**1.为什么使用Spring ?**

1). 方便解耦，简化开发

         通过Spring提供的IoC容器，可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。

    2). AOP编程的支持

         通过Spring提供的AOP功能，方便进行面向切面的编程，如性能监测、事务管理、日志记录等。

    3). 声明式事务的支持

    4). 方便集成各种优秀框架

    5). 降低Java EE API的使用难度

         如对JDBC，JavaMail，远程调用等提供了简便封装。

**2. 什么是IoC，为什使用IoC ?**

 IoC全称Iversion of Controller，控制反转。

以前创建和管理对象由程序代码完成，现在创建对象由容器完成

这种控制权的转移，称为控制反转。。

**3.什么是AOP，为什么使用AOP ?**

AOP全称：Aspect-Oriented Programming，面向切面编程。

     AOP，面向切面编程，就是把可重用的功能提取出来，然后将这些通用功能在合适的时候织入到应用程序中，比如**事务管理**、**权限控制**、**日志记录**、性能统计等。

**AOP并没有帮助我们解决任何新的问题，它只是提供了一种更好的办法，能够用更少的工作量来解决现有的一些问题，使得系统更加健壮，可维护性更好。**

**4.什么是Spring的事务管理？**

事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。这样可以防止出现脏数据，防止数据库数据出现问题。

    开发中为了避免这种情况一般都会进行事务管理。

**Spring的声明式事务通常是指在配置文件中对事务进行配置声明**，其中包括了很多声明属性，它是通过Spring Proxy帮你做代理，自己不用额外的写代码，只要在Spring配置文件中声明即可；通常用在数据库的操作里面；

**编程式事务就是指通过硬编码的方式做事务处理**，这种处理方式需要写代码，事务中的逻辑可以自己定制；可以是数据库的东东，也可以是其他的操作。

**Spring中也有自己的事务管理机制，一般是使用TransactionMananger进行管理，**可以通过Spring的注入来完成此功能。

**5.Spring框架支持以下五种bean的作用域?**

**singleton :** 默认值，bean在每个Spring ioc 容器中只有一个实例。

**prototype：**一个bean的定义可以有多个实例。

**request：**每次http请求都会创建一个bean，该作用域仅在基于web的Spring ApplicationContext情形下有效。

**session：**在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

**global-session：**在一个全局的HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

**6.什么是Spring的MVC框架？**

 Spring 配备构建Web 应用的全功能MVC框架。Spring可以很便捷地和其他MVC框架集成，如Struts，Spring 的MVC框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

    spring mvc是一个基于mvc的web框架。spring mvc是spring框架的一个模块，springmvc和spring无需通过中间整合层进行整合。

**7.如何启用注解?**

<context:annotation-config/>

    如果使用<context:component-scan base-package="com.tgb.web.controller.annotation"> </context:component-scan>  则上面内容可以省略

**8.Spring MVC的请求流程**?

    第一步：发起请求到**前端控制器**(DispatcherServlet)

    第二步：前端控制器请求**HandlerMapping**查找Handler可以根据xml配置、注解进行查找

    第三步：处理器映射器HandlerMapping向前端控制器返回**Handler**

    第四步：前端控制器调用**处理器适配器**去执行Handler

    第五步：处理器适配器去执行Handler

    第六步：Handler执行完成给适配器返回**ModelAndView**

    第七步：处理器适配器向前端控制器返回ModelAndView。ModelAndView是springmvc框架的一个底层对象，包括 Model和view

    第八步：前端控制器请求**视图解析器**去进行视图解析，根据逻辑视图名解析成真正的视图(jsp)

