

文档编号: 9iuspring 剧情说明文档

文档类别: ☐ 公司级 ☐ 部门级 ☒ 项目级 ☐ 普通级

保密级别: ☐ 绝密 ☐ 机密 ☐ 秘密 ☒ 普通

9iuspring 剧情说明文档

版本: 0.3

2013-06-22

文档标识符号

符 号	说 明	示 例
蓝色文字	名词或叙述	9iuspring
TM	商标	9iuspring™
®	注册商标	9iuspring®
上标数字	注释	9iuspring ¹
	帮助	 帮助:XXXXXX
	注意	 注意:XXXXXX
	警告	 警告:XXXXXX
	技巧	 技巧:XXXXXX
	说明	 说明:XXXXXX

版本说明

版本	更新日期	更新者	更新记录
0.1	2012-12-23	9iu.org	定稿
0.2	2013-04-05	9iu.org	稳定版本
0.3	2013-06-16	9iu.org	修复版本, 完善代码生成器

审核记录

版本	审核日期	审核者	审核记录

目录

文档标识符号.....	2
版本说明.....	2
审核记录.....	2
目录.....	3
1 引言.....	4
2 预告片.....	4
2.1 下载 9iuspring, 并导入 Eclipse.....	4
2.2 执行 sql 脚本.....	4
2.3 执行代码生成器.....	4
2.4 你可以下班了.....	5
3 探班.....	6
3.1 Freemarker 模版.....	6
3.1.1 页面预览.....	6
3.1.2 查询条件.....	6
3.1.3 字段排序.....	6
3.1.4 复选框插件.....	7
3.1.5 自动生成列名.....	7
3.2 Controller.....	8
3.3 Service.....	8
3.4 Dao.....	9
3.4.1 主要实现.....	9
3.4.2 数据库 Dialect.....	10
3.5 Entity.....	11
3.6 缓存.....	12
3.7 工具类.....	12
4 整体架构.....	13
4.1 架构图.....	13
5 其他.....	13

1 引言

从业多年, 参演多部屌丝程序猿主演的加班连续剧, 情节悠长丰富, 发人深省, 深刻描绘了现实中亲情, 爱情, 友情在加班和 bug 面前的无力和苍白. 揭露了代码建筑工的生存现状.

为了在加班路上和正在加班的程序猿, 为了节约剧组经费, 压缩拍摄周期, 减少群众演员的盒饭成本, 大戏上演, 敬请期待!

2 预告片

2.1 下载 9iuspring, 并导入 Eclipse

下载地址: <http://git.oschina.net/chunanyong/spring9iu>



下载后解压, 把 spring9iu 这个项目工程导入 Eclipse

2.2 执行 sql 脚本

在你机器上的 mysql 上新建名为 tesbdb1 的数据库, 账号 root 密码 root, 执行 9iuspring/sql/springrain.sql

2.3 执行代码生成器

轻轻的点击 9iuspring/gencode/rapid-gen.bat 在弹出的窗口内输入 gen user 然后狠狠的回车. 我刚才连续输入了 9 个点, 差不多应该该弹出了一个文件夹了吧. di_car/freemarker 对应拷贝到 9iuspring/WebROOT/WEB-INF/freemarker

di_car/js 对应拷贝到 9iuspring/WebROOT/js
di_car/src_main 对应拷贝到 9iuspring/src

2.4 你可以下班了

嗯, 你可以下班去搞基了, 因为功能已经完成了.

纳尼??!! 你不信, 运行项目看看.

在 eclipse 中运行 9iuspring 项目, 通过浏览器访问, 例如: <http://127.0.0.1:8080/9iuspring>

你发现没有, 用户管理 这个功能可以使用了, 这一切多亏你刚才拷贝的代码啊.

你不信测试下 添加 查询 修改 删除 导出.



算你心细, 竟然还有字段排序!

飘柔, 就是这么自信.....

3 探班

3.1 Freemarker 模版

框架使用了 freemarker 前台渲染, 非常简单稳定高效的模版引擎, 语法也很简单, 看下文档和例子就能上手开发.

