1. **前端**
2. **get与post提交的区别**

Get一般用来从服务器上查询获取资源；Post一般用来更新服务器上的资源；

Get方法将参数直接拼接在了URL后边，明文显示，可以通过浏览器地址栏直接访问；

Post请求用于提交表单，数据不是明文的，安全性更高；

Get请求有长度限制，Post请求没有

1. jquery中$.get() 提交和$.post() 提交的区别

1. $.get() 方法使用 GET 方式提交请求,而$.post()使用 POST 方式。

2. GET 方式传输的数据大小不能超过 2KB 而 POST 要大的多

3. GET 方式请求的数据会被浏览器缓存起来，因此有安全问题。

1. 你对Aajax的理解

AJAX 全称： 异步JavaScript及 XML。

Ajax的核心是JavaScript中的XmlHttpRequest(XHR)。

使用ajax可以提高用户的体验度，进行异步数据传输从而

提高性能。ajax不能跨域。

dataType:'jsonp',可以通过设置动态的<script>标签

所谓的不能跨域就是不能跨多个网站(多个域名),不能跨多个项目。

可以通过jsonp来解决ajax跨域的问题，而jsonp的实质就是通过动态添加script标签来 现的。

$.ajax({

url: some.php,

async: false,

success : function(){

}

});

1. **Java对XML文件的处理方式**

|  |  |
| --- | --- |
| **DOM解析** | **SAX解析** |
| 原理： 一次性加载xml文档，不适合大容量的文件读取 | 原理： 加载一点，读取一点，处理一点。适合大容量文件的读取 |
| DOM解析可以任意进行增删改查 | SAX解析只能读取 |
| DOM解析任意读取任何位置的数据，甚至往回读 | SAX解析只能从上往下，按顺序读取，不能往回读 |
| DOM解析面向对象的编程方法（Node，Element，Attribute）,Java开发者编码比较简单。 | SAX解析基于事件的编程方法。java开发编码相对复杂。 |

1. 说说你对 JSON的理解

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于标准JavaScript 的一个子集,是一个 Js 对象或数组结构的字符串。

JSON 有三类数据

1. 单个数据

有 number, string, boolean 和 null 四种类型数据

2. 多个有序的数据: 数组

用[]包含起来, 其元素可以是三类数据中的任意一种, 元素之间用,号隔开

3. 多个无序的数据: 对象

用{}包含起来, 其元素必须由 key-value 组成, key 是一个字符串, value 可以是任意类型数据, key 与 value 之间用:号隔开, 两个 key-value 之间用,号隔开.

1. **JavaWeb**
2. **tomcat配置优化**

1、server.xml文件中禁用ajp协议（新版中默认是屏蔽的）,减少不必要的线程开销

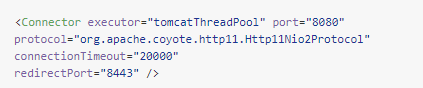
<!--<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />-->

2、server.xml文件修改元素，使用线程池提高性能



3、server.xml文件中修改连接器，可以使用NIO2通道提高性能

不能在tomcat7上配置NIO2通道，因为NIO2虽然性能高，但是必须是tomcat8及以上版本才可以配置。



1. **JSP**
2. **在JSP中提供了四种属性保存范围**

在一个页面范围内:pagecontext

在一次服务器请求范围内:request

在一次会话范围内:session

在一个应用服务器范围内:application

Request是客户端向服务端发送请求

Response是服务端对客户端请求做出响应

Session在servlet中不能直接使用，需要通过getSession()创建，如果没有设定它的生命周期，或者通过invildate()方法销毁，关闭浏览器session就会消失

Application不能直接创建，存在于服务器的内存中，由服务器创建和销毁

1. [**HTTP是不保存状态的协议,如何保存用户状态?**](https://snailclimb.gitee.io/javaguide/#/docs/network/计算机网络?id=%e4%b9%9d-http%e6%98%af%e4%b8%8d%e4%bf%9d%e5%ad%98%e7%8a%b6%e6%80%81%e7%9a%84%e5%8d%8f%e8%ae%ae%e5%a6%82%e4%bd%95%e4%bf%9d%e5%ad%98%e7%94%a8%e6%88%b7%e7%8a%b6%e6%80%81)

HTTP 是一种不保存状态，即无状态（stateless）协议。也就是说 HTTP 协议自身不对请求和响应之间的通信状态进行保存。那么我们保存用户状态呢？Session 机制的存在就是为了解决这个问题，Session 的主要作用就是通过服务端记录用户的状态。典型的场景是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。服务端给特定的用户创建特定的 Session 之后就可以标识这个用户并且跟踪这个用户了（一般情况下，服务器会在一定时间内保存这个 Session，过了时间限制，就会销毁这个Session）。

在服务端保存 Session 的方法很多，最常用的就是内存和数据库(比如是使用内存数据库redis保存)。既然 Session 存放在服务器端，那么我们如何实现 Session 跟踪呢？大部分情况下，我们都是通过在 Cookie 中附加一个 Session ID 来方式来跟踪。

Cookie 被禁用怎么办?

