# J2EE

###### 1.绪论

Maven项目结构：src项目资源的总目录，所有操作都在这个文件夹执行

Main：项目的主目录

webAPP：前端部分，包含前端页面，css，js等文件

WEB-INF：java的WEB应用安全目录，客户端无法直接访问，只有服务器端可以直接访问。如果页面想要访问这个目录必须通过web.xml进行映射

Web.xml：沟通前后端的桥梁，配置网站入口，声明哪些文件是后端配置文件

Pom.xml：主要描述maven坐标，包依赖关系

External Libraries：从字面就可以了解，就是外部的一些包。也就是通过Maven下载的包，JDK等等。

###### Spring

IOC：由容器来负责控制对象的生命周期和对象间的关系。控制反转，将设置好的类交给系统去控制

DI：依赖注入。容器动态的将某种依赖关系注入到组件之中。目的是为了提示组件的重用率。

DI的两种方式：

设值注入：使用IOC容器的setter方法注入。<property name=”userDao” ref=”userDao” />.

构造注入：使用构造器来注入被依赖的实例。<constructor-arg ref=”db” />.

设值注入的优点：

1. 设值注入的javabean的写法与传统写法一致，更容易被接受。
2. 对于复杂的依赖关系，如果采用构造注入会导致构造器过于臃肿，spring在创建bean对象的同时要依赖化其他所有实例化对象，导致性能下降。
3. 对于某些属性可选的情况，多参数构造器更复杂。

构造注入的优点：

1. 如果对依赖注入的顺序有要求则可以在构造器种设定依赖注入的顺序
2. 对依赖关系无需变化bean
3. 使用构造器注入，组件内部的依赖关系完全对组件调用者完全透明，更符合高内聚原则。

一般以设值注入为主，构造注入为辅。

Bean的配置管理：

Spring中bean的定义：

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

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

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

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- 配置stoneAxe实例，其实现类是StoneAxe -->

<bean id="stoneAxe" class="org.crazyit.app.service.impl.StoneAxe"/>

<!-- 配置steelAxe实例，其实现类是SteelAxe -->

<bean id="steelAxe" class="org.crazyit.app.service.impl.SteelAxe"/>

</beans>

配置依赖：各组件的相互引用实质就是依赖关系，可以通过spring的IOC容器完成注入。依赖关系可以是一个确定的常亮也可以是对其他bean的引用，通常普通属性不纳入配置管理，主要是配置bean实例之间依赖的关系。

注入普通属性：

<bean id="exampleBean" class="org.crazyit.app.service.ExampleBean">

<!-- 确定int型属性值 -->

<property name="integerProperty" value="1"/>

<!-- 确定double型属性值 -->

<property name="doubleProperty" value="2.3"/>

</bean>

注入bean引用：

<bean id="chinese" class="org.crazyit.app.service.impl.Chinese">

<!-- 将stoneAxe注入给axe属性 -->

<property name=“stoneAxe" ref="stoneAxe"/>

</bean>

<!-- 配置stoneAxe实例，其实现类是StoneAxe -->

<bean id="stoneAxe" class="org.crazyit.app.service.impl.StoneAxe"/>

使用自动装配完成bean的引用：设值bean的autowrie属性

<bean id="auto" class="example.autoBean" autowire="byType"/>

no：不使用自动装配

Byname：根据名字自动装配。在容器中找到id属性与需要注入的属性（setter方法的名字）同名的bean完成注入。

Bytype：根据类型自动匹配

constructor：与byType类似

autodetect：根据bean内部结构决定用constuctor还是byType

注入嵌套bean：

<bean id="exampleBean" class="org.crazyit.app.service.ExampleBean">

<property name="axe">

<bean class="org.crazyit.app.service.impl.StoneAxe"/>

/property>

</bean>

创建bean的实例：

<bean id="chinese" class="org.crazyit.app.service.impl.Chinese">

<property name="axe" ref="steelAxe"/>

</bean>

创建：

ApplicationContext ctx = new ClassPathXmlApplicationContext("bean.xml");

Person p = (Person)ctx.getBean("chinese");

//Person p = ctx.getBean("chinese" , Person.class);

p.useAxe();