3.1.1 页面预览

Freemarker 生成的列表和修改两张模版页面, 例如 userList.html, userCru.html, 以 User 的页面为例列表页面如下:



3.1.2 查询条件

查询条件的代码为:

```
名称:<input type="text" id="name" name="name" value="${(users.name)!}" class="inp_2" />
```

name 和 org.iu9.testdb1.entity.Users 的属性名称保持一直, 后台会自动封装查询条件. `${(users.name)!}` 就是直接从封装对象中取值.

3.1.3 字段排序

生成的页面中有以下代码:

```
<!--first_end_no_export-->
<th id="th_name">姓名</th>
```

类似 的 html 注释不要修改和删除, 导出会用到.

表头 th 列中 id 以 "th_" 开头的列具有排序功能, th_之后的是需要排序的字段, 本例中就是需要后台按照 "name" 字段进行排序, 当然也可以是其他, 例如添加了别名 可以是 "th_u.name", 这个主要和后台的查询语句有关.

3.1.4 复选框插件

复选框插件 js 为: js/plugins/jquery.checkbox.js

```

1  /**
2   * checkbox 全选操作
3   *
4   *
5   * @example $('input[@type=checkbox][@name=checkAll]').checkboxbox(); 自动切换 :
6   *      .toggle(element) 全选 : .checked(element) 反选 : .unchecked(element)
7   *      获取字符串值 : .val()
8   *
9   *
10  * $('input[name=checkAll]').checkboxbox().toggle('input[name=checkbox]');
11  * //自动切换全选/反选
12  * $('input[name=checkAll]').checkboxbox().checked('input[name=checkbox]'); //全选
13  * $('input[name=checkAll]').checkboxbox().unchecked('input[name=checkbox]'); //反选
14  * $('input[name=checkbox]').checkboxbox().val(); //获取字符串值
15  */

```

3.1.5 自动生成列名

列表表格的列 姓名 是代码生成器从数据库取值字段的说明, Users 表中, 字段 name 的注记(备注)是 “姓名”

建议大家维护好数据库中字段的备注说明, 这样生成的代码会友好很多.

栏位	索引	外键	触发器	选项	注记	SQL 预览
名						
id						varchar 50 0 允许空值 0 1
name						varchar 200 0 允许空值 0

默认:
 注记: 姓名 就是这个
 字符集: utf8
 整理: utf8_general_ci
 键长度:
☐ 二进制

3.2 Controller

框架使用 springmvc, springmvc 非常的强大灵活和高性能, 基于注解的方式, rest 风格.....
增加, 修改, 删除都比较简单, 我们主要说一下列表查询

```
@RequestMapping("/list")
public String list(HttpServletRequest request, Model model, Users users) throws Exception
// ==构造分页对象
Page page = newPage(request);
// ==执行分页查询
List<Users> datas=usersService.findListDataByFinder(null,page,Users.class,users);
// ==分页对象封装到前台
model.addAttribute("page", page);
// ==列表数据封装到前台
model.addAttribute("datas",datas);
// ==把查询条件重新封装到前台
model.addAttribute("users",users);
return listurl;
}
```

↑
前台参数封装到 users

主要使用 findListDataByFinder Service 方法进行查询, 也可以自己在 Service 构建 Finder 查询

```
@Override
public <T> List<T> findListDataByFinder(Finder finder, Page page, Class<T> clazz,
Object o) throws Exception{
//controller传递的就是 Users 对象,所以可以强转,
//框架并没有强制Entity作为QueryBean,只是默认为QueryBean,你可以自己封装QueryBean.
Users u=(Users) o;
//初始化 finder,并且为users 别名为 u
finder=new Finder("SELECT u.* FROM users u WHERE 1=1 ");
//设置表别名为 u
u.setFrameTableAlias("u");
//把QueryBean拼接SQL查询语句,语句必须已经包含 WHERE, 所以 finder初始化的时候有 WHERE 1=1
super.getFinderWhereByQueryBean(finder, u);
//返回查询结果
return super.queryForList(finder, clazz,page);
// return super.findListDataByFinder(finder,page,clazz,o);
}
```

3.3 Service

每个数据库都会基本的 Service 例如:数据库 testdb1

```
@Service("baseTestdb1Service")
public class BaseTestdb1ServiceImpl extends BaseServiceImpl implements IBaseTestdb1Service {
```