    第九步：视图解析器向前端控制器**返回View**

    第十步：前端控制器进行**视图渲染**。视图渲染将模型数据(在ModelAndView对象中)填充到request域

    第十一步：前端控制器向用户响应结果

**9.web.xml的配置**



**10.注解的处理器映射器和适配器?**

spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping注解映射器。

在spring3.1之后使用org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter注解适配器。

**使用 mvc:annotation-driven代替上边注解映射器和注解适配器配置**

**11.spring 与 mybatis整合过程?**

第一步：整合dao层

   mybatis和spring整合，通过spring管理mapper接口。

   使用mapper的扫描器自动扫描mapper接口在spring中进行注册。

第二步：整合service层

   通过spring管理 service接口。

   使用配置方式将service接口配置在spring配置文件中。

   实现事务控制。

第三步：整合springmvc

   由于springmvc是spring的模块，不需要整合

**主要配置有：**

      1). mybatis配置文件sqlMapConfig.xml配置别名自动扫描**(实体类)**

      2). mapper扫描器**(接口，数据库访问接口)**

      3). 数据库连接池配置

      4). 声明式事务配置

      5). 启用注解扫描：<context:component-scan base-package="cn.itcast.ssm.controller"></context:component-scan>

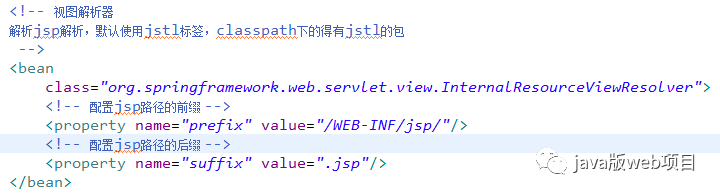
      6). 配置注解映射器和适配器： <mvc:annotation-driven></mvc:annotation-driven>

      7). 视图解析器：<bean  class="org.springframework.web.servlet.view.InternalResourceViewResolver">

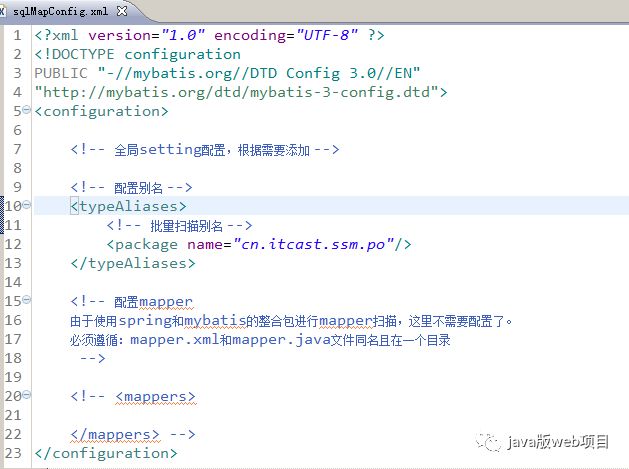
      8). 配置控制类： DispatcherServlet前端控制器

      9). 配置spring配置文件加载类：ClassLoadListener

**12.视图解析器配置前缀和后缀?**



**13.sqlMapConfig.xml，mybatis自己的配置文件?**



**14.配置数据源?**

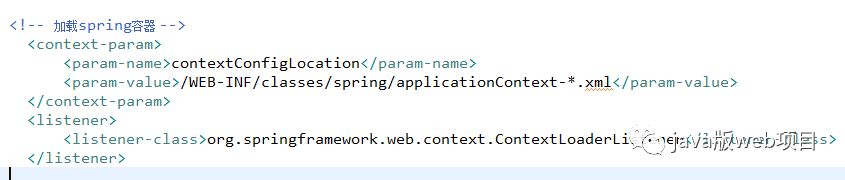


**15.事务控制(applicationContext-transaction.xml)?**

在applicationContext-transaction.xml中使用spring声明式事务控制方法。



**16.加载spring配置?**



**17.静态资源访问不被拦截?**

 <resources mapping="/resources/\*\*" location="/resources/" />

      <resources mapping="/images/\*\*" location="/images/" />

      <resources mapping="/js/\*\*" location="/js/" />

**18.@RequestMapping的作用?**

1). url映射

2). 窄化请求映射

3). 限制http请求方法

**19.controller方法的返回值?**

**1 返回ModelAndView**

    需要方法结束时，定义ModelAndView，将model和view分别进行设置。

**2 返回string**

     如果controller方法返回string，

     1). 表示返回逻辑视图名。真正视图(jsp路径)=前缀+逻辑视图名+后缀

     2). redirect重定向：返回字符串格式为："redirect:queryItem.action"