Bean的作用域：

Singleton：单例模式，整个容器中只有一个实例

Prototype:原型模式，每次通过getbean获取时创建一个实例

Request：每次http请求产生一个新的实例，需要在web.xml中配置listener

Session：每次http session产生一个新的实例

Bean的生命周期：

Prototype bean每次请求都会产生一个新的实例对象，spring无法跟踪其生命周期。

singleton bean：单例对象，每次客户请求都返回同一个共享实例，spring可以跟踪其创建和销毁，实现对其生命周期的管理

生命周期的管理时机：

注入之后---init method

在bean的配置中加入inti-method属性指定全部属性注入/设值成功后要执行的方法

<bean id="chinese" class="org.j2ee.service.impl.Chinese"

init-method="init">

<property name="axe" ref="steelAxe"/>

</bean>

Bean销毁之前

占用资源的回收

在<bean>的配置中加入destroy-method属性指定bean在销毁之前要执行的方法

<bean id="chinese" class="org.j2ee.service.impl.Chinese"

destroy-method=“destroy">

<property name="axe" ref="steelAxe"/>

</bean>

Singleton bean依赖Prototype bean作用域不同步，原因是Singleton bean只会创建一次之后容器内共享，而Prototype bean每次请求都会创建一个新的实例对象

解决方案

放弃容器注入，singleton bean每次使用prototype bean时主动从服务器请求新的prototype bean。耦合了SpringAPI

利用方法注入：使用lookup方法注入

Spring AOP的基本特点

Spring AOP的基本概念：

关注点：所关注的公共功能，表示“要做什么”

连接点：程序执行过程中某个特定的点，在spring AOP 中代表要执行某个业务方法的点。“在什么地方执行”

增强处理：在连接点上执行的动作包括包括“around”、“before”、 “after”和“Throws”等类型。Spring以拦截器做增强处理模型，并维护一个以连接点为中心的拦截器链。表示“具体怎么做”

切面：一个关注点的模块化，综合表示“在什么地方，做什么，怎么做”

织入（Weaving）：

把切面连接到其它的应用程序类型或者对象上，并创建一个被增强处理的对象的过程。

Spring AOP 事务增强的五种类型：

前置增强：在目标方法执行前进行增强

@Before(value = "execution(\* aop.annotation.\*.\*(..))")

后置增强：在目标方法执行后进行增强

@AfterReturning(value = "anyMethod()", returning = "result")

环绕增强：在目标方法之前前后进行增强，直接使用了AOP联盟定义的接口

@Around(value = "execution(\* aop.annotation.\*.\*(..))")

异常抛出增强：在目标方法抛出异常之后进行增强

@AfterThrowing(value = "execution(\* aop.annotation.\*.\*(..))", throwing = "e")

引介增强：在目标方法中加入新的属性和方法

基于注解的AOP配置：

/\*\*

\* 目标类。同上，很普通的一个类，实际开发中往往是一个业务类

\*/

public class Target {

public void say() {

System.out.println("我还要被增强！");

}

}

/\*\*

\* 增强类，使用注解开发，该类同样不再继承任何接口

\*

\* @Aspect用于将类标识为一个切面

\* @Before("execution(...)")用于将方法标识为增强方法，"切入点表达式"用于定位被切入方法。

\*/

@Aspect

public class Advice {

@Before("execution(\* \*..Target.\*(..))")

public void beforeAdvice() {

System.out.println("我是前置增强，很高兴为你服务！");

}

}

Xml配置

<aop:aspectj-autoproxy/>

<bean id="target" class="com.jimmy.mvn.a.BeforeAdviceAnnotation.Target"></bean>

<bean id="advice" class="com.jimmy.mvn.a.BeforeAdviceAnnotation.Advice"></bean>

###### 3.Hibernate

Hibernate的基本原理：hibernate是对JDBC的进一步封装。Hibernate属于持久层框架。Hibernate是开源的一个ORM（对象关系映射）框架。

Hibernate项目开发步骤：

1. 新建maven项目，创建数据库
2. 创建hibernate配置文件hibernate.cfg.xml
3. 创建持久化类

public class User {

private String id;

private String name;

private String password;