继承数据库的基本 service 派生 业务 service, 例如 usersService

```
@Service("usersService")
public class UsersServiceImpl extends BaseTestdb1ServiceImpl implements IUsersService {
```


父类 service 已经提供了基本的操作方法, 包含 增删改查, 有兴趣可以详细看下接口.
方法形参中如果传入泛型, 就会返回泛型的对象, 如果不传入泛型就会返回 Map 对象.

```
Finder finder=new Finder("SELECT * FROM Users order by id");
List<Map<String, Object>> listMap = usersService.queryForList(finder);
List<Users> listEntity = usersService.queryForList(finder,Users.class);

finder=new Finder("SELECT * FROM Users WHERE id='admin' ");
Map<String, Object> map = usersService.queryForObject(finder);
Users user = usersService.queryForObject(finder,Users.class);
```

详细参数方法, 可以参考 doc/javadoc, 代码中的注释已经比较详细了.

3. 4 Dao

3. 4. 1主要实现

每个数据库都会有有一个 Dao, 强烈建议一个数据库只有一个 Dao, 业务可以扩展 Service.
例如:

```
@Repository("baseTestdb1Dao")
public class BaseTestdb1DaoImpl extends BaseJdbcDaoImpl implements IBaseTestdb1Dao{
```

每个数据库只需要一个 Dao, 业务通过扩展 service

如果你的项目有多个数据库怎么办?

总共分三步:

1. 在 db.properties 中添加数据库的连接字符串和账号密码
2. 拷贝该 datasource 和 transaction 的配置文件

```
spring
  applicationContext-cache.xml
  applicationContext-comm.xml
  applicationContext-datasource.xml
  applicationContext-tx.xml
  applicationContext.xml
```

例如 新增 applicationContext-datasource-testdb2.xml 和 applicationContext-tx-testdb2.xml

3. 创建基本的 Dao 和 Service

在 dao 中注入配置文件声明的 NamedParameterJdbcTemplate 和 SimpleJdbcCall, 名称和 spring Bean 一致.

因为默认没有使用 JTA, 每个数据库的事务是独立的. 所以每个数据库应该有一个独立的根包路径, 主要是为了方便 spring 事务扫描, 当然如果配置了 JTA, 就无所谓了.

主要方法实现如下图:

```

/**
 * 抽象方法,每个数据库的代理Dao都必须实现,在多库情况下,用于区分底层数据库的连接对象,对数据库进行增删改查.</br>
 * 例如:testdb1数据库的代理Dao org.iu9.testdb1.dao.BaseTestdb1DaoImpl 实现返回的是spring的bean jdbc.</br>
 * testdb2 数据库的代理Dao org.iu9.testdb2.dao.BaseTestdb2DaoImpl
 * 实现返回的是spring的bean jdbc_testdb2.</br>
 *
 * @return
 */
public abstract NamedParameterJdbcTemplate getJdbc();

/**
 * 抽象方法,每个数据库的代理Dao都必须实现,在多库情况下,用于区分数据库实例的日志记录表,
 * 主要是为了兼容日志表(auditlog)的主键生成方式,UUID和自增.</br>
 * testdb1 数据库的auditlog 是自增,testdb2 数据库的 auditlog 是UUID
 *
 * @return
 */
public abstract IAuditLog getAuditLog();

/**
 * 抽象方法,每个数据库的代理Dao都必须实现,在多库情况下,用于区分底层数据库的连接对象,调用数据库的函数和存储过程.</br>
 * 例如:testdb1 数据库的代理Dao org.iu9.testdb1.dao.BaseTestdb1DaoImpl 实现返回的是spring的bean jdbcCall.</br>
 * datalog 数据库的代理Dao org.iu9.testdb2.dao.BaseTestdb2DaoImpl 实现返回的是spring的bean jdbcCall_testdb2.</br>
 *
 * @return
 */
public abstract SimpleJdbcCall getJdbcCall();

/**
 * 获取数据库方言,Dao 中注入spring bean.</br>
 * 例如mysql的实现是 mysqlDialect.
 * oracle的实现是 oracleDialect.
 * 如果使用了sequence 在entity使用@PKSequence实现自增
 * 详见 org.iu9.frame.dao.dialect.IDialect的实现
 *
 * @return
 */
public abstract IDialect getDialect();

```

3.4.2 数据库 Dialect

只有在查询分页时需要考虑数据库差异,实现 org.iu9.frame.dao.dialect.IDialect 接口即可.



