 新增 \*/

@RequestMapping(value = "/create", method = RequestMethod.***POST***, produces = MediaTypes.***JSON\_UTF\_8***)

**public** User create(@ModelAttribute("user") User user){

User u = **new** User();

u.setName(user.getName());

u.setMyState("0");

userManager.save(u);

**return** u;

}

/\*\* 修改 \*/

@RequestMapping(value = "/update", method = RequestMethod.***PUT***, produces = MediaTypes.***JSON\_UTF\_8***)

**public** User update(@RequestBody User user){

User u = userManager.get(user.getId());

u.setName(user.getName());

userManager.save(u);

**return** u;

}

/\*\* 删除 \*/

@RequestMapping(value = "/remove/{id}", method = RequestMethod.***DELETE***, produces = MediaTypes.***JSON\_UTF\_8***)

**public** **void** remove(@PathVariable("id") String id){

userManager.delete(id);

}

/\*\* 返回单个 \*/

@RequestMapping(value = "/{id}", method = RequestMethod.***GET***, produces = MediaTypes.***JSON\_UTF\_8***)

**public** User get(@PathVariable("id") String id){

User u = userManager.get(id);

**return** u;

}

/\*\* 返回全部 \*/

@RequestMapping(value = "/", method = RequestMethod.***GET***, produces = MediaTypes.***JSON\_UTF\_8***)

**public** List<User> getAll(){

List<User> list = userManager.findAll();

**return** list;

}

/\*\* 返回分页 \*/

@RequestMapping(value = "/{start}/{limit}", method = RequestMethod.***GET***, produces = MediaTypes.***JSON\_UTF\_8***)

**public** List<User> getPage(@PathVariable("start") String start, @PathVariable("limit") String limit){

Page page = **new** Page();

page.setStart(start);

page.setPageSize(limit);

page.setQuery(" from User where myState=?0 order by id ");

page.setValues(**new** Object[] {"0"});

userManager.findPage(page);

//or

//List<User> list = userManager.findPageList(start, limit, " from User where myState=?0 or myState=?1 order by id ", "0", "1");

@SuppressWarnings("unchecked")

List<User> list = (List<User>)page.getResult();

**return** list;

}

}

注意

* 该层是与前端通讯的交互层，前端数据的接收、发送都依赖于该层；
* 该层基于Spring MVC 架构实现；
* 该层使用基于自定义Controller的BaseRestController构建；
* 对web层提供的方法应符合restful风格，具体体现在：
  + rest是面向资源的，因此方法命名应尽量符合资源的语义。例如：请求支付列表方法，应定义为payList，而不是getPayList。
  + 使用http原有的get post put delete方式区分操作。其中get用于查询，post用于修改，put用于新增，delete用于删除。例如请求支付列表方法payList，应该只支持get方式，如果传入参数过多，可以适当使用post方式，而支付方法pay，应该只支持post方式。如果所用浏览器支持的http操作不全，可以采用post方式替代其他操作。
* 采用spring依赖注入的方式使用spring 的bean.一般service\mananger、dao等bean定义为单例模式，而action、controller等bean定义为prototype模式；
* Sql语句符合JPA语法规范，尽量采用传参方式而不是使用拼写sql进行硬解析的方式进行数据库查询；

## 单元测试

1. 在src\test\java\com\infohold\bdrp\samples\web\rest目录下创建UserRestControlTest.java，如下：

**package** com.infohold.bdrp.samples.web.rest;

**import** org.junit.Ignore;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.springframework.boot.test.SpringApplicationConfiguration;

**import** org.springframework.boot.test.TestRestTemplate;

**import** org.springframework.boot.test.WebIntegrationTest;

**import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

**import** org.springframework.util.LinkedMultiValueMap;

**import** org.springframework.util.MultiValueMap;

**import** org.springframework.web.client.RestTemplate;

**import** com.infohold.bdrp.samples.model.User;

**import** com.infohold.core.IcoreApplication;

**import** com.infohold.core.utils.JSONUtils;

//下面三行注释就不会启动tomcat，否则启动

//如果单独启动项目可注释下面三行，这样此类就相当于浏览器

@RunWith(SpringJUnit4ClassRunner.**class**)

@SpringApplicationConfiguration(classes=IcoreApplication.**class**)

@WebIntegrationTest

**public** **class** UserRestControlTest {

**private** String url = "http://localhost:8088";

@Test

@Ignore

**public** **void** create() **throws** Exception{

System.***out***.println("create");

RestTemplate template = **new** TestRestTemplate();

MultiValueMap<String, String> bodyMap = **new** LinkedMultiValueMap<String, String>();

bodyMap.add("name", "张三");

String result = template.postForObject(url+"/user/create", bodyMap, String.**class**);

org.junit.Assert.*assertNotNull*(result);

org.junit.Assert.*assertThat*(result, org.hamcrest.Matchers.*containsString*("张三"));

}

@Test

@Ignore

**public** **void** update(){

System.***out***.println("update");

RestTemplate template = **new** TestRestTemplate();

User user = **new** User();

user.setId("4028f6814eb4fb57014eb4fb649d0000");

user.setName("张三2");

template.put(url+"/user/update", user);

String result = JSONUtils.*serialize*(user);

org.junit.Assert.*assertNotNull*(result);

org.junit.Assert.*assertThat*(result, org.hamcrest.Matchers.*containsString*("张三2"));

}

@Test

@Ignore

**public** **void** remove(){

System.***out***.println("remove");

RestTemplate template = **new** TestRestTemplate();

template.delete(url+"/user/remove/{id}", "8a81808b4eb49f64014eb49f72040000");

}

@Test

@Ignore

**public** **void** getOne(){

System.***out***.println("getOne");

RestTemplate template = **new** TestRestTemplate();

String result = template.getForObject(url+"/user/{id}", String.**class**, "4028f6814eb4fb57014eb4fb649d0000");

System.***out***.println("client receive:"+result);

org.junit.Assert.*assertNotNull*(result);

org.junit.Assert.*assertThat*(result, org.hamcrest.Matchers.*containsString*("张三"));

}

@Test

@Ignore

**public** **void** getAll(){

System.***out***.println("getAll");

RestTemplate template = **new** TestRestTemplate();

String result = template.getForObject(url+"/user/", String.**class**);

System.***out***.println("client receive:"+result);

org.junit.Assert.*assertNotNull*(result);

org.junit.Assert.*assertThat*(result, org.hamcrest.Matchers.*containsString*("张三"));

}

@Test

@Ignore

**public** **void** getPage(){

System.***out***.println("getPage");

RestTemplate template = **new** TestRestTemplate();

String result = template.getForObject(url+"/user/{start}/{limit}", String.**class**, 0, 2);

System.***out***.println("client receive:"+result);

org.junit.Assert.*assertNotNull*(result);

}

}

# 常见场景代码

推荐实体间关系对象上不使用jpa的关系注解，自己手工维护关系（实体内关联对象上要使用@Transient注解，下面的树是例外）。

最新的代码请参考bdrp-samples工程。

建表语句如下：



## 一对一



## 一对多



## 多对多



## 复合主键



## 多表查询



## 树结构

这里是单向树例子，内部没用到懒加载



## 执行sql

例子里包含调用存储过程



## 继承