…..getters/setters

}

1. 创建类的映射文件

<hibernate-mapping>

<class name="com.bjsxt.hibernate.User“ table=“tuser”>

<id name="id">

<generator class="uuid"/>

</id>

<property name="name"/>

<property name="password"/>

……

</class>

</hibernate-mapping>

1. 将类的映射文件加入hibernate

在Hibernate配置文件中加入：

<mapping resource="com/bjsxt/hibernate/User.hbm.xml“/>

1. 将对象保存到数据库

public static void main(String[] args) throws Exception{

Configuration cfg = new Configuration().configure();

SessionFactory factory = cfg.buildSessionFactory();

Session session = factory.openSession();

session.beginTransaction();

User user = new User();

user.setName("管理员");

user.setPassword("admin");

session.save(user);

session.getTransaction().commit();

if(session.isOpen()){

session.close();

}

}

总结：

创建项目

加入hibernate的jar包

加入数据库的jar包

在src/hibernate.cfg.xml中配置数据库基本信息

将每张表映射成一个javabean

创建javabean

为javabean创建对应的beanname.hbm.xml配置文件

在配置文件中建立bean的属性与对应表字段的映射

在cfg.xml中为每一个hbm.xml配置mapping元素

利用javabean操作数据库

构建sessionFactory

用sessionFactory创建session

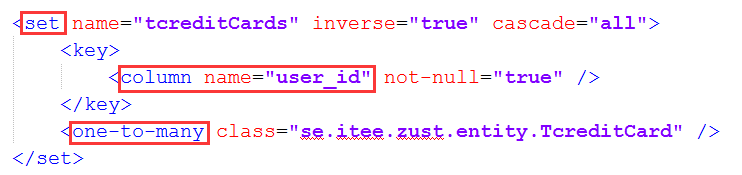
用session通过java对象操作数据库

Hibernate双向1对多映射配置：

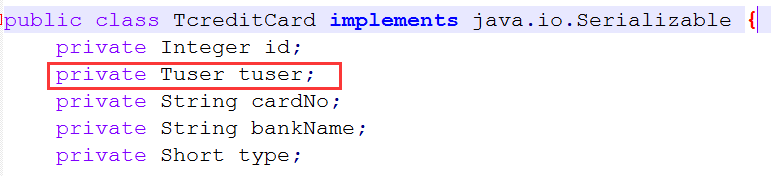
1端：1端持久化类中增加N端持久化类的集合



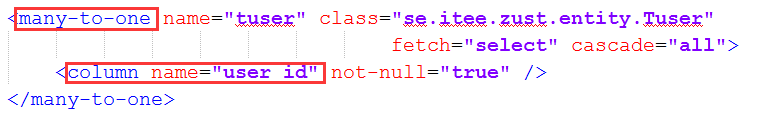
关联映射



N端：N端持久化类中增加1端持久化对象的引用



关联映射



持久化对象的状态与生命周期：

瞬时对象：使用new操作初始化的对象不是立刻就持久的，它们还没有与任何数据库关联。

持久化对象：持久实力是具有数据库表示的实例，由session统一管理，它们的状态在事务提交时与数据库同步。

离线对象：session关闭后持久化对象就变成离线对象，不再与数据库保持同步。

基本hql语句：

简单例子：查询用户名以“J”开头的所有用户。

Query query = session.createQuery(

"from User user where user.name like 'J%'");

List<User> users = query.list();

复杂例子：从User和Group中查找属于“admin”组的所有用户。

Query query = session.createQuery(

“from User user where user.group.name=‘admin’”);

###### 4.springMVC

Spring mvc的请求处理及响应的处理全流程：

1. DispatcherServlet前端控制器接受请求，交给HandleMapping处理器映射器
2. HandleMapping根据请求路径找到相应的HanderAdapter处理器适配器（也就是controller）
3. HandleMpping处理的功能请求返回一个ModelAndView对象
4. ViewResolver视图解析器先根据ModelAndView中设置的view解析具体视图，然后再将model中的数据渲染到view上

SpringMVC控制器的分类与作用：

前端控制器:DispatcherServlet

应用控制器：映射管理器（HandleMapping）+视图解析器（ViewResolver）

页面控制器：Controller

SpringMVC的URL映射：

@Controller

@RequestMapping("/user")

public class HelloController {

@RequestMapping(value="/hello")

public String sayHello() {

System.out.println("hello spring mvc 哈哈哈");

return "success"; //WEB-INF/jsps/success.jsp

}

}

SpringMVC的参数绑定/命令对象：

@RequestMapping(value="/delete", params="userId")

public String test1(@RequestParam("userId") String userId){

...