数据库名称	实现分页函数	说明
SQLServer	ROW_NUMBER()	支持 sql2005+的版本, 不支持 sql2000
Oracle	rownum	
DB2	ROWNUMBER()	
Informix	SKIP FIRST	
PostgreSql	limit	
SQLite3	limit	
Sybase	未实现	

在 dao 中注入实现的 spring bean 即可.

```
/**
 * mysqlDialect 是mysql的方言,springBean的名称,可以参考 IDialect的实现
 */
@Resource
public IDialect mysqlDialect;
@Override
public IDialect getDialect() {
    return mysqlDialect;
}
```

这是直接注入 mysqlDialect, 就是 mysql 的数据库, 你可以注入 IDialect 的实现即可, 例如 oracleDialect. 如果你使用了 oracle 的 sequence, 需要在 Entity 使用@PKSequence, 配合@Id 实现 Sequence 自增.

3.5 Entity

Entity 默认包是 org.iu9.frame.entity.BaseEntity 为基础父类, 所有的实体 Entity 必须继承 BaseEntity

使用的注解是

@Table 为映射的表名

@TableGroup 分表后缀. 值为获取分表后缀的字段, 在 save 或者 update 对象操作时, 可以根据对象的属性值确定分表的后缀. 参见 org.iu9.testdb1.entity.AuditLog

@Id 为主键 ID, 放在字段的 get 方法上, 可以支持 UUID 和自增, 默认为 UUID

@Transient 放在字段的 get 方法上, 标示数据库不存在的字段

@WhereSQL, 拼装 sql 的 where 条件, 对于简单查询, entity 可以直接作为 querybean 作为查询条件.

最后通过 org.iu9.frame.dao.BaseJdbcDaoImpl.getFinderWhereByQueryBean(Finder, Object)

拼装 where 条件, Object 形参就是 QueryBean, 默认为 Entity.

```
@WhereSQL(sql="operatorName<:Auditlog_operatorName")
```

```
@WhereSQL(sql="operationType like :%Auditlog_operationType%")
```

Entity 的属性名需要和数据库完全一致, 也可以再拼写 sql 语句时起别名.

@PKSequence, 处理数据库 sequence 的主键自增, 这个注解必须和@Id 配合使用, 当 Number 类型的主键值为 null 时, 会取值@PKSequence 下面的是 oracle 的序列

```
@PKSequence(name="test.nextvalue")
@Id
@WhereSQL(sql="id=:testTable_id")
public java.lang.Integer getId() {
```

最终执行的语句 类似如下

```
insert into testtable(id ,name) values(test.nextvalue,"testName");
```

3.6 缓存

框架已经启用了 spring 的缓存管理. 配置文件为:applicationContext-cache.xml. 整体来说 spring 的缓存管理相当的灵活和强大.

使用方法例如:

```
@Cacheable(value = GlobalStatic.cacheKey, key = "getUserByNameById_'+#userId ")
public String getUserByNameById(Integer userId) throws Exception {
    Finder f = new Finder("select Name from [user] where ID=:userId");
    f.setParam("userId", userId);
    String name = this.queryForObject(f, String.class);
    return name;
}
```

具体详见 spring 的帮助文档

3.7 工具类

Finder: 拼装 sql 语句的工具类, 所有的自行拼装语句都必须借助这个工具类, 即统一了编码风格, 也有利于以后的维护.