3). forward页面转发：返回字符串格式为：“forward:queryItem.action”

**3 返回void**

      在controller方法形参上可以定义request和response，使用request或 response指定响应结果：

     1). 使用request转向页面，如下：request.getRequestDispatcher("页面路径").forward(request, response);

     2). 也可以通过response页面重定向：response.sendRedirect("url")

     3). 也可以通过response指定响应结果，例如响应json数据如下：

           response.setCharacterEncoding("utf-8");

           response.setContentType("application/json;charset=utf-8");

           response.getWriter().write("json串");

**20.参数绑定**

**1 默认支持的类型**

直接在controller方法形参上定义下边类型的对象，就可以使用这些对象。在参数绑定过程中，如果遇到下边类型直接进行绑定。

1). HttpServletRequest：通过request对象获取请求信息

2). HttpServletResponse：通过response处理响应信息

3). HttpSession：通过session对象得到session中存放的对象

4). Model/ModelMap：model是一个接口，modelMap是一个接口实现 。作用：将model数据填充到request域。

**2 简单类型**

通过@RequestParam对简单类型的参数进行绑定。

**如果不使用@RequestParam，要求request传入参数名称和controller方法的形参名称一致，方可绑定成功。**

**如果使用@RequestParam，不用限制request传入参数名称和controller方法的形参名称一致。**

通过required属性指定参数是否必须要传入，如果设置为true，没有传入参数，会报错。

**3 pojo绑定**

**页面中input的name和controller的pojo形参中的属性名称一致，将页面中数据绑定到pojo。(usename,age；不需要user.username,user.age)**

**4 自定义参数绑定实现日期类型绑定**

对于controller形参中pojo对象，如果属性中有日期类型，需要自定义参数绑定。将请求日期数据串转成 日期类型，要转换的日期类型和pojo中日期属性的类型保持一致。

**21.Spring MVC 和 Struts2 对比?**

1). Struts2是类级别的拦截， 一个类对应一个request上下文，SpringMVC是方法级别的拦截，一个方法对应一个request上下文，而方法同时又跟一个url对应，所以说从架构本身上SpringMVC 就容易实现restful url

2). 由上边原因，SpringMVC的方法之间基本上独立的，独享request response数据，请求数据通过参数获取，处理结果通过ModelMap交回给框架，方法之间不共享变量，而Struts2搞的就比较乱，虽然方法之间也是独立的，但其所有Action变量是共享的，这不会影响程序运行，却给我们编码 读程序时带来麻烦，每次来了请求就创建一个Action，一个Action对象对应一个request上下文。

3). 由于Struts2需要针对每个request进行封装，把request，session等servlet生命周期的变量封装成一个一个Map，供给每个Action使用，并保证线程安全，所以在原则上，是比较耗费内存的。

4). SpringMVC集成了Ajax，使用非常方便，只需一个注解@ResponseBody就可以实现，然后直接返回响应文本即可，而Struts2拦截器集成了Ajax，在Action中处理时一般必须安装插件或者自己写代码集成进去，使用起来也相对不方便。

5). springmvc面向方法开发的（更接近service接口的开发方式），struts2面向类开发。

6). springmvc可以单例开发，struts2只能是多例开发。

**22. 乱码处理?**

1). post乱码

     在web.xml添加post乱码filter：CharacterEncodingFilter

2). 对于get请求中文参数出现乱码解决方法有两个：

     a. 修改tomcat配置文件添加编码与工程编码一致，如下：

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

     b. 对参数进行重新编码：

String userName = new String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码

**23. 集合类型绑定**

1). 数组绑定：

     controller方法参数使用：(Integer[] itemId)

