**第五部分 : 整体构架**

**设计模式：**

1. 单例模式：添加一个静态变量instance用于判断是否第一次构建，private构造函数放在静态方法getInstance()中。通常Service,Dao组件都应该设置成单例模式
2. 简单工厂：Computer有成员变量Output(是一个接口)，各类Printer都实现了Output，一个OutputFactory类含一个方法getOutout(String type).这样但需要不同的Printer时，只需要让getOutput()返回不同的new Printer()即可。Spring本身就是一个工厂，所以省略了创建OutputFactory过程，直接依赖注入不同的Printer即可。
3. 代理模式：好处:提高系统性能，A中关联B，加载A时，加载的是B的代理，而到真正用B的属性或方法时，才生成真正的B。hibernate中session.load(A.class,pk)就是加载代理；A实体关联B实体，当加载数据库A实体时，B加载的也只是代理，用到B才会去数据库加载。
4. 命令模式：execute(target t,Command m){m.process(t);} //根据m来决定对t处理方法，Command定义为一个函数式接口，java中匿名内部类new Command(){//函数实现代码}和Lambda表达式{函数参数}->{//函数实现代码}都可以实现命令模式。
5. 策略模式:客户端使用通用接口DiscountStrategy,底层含有各种接口实现类，只需要为客户端配置不同实现类就可以调用不同策略，而客户端代码不用改变。Spring中通过xml可以配置不同策略。和工厂模式的本质区别：工厂是创建不同对象，策略是选择不同行为，共同点：代码构架相同。
6. 门面模式：Service整合不同Dao组件，使得Action只需操作Service就可完成带有逻辑顺序的Dao操作，而不用在Action内部安排Dao顺序。
7. 桥接模式：每个Dao组件对应一个Model，Service组合不同Dao接口可以形成很多不同功能的Service，从而不需要为每一种具体的Dao组合制定一个类。
8. 观察者模式：被观察者中保存着所有观察者对象(可增加可删除)，当被观察者属性改变时，调用相应的观察者的方法。事件机制的实现就是观察者模式。

**架构设计：**

1. 贫血模型：Model作为单纯的数据体，只有属性和setter/getter方法，持久化操作交给Dao，业务逻辑交给Service，前端路由交给Struts2。依靠Spring完成依赖注入。
2. 领域对象模型：Model不仅有属性setter/getter方法，还有一些基本的逻辑操作(不需要持久化)，然后Service层就可以调用这些操作，一定程度上减轻了Service层复杂度。

**实现任务自动调度：借助quartz框架**

配置：下载quart-2.2.2.tar.gz，将lib\quartz-2.2.2.jar&SLF4J相关的jar放到类加载路径WEB-INF/lib默认使用quartz-2.2.2.jar中的quartz.properties作为配置文件，放在WEB-INF/lib下系统自动加载。

步骤：

调度的作业：编写java类(用作任务调度类有规定的格式) 依赖注入所需要的service组件，然后在execute()方法中调用service中自动执行的函数，并在xml中配置成Bean。然后需要包装作业类(如下)

<bean id="payQuartzDetail" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<property name="targetObject" ref="payJob"></property>

<property name="targetMethod" value="execute"></property>

</bean>

触发器：SimpleTrugger(给定时间或者间隔时间重复执行)、CronTrigger(利用Cron表达式：Seconds Minutes Hours DayofMonth Month DayofWeek Year)

利用工厂Bean创建，依赖注入上面包装好的作业Bean

<bean id="cronTriggerPay" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">

<property name="cronExpression" value="0 0 2 1 \* ? \*"></property>

<property name="jobDetail" ref="payQuartzDetail"></property>

</bean>

调度器：里面添加触发器，完成配置

<bean id="" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="triggers">

<list>

<ref bean="cronTriggerPay"></ref>

<ref bean="cronTriggerPunch"></ref>

</list>

</property>

</bean>

**Service层函数设置：**

设置一个接口，里面设置public static final int LOGIN\_MGR = 2 这种数据，因为经常需要利用函数的返回值，这样统一规范。

**Action层设计模式：**

先设计出基类，里面只有需要的service成员变量和setter方法(需要依靠Bean依赖注入这些service)，然后具体的子类拓展自己的功能，在spring中建立模板bean，完成对各个action的注入service。

