**SSH框架开发流程**

1 项目中启用Struts2的核心Filter，让Filter拦截所有用户请求。

**1.1 maven配置Struts2需要的jar包，只需要配置如下一个包，其余依赖包自动下载**

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-core</artifactId>

<version>2.5</version>

</dependency>

**1.2 应用中启用Struts2, 需在web.xml中配置核心Filter，这里注意最新版本的filter中class改变了，没有ng**

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

2 Struts2中需要调用Action来请求用户处理，Action又依赖Service，先要配置Spring容器解决依赖问题，Spring又依赖于commons-logging包,maven会自动帮我们解决依赖问题。

**2.1 maven配置Spring需要的jar包**

spring-context包：定义了Spring依赖注入功能

spring-orm包：定义了Spring持久化依赖项-hibernate、jdbc

spring-web包：Spring web项目的需要，如web.xml中配置的Spring监听器

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.2.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.2.6.RELEASE</version>

</dependency>

**2.2 web.xml中添加Spring监听器，并配置spring.xml文件**

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>WEB-INF/classes/spring.xml</param-value>

</context-param>

**2.3 Spring整合Struts2框架,此时Struts2.xml中配置的class都是由Spring容器依赖注入，依靠下面插件自动完成**

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>2.5</version>

</dependency>

**2.4 Spring使用aop时需要aspectjweaver包依赖，利用maven配置此包**

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.9</version>

</dependency>

**2.5 Spring整合quartz框架，需要maven配置spring-context-support包**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>4.2.6.RELEASE</version>

</dependency>

3 搭建hibernate框架,这里把基于hibernate的缓存配置加入

**3.1 maven配置hibernate需要的jar包**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.1.0.Final</version>

</dependency>

**3.2 maven配置MySQL驱动包JDBC，hibernate封装了底层JDBC操作，尽量不要使用高版本的JDBC，连接方式会改变**

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.38</version>

</dependency>

**3.3 maven配置连接池C3P0包, Web应用中一般利用连接池连接数据库，连接池能够减少与数据库交互的次数，提高系统性能**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-c3p0</artifactId>

<version>5.1.0.Final</version>

</dependency>

**3.4编写hibernate.cfg.xml配置文件**

重点信息包括：数据库连接信息；c3p0配置信息；映射实体类信息(结合Domain层类编写)；

**3.5 配置二级缓存和查询缓存，需在hibernate.cfg.xml中配置，maven配置hibernate-ehcache依赖包**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<version>5.1.0.Final</version>

</dependency>

<!-- 配置二级缓存和查询缓存 -->

<property name="hibernate.cache.use\_query\_cache">true</property>

<property name="cache.use\_second\_level\_cache">true</property>

<!-- 二级缓存提供类 -->

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

<!-- 二级缓存配置文件的位置 -->

<property name="hibernate.cache.provider\_configuration\_file\_resource\_path">Hehcache.xml</property>

编写ehcache.xml文件，针对每一个实体类配置对应的cache参数：

< defaultCache

maxElementsInMemory="10000"

eternal="false"

timeToIdleSeconds="120" //空闲时间

timeToLiveSeconds="120" //生存时间

overflowToDisk="true"

/>

Domain层基于@注解实体类，表示此实体类开启二级缓存

@Cache(usage=CacheConcurrencyStrategy.READ\_ONLY)

@Cacheable

Dao层通过.setCacheable(true)方法在对应查询操作时开启查询缓存

getSessionFactory().getCurrentSession().createQuery(hql).setCacheable(true).list()

**3.6 maven配置hibernate验证器依赖的jar包，实体类写入数据库时校验器会触发**

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.2.4.Final</version>

</dependency>

Domain层实体类的属性基于@注解编写校验规则

Dao层插入数据库操作时加入验证策略：

//就只有插入的时候需要判断是否符合要求domain层约束，要是不符合则返回错误

ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

Validator validator = factory.getValidator();

Set<ConstraintViolation<T>> constraintViolations =validator.validate(entity);

if (constraintViolations.size() != 0) {

return false;

}else {

getSessionFactory().getCurrentSession().save(entity);

return true;

}

**3.7 hibernate使用SLF4J API记录日志，所以在Hibernate的lib中，不提供Log4J的包，但Log4J记录日志操作更加简单，通常都用Log4j管理日志，这样导致了兼容性问题。**

maven解决办法，两步：

一、添加Log4J包及Log4J-core包

二、再添加一个叫做slf4j-log4j12-1.5.11.jar的包进行转换

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.7</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.5</version>