     页面统一使用:itemId 作为name

2). list绑定：

     pojo属性名为：itemsList

     页面：itemsList[index].属性名

3). map 绑定：

     pojo属性名为：Map<String, Object> itemInfo = new HashMap<String, Object>();

     页面： <td>姓名：<inputtype=*"text"*name=*"itemInfo['name']"*/>

**24.spring 校验 ?**

1). 项目中，通常使用较多是前端的校验，比如页面中js校验。对于安全要求较高点建议在服务端进行校验。

2). springmvc使用hibernate的校验框架validation(和hibernate没有任何关系)。

校验思路：页面提交请求的参数，请求到controller方法中，使用validation进行校验。如果校验出错，将错误信息展示到页面。

**25.数据回显?**

1). @ModelAttribute还可以将方法的返回值传到页面：在方法上加注解@ModelAttribute

2). 使用最简单方法使用model，可以不用@ModelAttribute：model.addAttribute("id", id);

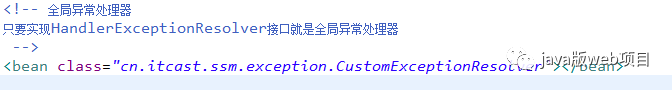
3). springmvc默认对pojo数据进行回显。pojo数据传入controller方法后，springmvc自动将pojo数据放到request域，key等于pojo类型（首字母小写）

4). public String testParam(PrintWriter out, @RequestParam("username") String username) { //out直接输出

**26.异常处理?**

springmvc提供全局异常处理器（一个系统只有一个异常处理器）进行统一异常处理。

**系统遇到异常，在程序中手动抛出，dao抛给service、service给controller、controller抛给前端控制器，前端控制器调用全局异常处理器。**



**27.上传图片?**

1). 在页面form中提交enctype="multipart/form-data"的数据时，需要springmvc对multipart类型的数据进行解析。

2). 在springmvc.xml中配置multipart类型解析器。



3). 方法中使用：MultipartFile attach (单个文件上传) 或者  MultipartFile[] attachs (多个文件上传)

**28.Json处理**

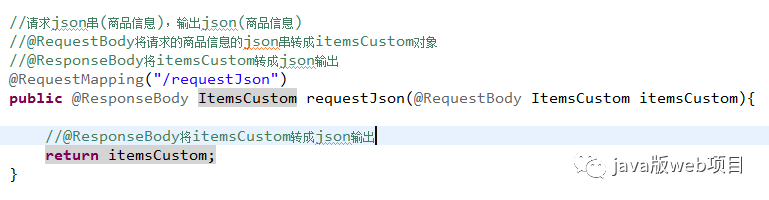
1). 加载json转换的jar包：springmvc中使用jackson的包进行json转换（@requestBody和@responseBody使用下边的包进行json转）

2). 配置json转换器。在注解适配器RequestMappingHandlerAdapter中加入messageConverters。**如果使用<mvc:annotation-driven /> 则会自动加入。**

3). ajax



4). Controller**(ResponseBody、RequestBody)**



5). 注意ajax中contentType如果不设置为json类型，则传的参数为key/value类型。**上面设置后，传的是json类型。**

**29.拦截器?**

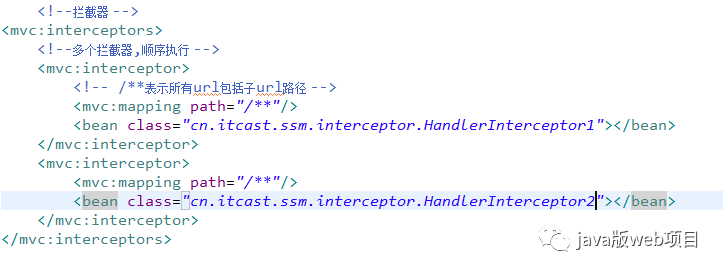
**1). 定义拦截器，实现HandlerInterceptor接口。接口中提供三个方法。**