}

params和headers分别通过请求参数及报文头属性进行映射，它们支持简单的表达式，下面以params表达式为例说明，headers可以参照params进行理解之。

"param1"：表示请求必须包含名为param1的请求参数。

"!param1"：表示请求不能包含名为param1的请求参数。

"param1!=value1"：表示请求包含名为param1的请求参数，但其值不能为value1。

{"param1=value1","param2"}：请求必须包含名为param1和param2的两个请求参数，且param1参数的值必须为value1。

SpringMVC与servlet API：

可以通过@Autowired将HttpServletRequest、HttpServletResponse和HttpSession对象自动注入进来

@Autowired

HttpServletRequest request;

@Autowired

HttpServletResponse response;

@RequestMapping(value = "/handle21")

public void handle21() {

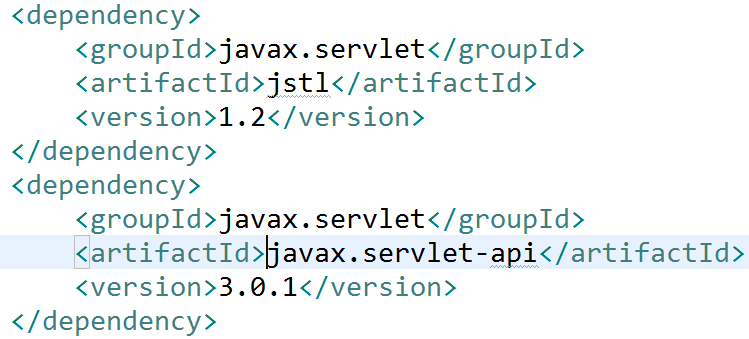
String userName = WebUtils.findParameterValue(request, "userName");

response.addCookie(new Cookie("userName", userName));

