# Spring与Hibernate整合

1.配置数据源 3种方式：推荐用c3p0

2.配置hibernate的sessionFactory

3.把sessionFactory注入到Dao层

导包:

Spring基础类库+Hibernate基础类库

整合需要使用的类库:

commons-dbcp-1.4.jar //c3p0

commons.pool-1.6.jar

或者

commons-dbcp2-2.1.1.jar

commons-pool2-2.4.2.jar

spring-jdbc-4.3.2.RELEASE.jar

spring-orm-4.3.2.RELEASE.jar

spring-tx-4.3.2.RELEASE.jar

Spring Bean的作用域:

Bean Scope Description

singleton This scopes the bean definition to a single instance per Spring IoC container (default).

prototype This scopes a single bean definition to have any number of object instances.

request This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

session This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

global-session This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

c3p0的配置参考：

<property name="dataSource">

<bean class="com.mchange.v2.c3p0.ComboPooledDataSource">

<!-- 数据连接信息 -->

<property name="jdbcUrl" value="${jdbcUrl}"></property>

<property name="driverClass" value="${driverClass}"></property>

<property name="user" value="${user}"></property>

<property name="password" value="${password}"></property>

<!-- 其他配置 -->

<!--初始化时获取三个连接，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->

<property name="initialPoolSize" value="3"></property>

<!--连接池中保留的最小连接数。Default: 3 -->

<property name="minPoolSize" value="3"></property>

<!--连接池中保留的最大连接数。Default: 15 -->

<property name="maxPoolSize" value="5"></property>

<!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->

<property name="acquireIncrement" value="3"></property>

<!-- 控制数据源内加载的PreparedStatements数量。如果maxStatements与maxStatementsPerConnection均为0，则缓存被关闭。Default: 0 -->

<property name="maxStatements" value="8"></property>

<!--maxStatementsPerConnection定义了连接池内单个连接所拥有的最大缓存statements数。Default: 0 -->

<property name="maxStatementsPerConnection" value="5"></property>

<!--最大空闲时间,1800秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->

<property name="maxIdleTime" value="1800"></property>

</bean>

</property>

1. SSH整合
   1. 项目的配置文件以及包的路径的配置
      1. 修改web.xml 添加spring 的contextConfigLocation web项目启动的时候自动加载Spring配置文件
      2. 修改web.xml 添加ContextLoaderListener,注意listener声明顺序
      3. web.xml配置字符编码
      4. web.xml配置视图使用hibernate的entity manager
      5. web.xml配置JSON拦截器：默认返回JSON数据
   2. Spring整合hibernate4与hibernate5.1支持jdk7, hibernate5.2只支持jdk8
   3. hibernate5.1的主键策略需要明确指定.
      1. @GeneratedValue(strategy=GenerationType.IDENTITY)
   4. 三层架构添加注解与扫描机制: 直接扫描根目录
      1. <context:component-scan base-package=*"com.ssh.demo"* />
      2. 持久层:@Repository
      3. 业务层:@Service
      4. 控制层:@Controller
      5. 组建:@Component:以上三层或者自定义类/工具

# Spring Data JPA

<http://www.ibm.com/developerworks/cn/opensource/os-cn-spring-jpa/>

<http://blog.csdn.net/yingxiake/article/details/51014223>

1. 启动事务管理

<!-- enable the configuration of transactional behavior based on annotations -->

<tx:annotation-driven transaction-manager="txManager"/><!-- a PlatformTransactionManager is still

required -->

<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!-- (this dependency is defined somewhere else) -->

<property name="dataSource" ref="dataSource"/>

</bean>

1. Spring Data JAP 与 Hibernate的整合

3．修改 spring orm配置文件：整合hibernate 的jpa支持

4．四大接口

4．1详细四大接口的用法:CRUD+分页查询.

4．2 高级查询

5．@Query

6. 持久层命名规则

前后台交互1:实现分页查询

Spring MVC + Spring Data JPA

1. 获取前端组件的分页参数:rows:10,page:1,sidx:id,sord:desc
2. Web层接收请求分页参数(封装为分页JavaBean)

@RequestParam("rows") **int** rows,

@RequestParam("page") **int** page,

@RequestParam("sidx") String sidx,//id

@RequestParam("sord") String sord

1. 排序:Sort sort = **new** Sort(Direction.***DESC***, sidx);
2. 分页+排序(起始页为0):Pageable pageable = **new** PageRequest(page-1, rows,sort);
3. 分页结果集:Page<User> userPage = userService.findAll(pageable);

注意:Web层要针对不同的前端的框架Jquery/Extjs/…做适配.(前端组件的默认参数名,起始页,响应的数据格式等,)

# 使用 Spring TestContext 测试框架

# Spring MVC Test

http://www.ibm.com/developerworks/cn/java/j-lo-spring25-test/

http://www.cnblogs.com/wade-xu/p/4311657.html

1. 导入测试包:

<properties>

<!-- spring -->

<spring.version>4.3.2.RELEASE</spring.version>

<junit.version>4.12</junit.version>

</properties>

<dependencys>

<!-- Test -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>xmlunit</groupId>

<artifactId>xmlunit</artifactId>

<version>1.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<version>0.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.hamcrest</groupId>

<artifactId>hamcrest-library</artifactId>

<version>1.3</version>

<scope>test</scope>

</dependency>

</dependencies>

1. 编写抽象的控制层上下文测试基类AbstractContextControllerTests,被所有测试类所继承.
   1. 用途:加载配置文件,并且根据配置文件初始化测试环境

//1.启动Web项目的测试配置

@WebAppConfiguration

//2.加载Spring配置文件初始化上下文环境.