a. preHandle ：进入 Handler方法之前执行，用于身份认证、身份授权，比如身份认证，如果认证通过表示当前用户没有登陆，需要此方法拦截不再向下执行

b. postHandle：**进入Handler方法之后，返回modelAndView之前执行**，应用场景从modelAndView出发：将公用的模型数据(比如菜单导航)在这里传到视图，也可以在这里统一指定视图

c. afterCompletion：执行Handler完成执行此方法，应用场景：统一异常处理，统一日志处理

**2). 拦截器配置：**



a. 针对HandlerMapping配置(不推荐)：springmvc拦截器针对HandlerMapping进行拦截设置，如果在某个HandlerMapping中配置拦截，经过该 HandlerMapping映射成功的handler最终使用该 拦截器。  (一般不推荐使用)

b. 类似全局的拦截器：springmvc配置类似全局的拦截器，springmvc框架将配置的类似全局的拦截器注入到每个HandlerMapping中

**30.spring中自动装配的方式有哪些？**

1、No：即不启用自动装配。

2、byName：通过属性的名字的方式查找JavaBean依赖的对象并为其注入。比如说类Computer有个属性printer，指定其autowire属性为byName后，Spring IoC容器会在配置文件中查找id/name属性为printer的bean，然后使用Seter方法为其注入。

3、byType：通过属性的类型查找JavaBean依赖的对象并为其注入。比如类Computer有个属性printer，类型为Printer，那么，指定其autowire属性为byType后，Spring IoC容器会查找Class属性为Printer的bean，使用Seter方法为其注入。

4、constructor：通byType一样，也是通过类型查找依赖对象。与byType的区别在于它不是使用Seter方法注入，而是使用构造子注入。

5、autodetect：在byType和constructor之间自动的选择注入方式。

6、default：由上级标签<beans>的default-autowire属性确定。

**31.Spring中AOP的应用场景、Aop原理、好处？**

AOP--Aspect Oriented Programming面向切面编程；用来封装横切关注点，具体可以在下面的场景中使用:

Authentication 权限、Caching 缓存、Context passing 内容传递、Error handling 错误处理Lazy loading懒加载、Debugging调试、logging, tracing, profiling and monitoring 记录跟踪优化　校准、Performance optimization　性能优化、Persistence 持久化、Resource pooling　资源池、Synchronization　同步、Transactions 事务。

**原理：**AOP是面向切面编程，是通过动态代理的方式为程序添加统一功能，集中解决一些公共问题。

**优点：**1.各个步骤之间的良好隔离性耦合性大大降低   
           2.源代码无关性，再扩展功能的同时不对源码进行修改操作

**32.Spring中IOC的作用与原理？对象创建的过程？**

IOC--Inversion of Control控制反转。当某个角色需要另外一个角色协助的时候，在传统的程序设计过程中，通常由调用者来创建被调用者的实例对象。但在spring中创建被调用者的工作不再由调用者来完成，因此称为控制反转。创建被调用者的工作由spring来完成，然后注入调用者 直接使用。

**33.Spring常见创建对象的注解？**

@Component@Controller@ Service@ Repository

**34.Spring中用到的设计模式？**

简单工厂、工厂方法、单例模式、适配器、包装器、代理、观察者、策略、模板方法

**35.Spring的优点？**

1.降低了组件之间的耦合性 ，实现了软件各层之间的解耦   
2.可以使用容易提供的众多服务，如事务管理，消息服务等   
3.容器提供单例模式支持   
4.容器提供了AOP技术，利用它很容易实现如权限拦截，运行期监控等功能   
5.容器提供了众多的辅助类，能加快应用的开发   
6.spring对于主流的应用框架提供了集成支持，如hibernate，JPA，Struts等   
7.spring属于低侵入式设计，代码的污染极低   
8.独立于各种应用服务器   
9.spring的DI机制降低了业务对象替换的复杂性   
10.Spring的高度开放性，并不强制应用完全依赖于Spring，开发者可以自由选择spring的部分或全部