**利用好Session：使得在整个会话中始终可以访问用户名，方便service操作数据**

可以定义一个接口，如下，在用户登录时LoginAction中把这些数据(用户名，密码等常用)添加到Session中。

public interface WebConstant

//HttpSession里代表经理的level值

String MGR\_LEVEL = "mgr";

//HttpSession里代表员工的level值

String EMP\_LEVEL = "emp";

//HttpSession里代表用户级别的属性名

String LEVEL = "level";

//HttpSession里代表用户名的属性名

String USER = "user";

ctx.getSession().put(WebConstant.USER,manager.getName());

ctx.getSession().put(WebConstant.LEVEL,WebConstant.EMP\_LEVEL);

**Struts2的优势：**

Action.java机制+Interceptor拦截器机制(AOP式编程)

**Hibernate4的优势：**

注解持久化对象(关系映射必须要，否则hql无法面向对象)+ HQL查询语句+数据缓存机制+domian验证策略

**Spring的优势：**

Bean依赖注入+Aop切面编程+依赖注入事务机制(原本是Hibernate中需要)+处理缓存机制

**缓存机制：**

三种缓存是目前缓存实现时通常碰到的三种状况，里面按使用的多少来排序应该是：数据缓存(Hibernate)、页面缓存(chrome)和处理缓存(Spring)；实现的难度上从难到易的顺序应该是：处理缓存、页面缓存、数据缓存；对于系统响应速度提升的效果来说从最好到好的顺序应该是：页面缓存、处理缓存、数据缓存。

第三方缓存：ehcache和redis是同一级别的缓存，hibernate\spring实现缓存都需要借助第三方工具，ehcache轻便，redis功能强大。

**Mysql数据库主键id设置机制：**

逻辑主键（代理主键）：在数据库表中采用一个与当前表中业务逻辑信息无关的字段作为其主键，或称为“伪主键”，一般采用自增长int作为id号。

业务主键（自然主键）：在数据库表中把具有业务逻辑含义的字段作为主键 ;

关系数据库学的最重要的一个理论就是：不要给关键字赋予任何业务意义。假如关键字具有了业务意义，当用户决定改变业务含义，也许他们想要为关键字增加几位数字或把数字改为字母，那么就必须修改相关的关键字。一个表中的主关键字有可能被其他表作为外键。就算是一个简单的改变，譬如在客户号码上增加一位数字，也可能会造成极大的维护上的开销。

使用逻辑主键的原因：

1. 业务主键一旦改变则系统中关联该主键的部分的修改将会是不可避免的，并且引用越多改动越大。
2. 业务主键过大，不利于传输、处理和存储。
3. 使用 int 或者 bigint 作为外键进行联接查询，性能会比以字符串作为外键进行联接查询快。
4. 存在用户或维护人员误录入数据到业务主键中的问题。例如错把 RMB 录入为 RXB ，相关的引用都是引用了错误的数据，一旦需要修改则非常麻烦。如果使用逻辑主键则问题很好解决，如果使用业务主键则会影响到其他表的外键数据，当然也可以通过级联更新方式解决，但是不是所有都能级联得了的。

**Mysql数据库索引机制：**

1、索引目的：索引的目的在于提高查询效率，可以类比字典，如果要查“mysql”这个单词，我们肯定需要定位到m字母，然后从下往下找到y字母，再找到剩下的sql。如果没有索引，那么你可能需要把所有单词看一遍才能找到你想要的

2、索引背景：cache(0.5ns) > main memory(100ns) > disk seek(10,000,000ns = 0.1s)。

数据库中数据都是存放在·磁盘上，你每次查找数据时把磁盘IO次数控制在一个很小的数量级，最好是常数数量级。

1. 索引数据结构：

B+树：非叶子结点不存储真实的数据

1. 索引类型：

普通索引：CREATE INDEX index\_name ON table(column(length))

唯一索引：索引列的值必须唯一，但允许有空值（注意和主键不同）。如果是组合索引，则列值的组合必须唯一，创建方法和普通索引类似：CREATE UNIQUE INDEX indexName ON table(column(length))

组合索引(最左前缀)：ALTER TABLE article ADD INDEX index\_titme\_time (title(50),time(10)) 建立这样的组合索引，其实是相当于分别建立了下面两组组合索引：