@ContextConfiguration(locations={

"file:src/main/webapp/WEB-INF/spring-web-servlet.xml",

"file:src/main/resources/spring/applicationContext-bean.xml",

"file:src/main/resources/spring/applicationContext-orm.xml"

})

**public** **class** AbstractContextControllerTests {

//3.注入web程序的上下文对象,获取控制层对象

@Autowired

**protected** WebApplicationContext wac;

}

1. 编写测试用例

//1.在测试环境下运行 4 = for

@RunWith(SpringJUnit4ClassRunner.**class**)

//2.启动事务

@Transactional

**public** **class** UserControllerTests **extends** AbstractContextControllerTests

{

**private** MockMvc mockMvc;

//3.@Before : @Test运行之前执行:测试前的准备工作,例如初始化环境

@Before

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

**this**.mockMvc = *webAppContextSetup*(**this**.wac).alwaysExpect(*status*().isOk()).build();

}

//@Before //测试数据的初始化

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

}

//4.测试用例:一个测试类 可以有多个用例@Test,如果不希望某些用例运行则直接注释掉@Test即可.

@Test

@Rollback(**true**)//事务回滚:保护数据现场.配合@Transactional注解使用.

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

//使用 get方式 请求地址:/user/save 匹配返回数据(String)

**this**.mockMvc.perform(*get*("/user/save")).andExpect(*content*().string("Hello world!"));

}

/\*\*

\* 编写分页查询的测试用例

\* **@throws** Exception

\*/

//@Test//不运行

@Rollback(**true**)

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

}

}

1. 数据保护: @Transactional +　@Rollback(**true**)

前后台交互2:JqGrid+Spring Mvc(JSON解析)

## Data Mapping/JSON Mapping

1. 格式化JSON <http://www.bejson.com/> 或者谷歌浏览器F12 网络视图

{

"content": [

{

"id": 1,

"creater": "Sailer Wen",

"createrTime": 1473754124000,

"updater": null,

"updateTime": null,

"remaker": null,

"userName": "admin",

"password": "admin"

}

],

"last": true,

"totalPages": 1,

"totalElements": 1,

"sort": [

{

"direction": "DESC",

"property": "id",

"ignoreCase": false,

"nullHandling": "NATIVE",

"ascending": false

}

],

"first": true,

"numberOfElements": 1,

"size": 10,

"number": 0

}

1. 在页面添加视图组件Jqgrid
   1. 参考http://www.trirand.com/blog/jqgrid/jqgrid.html
2. 修改Jqgrid的id和工具栏id
3. 修改Jqgrid的列头和列模型
4. 修改URL+DataType(JSON)+jsonReader

jsonReader: {

root: "content", // 结果集引用名: json中代表实际模型数据的入口

total: "totalPages", // json中代表页码总数的数据

records: "totalElements", // json中代表数据行总数的数据

},

* 1. 解决日期格式不显示问题: **NaN-NaN-NaN NaN:NaN:NaN**

**Entity 的时间字段的get方法添加**

@JsonFormat(pattern="yyyy/MM/dd HH:mm:ss",timezone="GMT+8")

**Jqgrid的colModel字段的添加**

formatter:'date',formatoptions: {srcformat:'Y-m-d H:i:s',newformat:'Y/m/d H:i:s'}},

1. 测试

## New in version 3.2 / Custom Button and Forms

1. 添加一个按钮

$("#userList").jqGrid(**'navButtonAdd'**,'#userListToolBar',{

caption:"Edit",

onClickButton:**function**()//按钮的点击事件

{

//获取选中记录的id

**var** selrow = $("#userList").jqGrid('getGridParam','selrow');

**if**(selrow){

//alert("select Row userName:"+selrow);

//'GridToForm' 按选中的ID selrow 填充数据到表单 userForm

$("#userList").jqGrid('GridToForm',selrow,"#userForm");//重点'GridToForm'

} **else** {

alert("请选中一行记录!")

}

}

});

1. 添加表单：注意ID及其他系统维护字段，表单日期格式。action和method属性

<form method=*"post"* id=*"userForm"* action=*"user/saveOrUpdate"* title=*''* style="width:*350px*;margin:*0px*;">

<fieldset>

<legend>用户明细</legend>

<table>

<tbody>

<tr>

<!-- ID:用户不需要关系,由系统进行维护 -->

<td></td>

<td><input type=*"text"* name=*"id"* hidden=*"true"* readonly=*"true"* id=*"invid"*/></td>

</tr>

<tr>

<td> 用户名:</td>

<td><input type=*"text"* name=*"userName"* /></td>

</tr>

<tr>

<td>密码</td>

<td><input type=*"password"* name=*"password"* /></td>

</tr>

<tr>

<td>备注</td>

<td><input type=*"text"* name=*"remaker"* /></td>

</tr>

<tr>

<td>&nbsp;</td>

<td><input type=*"submit"* value=*"提交"*/></td>

</tr>

</tbody>

</table>

</fieldset>

</form>

1. 修改UserController的saveOrUpdate函数

@RequestMapping("/saveOrUpdate")

**public** @ResponseBody String saveOrUpdate(User user,HttpSession session)

{

**if**(user.getId()!=**null**){

//更新

//user.setUpdater(session.getAttribute("userName")+"");

user.setUpdateTime(**new** Date());

}**else**{

//保存/插入

//user.setCreater(session.getAttribute("userName")+"");

user.setCreaterTime(**new** Date());

}

//jpa save ==hibernate saveOrUpdate,根据id判断save or update

userService.save(user);

**return** "success:true";

}