**36.Spring Bean的作用域之间有什么区别？**

Spring容器中的bean可以分为5个范围。所有范围的名称都是自说明的，但是为了避免混淆，还是让我们来解释一下：

singleton：这种bean范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个bean的实例，单例的模式由bean factory自身来维护。

prototype：原形范围与单例范围相反，为每一个bean请求提供一个实例。

request：在请求bean范围内会每一个来自客户端的网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。

Session：与请求范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

global-session：global-session和Portlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多portlet。如果你想要声明让所有的portlet共用全局的存储变量的话，那么这全局变量需要存储在global-session中。

全局作用域与Servlet中的session作用域效果相同。

**37.Spring管理事务有几种方式？**

有两种方式：

1、编程式事务，在代码中硬编码。(不推荐使用)

2、声明式事务，在配置文件中配置（推荐使用）

声明式事务又分为两种：

a、基于XML的声明式事务

b、基于注解的声明式事务

**38.spring中的核心类有那些，各有什么作用？**

BeanFactory：产生一个新的实例，可以实现单例模式

BeanWrapper：提供统一的get及set方法

ApplicationContext:提供框架的实现，包括BeanFactory的所有功能

**39.Bean的调用方式有哪些？**

有三种方式可以得到Bean并进行调用：  
1、使用BeanWrapper  
HelloWorld hw=new HelloWorld();   
BeanWrapper bw=new BeanWrapperImpl(hw);   
bw.setPropertyvalue(”msg”,”HelloWorld”);  
system.out.println(bw.getPropertyCalue(”msg”));

2、使用BeanFactory  
InputStream is=new FileInputStream(”config.xml”);  
XmlBeanFactory factory=new XmlBeanFactory(is);  
HelloWorld hw=(HelloWorld) factory.getBean(”HelloWorld”);  
system.out.println(hw.getMsg()); 

3、使用ApplicationConttext  
ApplicationContext actx=new FleSystemXmlApplicationContext(”config.xml”);  
HelloWorld hw=(HelloWorld) actx.getBean(”HelloWorld”);  
System.out.println(hw.getMsg());

**40.什么是IOC，什么又是DI，他们有什么区别？**

依赖注入DI是一个程序设计模式和架构模型， 一些时候也称作控制反转，尽管在技术上来讲，依赖注入是一个IOC的特殊实现，依赖注入是指一个对象应用另外一个对象来提供一个特殊的能力，例如：把一个 数据库连接已参数的形式传到一个对象的结构方法里面而不是在那个对象内部自行创建一个连接。控制反转和依赖注入的基本思想就是把类的依赖从类内部转化到外 部以减少依赖

应用控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用，传递给它。也可以说，依赖被注入到对象中。所以，控制反转是，关于一个对象如何获取他所依赖的对象的引用，这个责任的反转。

**41.spring有两种代理方式？**

若目标对象实现了若干接口，spring使用JDK的java.lang.reflect.Proxy类代理。

      优点：因为有接口，所以使系统更加松耦合

      缺点：为每一个目标类创建接口

若目标对象没有实现任何接口，spring使用CGLIB库生成目标对象的子类。

      优点：因为代理类与目标类是继承关系，所以不需要有接口的存在。

      缺点：因为没有使用接口，所以系统的耦合性没有使用JDK的动态代理好。

**42.springMVC的流程？**

1.用户发送请求至前端控制器DispatcherServlet

2.DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3.处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

4.DispatcherServlet通过HandlerAdapter处理器适配器调用处理器

5.执行处理器(Controller，也叫后端控制器)。

6.Controller执行完成返回ModelAndView

7.HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8.DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9.ViewReslover解析后返回具体View

10.DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。

11.DispatcherServlet响应用户

**43.Springmvc的优点？**

1.它是基于组件技术的.全部的应用对象,无论控制器和视图,还是业务对象之类的都是 java组件.并且和Spring提供的其他基础结构紧密集成.

2.不依赖于Servlet API(目标虽是如此,但是在实现的时候确实是依赖于Servlet的)

3. 可以任意使用各种视图技术,而不仅仅局限于JSP

4 . 支持各种请求资源的映射策略

5 .它应是易于扩展的