</dependency>

4 完成软件架构层次

**Domain层**：实体类编写，只含有成员变量及对应的setter/getter方法，还有构造函数

**Dao层**：采用接口模式，先完成接口方法定义，在编写具体实现类；

采用继承机制，利用Spring依赖注入成员变量SessionFactory，然后通过session封装出CRUD基本操作成一个BaseDao<T>类，里面都是通用方法。

针对每一个实体类都应该编写对应的Dao类，里面包含其独有的查询方法。

**Service层**：采用接口模式，这里不用写出一个Base类用来继承，但为了避免一个Service类方法过于庞大，应该把相近的操作整合在一个Service类中，当然Service中需要依赖注入相应的Dao组件。

其次针对所有Service类公用的方法，可以在一个类中集成，采用静态方法。

**VOS层**：编写Service层需要的数据对象，本质是javabean，解决Domain层实体类属性和Service类需要属性不匹配问题。

**Action层**：采用继承机制，先编写父类，只依赖注入对应的Service，一个父类action依赖注入一个Service，然后编写每一个请求对应的子类action。

为避免所有的actions都在同一个包下，针对相近的actions整合在一个包下。

**Schedules层**：主要配置自启动任务，有规定的编码格式，也需要依赖注入Service逻辑。

5 Spring容器配置xml文件，依赖注入可通过设置模板简化操作，不同xml之间引用通过<import resource="spring.xml"></import>完成

**4.1 datasourceAndDao.xml: 配置数据源hibernate.cfg.xml并为Dao组件依赖注入session**

<!-- 配置数据源 -->

<bean id="sessionFactory" class="org.springframework.orm.**hibernate5**.LocalSessionFactoryBean">

<property name="configLocations" value="classpath:hibernate.cfg.xml"></property>

</bean>

这里特别注意需要根据hibernate的版本更换，若是4.0的则hibernate4，若是5.0的则hibernate5

**4.2 serviceAndActions.xml: 配置Service并依赖注入Dao组件，配置Action并依赖注入Service组件**

**4.3 transactionAndCache.xml: 配置数据库事务机制和缓存机制(Spring的处理缓存，需要在Service层对应的方法上通过@Cacheable注解标识)**

<bean id="transactionManager"

class="org.springframework.orm.**hibernate5**.HibernateTransactionManager">

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

</bean>

这里需要注意hibernate的版本

**4.4 autoSchedules.xml: 配置自启动机制，即通过quartz时间调度器控制Schedules层中自动执行的方法**

这里通过maven配置quartz依赖的包

<dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz</artifactId>

<version>2.2.3</version>

</dependency>

配置quartz.properties，声明自启动线程的一些规定

**4.5 依赖注入config.properties文件：配置参数都放在config.xml中，其余配置文件读取其的参数**

<!-- spring的属性加载器，加载properties文件中的属性 -->

<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location" value="classpath:config.properties"></property>

</bean>

<property name="logsurl" value="${Voronoilogs.url}"></property>

通过${\*\*\*}这种方式读取config.properties中配置，并依赖注入需要的属性中。

6 引用特殊类方法，利用maven配置相应jar包

6.1 Java中使用JSON需要的jar包

<dependency>

<groupId>net.sf.json-lib</groupId>

<artifactId>json-lib</artifactId>

<version>2.4</version>

<classifier>jdk15</classifier>

</dependency>

6.2 JTS套件需要的jar包

<dependency>

<groupId>com.vividsolutions</groupId>

<artifactId>jts</artifactId>

<version>1.13</version>

</dependency>

6.3 javacsv需要的jar包

<dependency>

<groupId>net.sourceforge.javacsv</groupId>

<artifactId>javacsv</artifactId>

<version>2.0</version>

</dependency>

7 配置struts2.xml文件，页面路由作用，其中class也是由Spring依赖注入

**7.1 maven解决struts2对JSON数据传输的支持**

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-json-plugin</artifactId>

<version>2.5</version>

</dependency>

**7.2前端url：getBSDataAction；后端处理类：VoronoiDrawAction；返回的JSON格式数据：在VoronoiDrawAction 中定义的Bsinfo属性**

<action name="getBSDataAction" class="VoronoiDrawAction">

<result type="json">

<param name="root">BSinfo</param>

</result>

</action>

这个action使得localhost:8080/\*\*project/能够跳转到首页，然后接下去的页面资源应用都可以同Webstorm创建的项目同步路径

<action name="\*">

<result>TJpages/index.html</result>

</action>

8 把webStorm开发的整个页面文件放在webapp下面