–title,time

–title

为什么没有time这样的组合索引呢？这是因为MySQL组合索引“最左前缀”的结果。简单的理解就是只从最左面的开始组合。并不是只要包含这两列的查询都会用到该组合索引，如下面的几个SQL所示：

1–使用到上面的索引

2 SELECT \* FROM article WHREE title='测试' AND time=1234567890;

3 SELECT \* FROM article WHREE utitle='测试';

4–不使用上面的索引

5 SELECT \* FROM article WHREE time=1234567890;

1. 索引原则:
2. 较频繁作为查询条件的字段
3. 唯一性太差的字段，更新太频繁地字段，不会出现在where条件中的字段，不适合建立索引
4. 一般主键就自动形成聚集索引，无法对非主键建立聚集索引，其余缺省情况下建立索引的都是非聚集索引。唯一列默认建立唯一索引。

6、注意事项：

1. like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

1. 如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引

**Config.Properties配置文件:**

可以和spring结合使用，利用spring中工厂获取properties中属性，然后以${path}形式依赖注入给需要的bean 类

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

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

</bean>

2、<bean id="updateVoronoi" class="schedules.UpdateVoronoi" parent="autoTemplate">

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

</bean>

**Maven管理器：**

eclipse建立maven Web项目步骤：

1. 配置项目properties: Project Facets中Java版本修改；Runtimes添加Tomcat；
2. Java Build Path修改：删除原来的Source，然后手动添加如下source folder：src/main/java src/test/java src/test/resources; 再次修改Source,把新添加的folders加入，并修改对应的output路径：main==》classes；test==》test-classes。
3. 修改Deployment Assembly: 把test部分和m2e-webresources部分删除
4. mvn install即可打包成war(普通发布的项目结构一样)发布项目到target文件下；eclipse上也可以直接运行，或者发布到Tomcat服务器下；
5. 添加依赖包最好的方法通过网站搜索，然后复制<dependency>标签即可，Maven会把相关依赖包全部下载(这样不用担心缺少包了)。Maven会先从本地加载，若是没有则从中央仓库加载。
6. pom.xml文件中设置编码方式，否则默认采用平台编码， pom.xml中通过maven-compiler-plugin设置jdk版本，否则使用默认的jdk1.5(用eclipse则这个功能几乎没用，eclipse已经把java编译成class了，maven不用再编译一次)

<properties>

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

</properties>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source> //这是开发版本

<target>1.8</target> //这是编译版本

<encoding>UTF8</encoding>

</configuration>

</plugin>

1. 开启eclipse自带搜索包功能，这需要勾选preferences—Maven—Download repository，然后更新local/global repository(第一次更新很久)；更新完之后就不要在继续勾选了，否则以后每次启动eclipse都会自动先更新maven目录。

**Log4j日志管理工具：SSH整合Log4j是一种经典的Web开发方案**

**优势：**动态的改变记录级别和策略，即修改log4j2.xml,不需要重启Web应用。把log文件定在 /WEB-INF/logs/ 而不需要写绝对路径。

**注意事项：**

1. Log4j2.x以上版本的配置文件只能用log4j2.xml/json(名字都不能换)， 不能再用log4j.properties，系统默认根据Log4j2名字找到对应配置文件，所以不用在web.xml中配置Log4j2.xml的路径。
2. 自定义日志信息时引用如下包，而不是import org.apache.log4j.Logger;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

1. 在log4j2.xml中使用web.xml中配置的项目根目录形式：${sys: Tianjin.root}；不再是原来的${ Tianjin.root}

**配置步骤：**

1. web.xml中声明使用Log4j2，且配置必须在Spring容器之前

<!-- 配置log4j日志管理器，这个必须放在spring配置之前 -->

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

<!-- log4j2.xml文件中写入${sys: Tianjin.root}代表项目路径 -->

<context-param>

<param-name>webAppRootKey</param-name>

<param-value>Tianjin.root</param-value>

</context-param>

1. 编写log4j2.xml配置文件：配置输出信息，位置，格式等。较复杂需参见具体文件
2. 把hibernate.cfg.xml配置文件把 show\_sql 关闭，因为在Log4j中已经声明输出了
3. 类中主动声明Logger类，然后使用logger.info/debug/error/warn 输出自定义的信息，比如记录Dao层的hql/sql语句，记录Service层重要的数据库增删的数据，记录定时器任务工作情况等。