}

SpringMVC的视图解析：

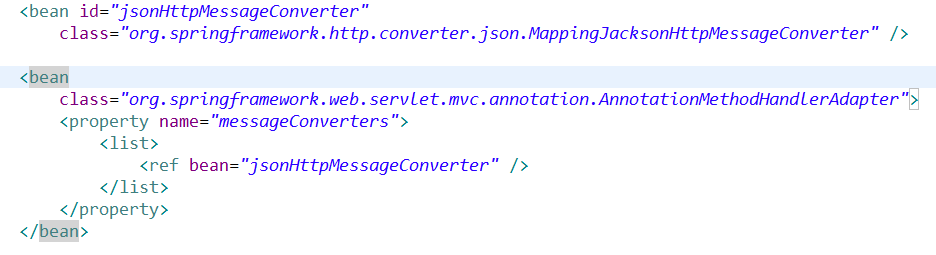
在pom.xml中添加jstl和servlet依赖



SpringMVC的json数据支持：

Xml配置： messageConverters（在pom中加上jar包依赖）

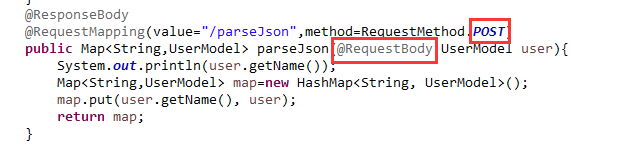
dispatcher-servlet.xml



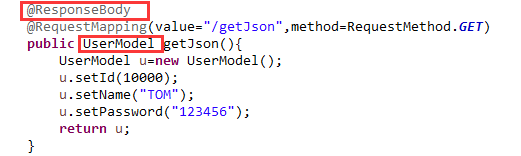
输出json——@ResponseBody

输入json——@RequestBody

用Jason发送数据请求：

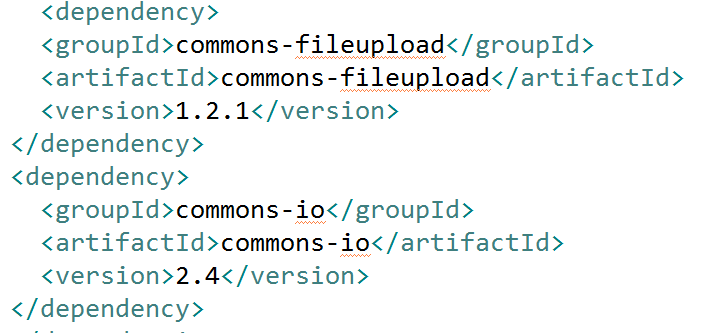


返回Jason：



SpringMVC文件上传：

导入jar包依赖：



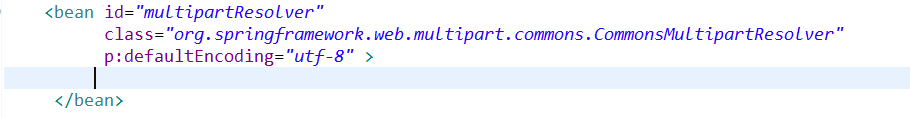
页面：



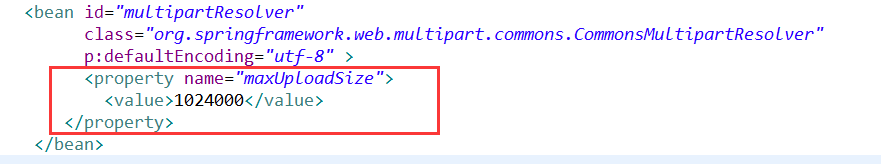
Controller方法：



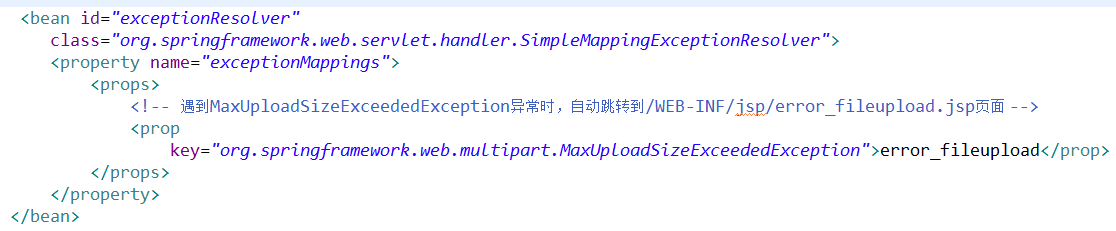
文件上传配置：



文件大小限制：



异常处理：



###### 5.SSH整合

SSH整合的基本思路：

利用springMVC来处理用户的请求和响应。利用spring容器管理各类bean并处理业务和依赖、利用hibernate提供数据服务。

WEB层：处理用户请求并响应，需要业务层bean提供业务支持。

业务层：为WEB层的controller方法提供业务支持，需要数据层bean提供支持。

数据层：为业务层的bean方法提供数据支持，需要提供sessionFactory来管理好实用映射对象。

SpringMVC天然与spring整合。

Hibernate与spring整合：

关键是sessionFatory

将dataSource和sessionFactory作为bean配置到spring中

Dao中注入sessionFaction

将Dao通过@Respository加入到spring容器

整合步骤：

在WEB中启动spring容器

启动监听器

将所有业务bean通过注解添加到容器中

包扫描

Controller用@controller注解

Service用@service注解和@transaction注解

Dao用@repository注解

Bean配置（dataSource和sessionFactory）

Spring容器与SpringMVC容器的关系：

Spring的容器由listener启动，为父容器。

SpringMVC容器由dispatcherServlet创建，为子容器。

SpringMVC在配置controller时可以直接使用在spring中配置的service。

在web.xml配置中要先配置spring然后配置springMVC，否则springMVC子容器无法使用父容器中的资源。

父容器中的bean无法使用子容器中的bean。