ClassUtils: 进行反射的工具类, 一般反射之后, 就会把信息缓存

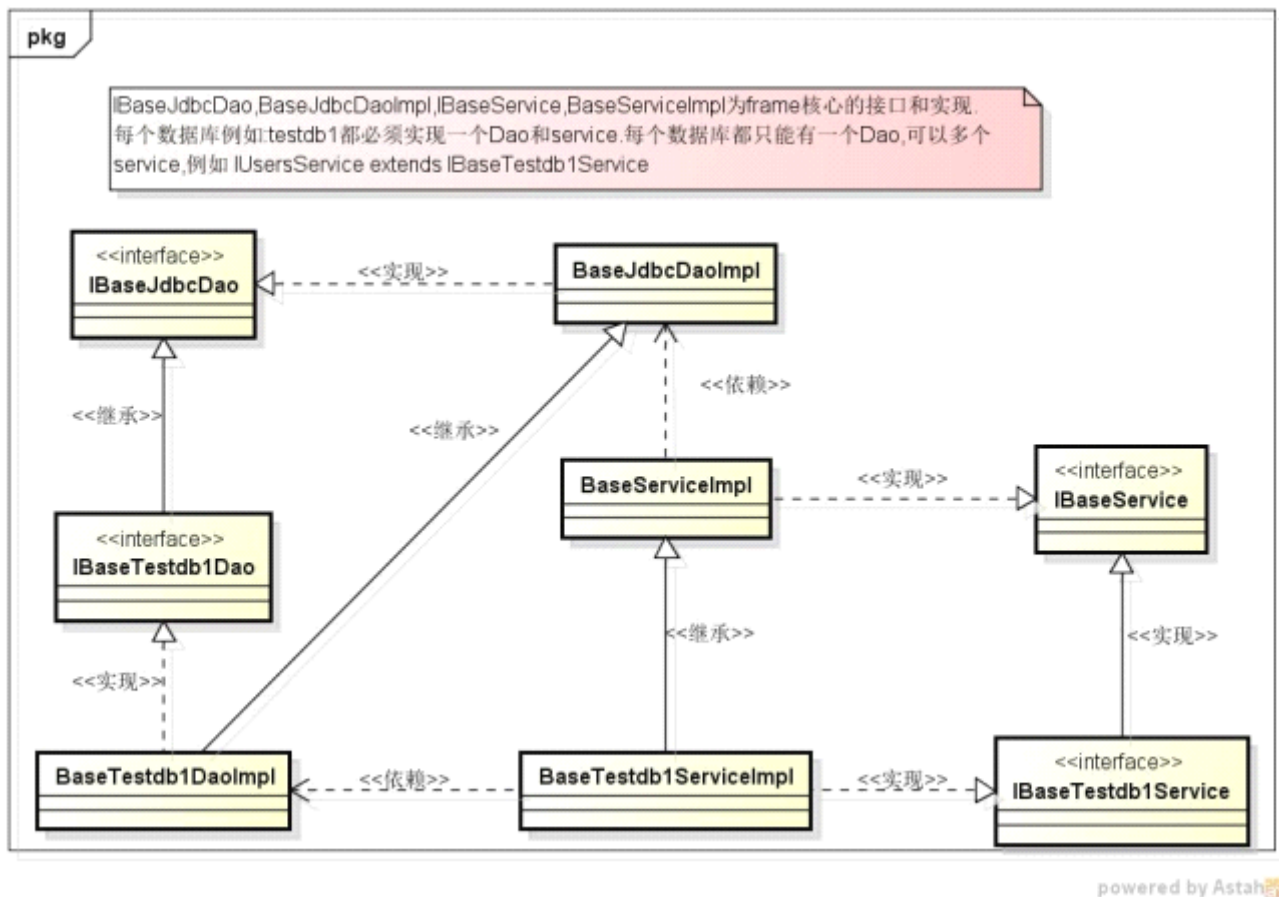
SpringUtils: 主要是用来手动获取 spring bean, 已经注入到了 BaseServiceImpl, 每个 service 都可以通过 getBean(String beanName) 获取任意一个 springBean, 是为了应对不可预料的复杂情况

GlobalStatic: 定义全局变量.

SessionUser: 一个静态类, 可以随时获取当前用户的登陆情况, 主要是通过 HttpSession session = FWInterceptor.sessionLocal.get(); 这样我们就可以再任何地方拿到当前用户的 session 信息. 当然, session 的键值可以自行定义.

4 整体架构

4.1 架构图



基本架构就是如此了，先写到这吧。

5 其他

很感谢你看到此处，你能看到这句话，本身就是对我的支持！