最常用的就是利用 URL 重写把 Session ID 直接附加在URL路径的后面。

1. **Cookie**

HttpCookie是存储在浏览器端的会话对象,cookie可以存储在浏览器的内存或硬盘中.

cookie只能存储字符串类型的键值对("JSESSIONID","5435dsadfdsfsfdsf")。

cookie常用于记住密码、购物车等功能的实现.

使用步骤：

1. 创建Cookie对象，绑定数据

\* new Cookie(String name, String value)

2. 发送Cookie对象

\* response.addCookie(Cookie cookie)

3. 获取Cookie，拿到数据

\* Cookie[] request.getCookies()

1. **Session**

session对象:主要用于保存用户的各种信息，直到它的生命周期(一般为900s)超时或人为释放掉为止。用户只要一连接到服务器，就立刻分配一个session给用户。

可以通过session对象来判断此用户是否是合法用户

Session的主要方法：

 服务器上通过sessionID来区分不同的用户, 任何连接到服务器上的用户，服务器都会为之分配唯一的一个不会重复的sessionID。session.getId() ，长度为:32

 判断是否是新的session。Session.isNew()

 设置属性:

public void setAttribute(String name,Object value)

 取得属性:

public Object getAttribute(String name)

 删除属性

public void removeAttribute(String name)

让用户的session失效

Session.invalidate()

 取得用户最后一次操作的时间

public long getLastAccessedTime()

获得创建时间

public long getCreationTime();

1. **Session的实现原理？为什么tomcat可以根据不同的用户取到不同的session？**

session是通过cookie来进行对应的存取的,当tomcat创建用户的session的时候,会向浏览器响应一个记录着该用户sessionId的cookie,("JSESSIONID","5435dsadfdsfsfdsf")

浏览器接收到cookie之后会将cookie保存在浏览器内存中,以后该客户端每次访问tomcat的时候,都会携带着该cookie。

1. **Cookie的作用是什么?和Session有什么区别？**

Cookie 和 Session都是用来跟踪浏览器用户身份的会话方式，但是两者的应用场景不太一样。

Cookie 一般用来保存用户信息 比如①我们在 Cookie 中保存已经登录过得用户信息，下次访问网站的时候页面可以自动帮你登录的一些基本信息给填了；②一般的网站都会有保持登录也就是说下次你再访问网站的时候就不需要重新登录了，这是因为用户登录的时候我们可以存放了一个 Token 在 Cookie 中，下次登录的时候只需要根据 Token 值来查找用户即可(为了安全考虑，重新登录一般要将 Token 重写)；③登录一次网站后访问网站其他页面不需要重新登录。

Session 的主要作用就是通过服务端记录用户的状态。 典型的场景是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。服务端给特定的用户创建特定的 Session 之后就可以标识这个用户并且跟踪这个用户了。

Cookie 数据保存在客户端(浏览器端)，Session 数据保存在服务器端。

Cookie 存储在客户端中，而Session存储在服务器上，相对来说 Session 安全性更高。如果要在 Cookie 中存储一些敏感信息，不要直接写入 Cookie 中，最好能将 Cookie 信息加密然后使用到的时候再去服务器端解密。

1. **如何防止表单重复提交?**

1.禁掉提交按钮。表单提交后使用Javascript使提交按钮disable。这种方法防止心急的用户多次点击按钮。但有个问题，如果客户端把Javascript给禁止掉，这种方法就无效了。

2.Post/Redirect/Get模式。在提交后执行页面重定向，这就是所谓的Post-Redirect-Get (PRG)模式。简言之，当用户提交了表单后，你去执行一个客户端的重定向，转到提交成功信息页面。

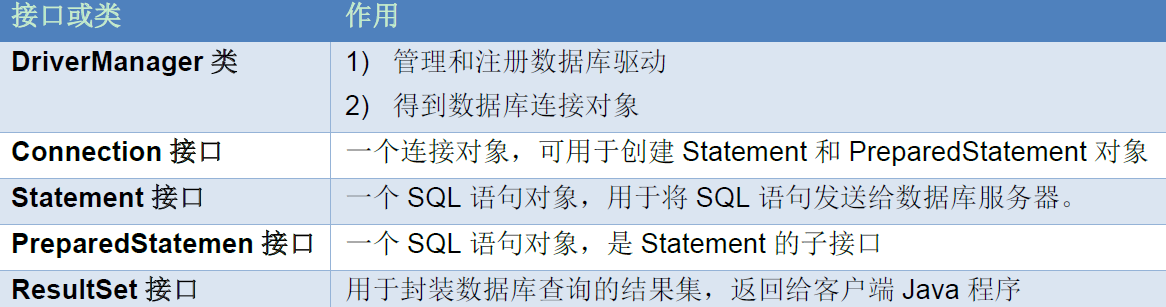
这能避免用户按F5导致的重复提交，而其也不会出现浏览器表单重复提交的警告，也能消除按浏览器前进和后退按导致的同样问题。

3.在session中存放一个特殊标志。当表单页面被请求时，生成一个特殊的字符标志串，存在session中，同时放在表单的隐藏域里。接受处理表单数据时，检查标识字串是否存在，并立即从session中删除它，然后正常处理数据。

如果发现表单提交里没有有效的标志串，这说明表单已经被提交过了，忽略这次提交。

4.在数据库里添加约束。在数据库里添加唯一约束或创建唯一索引，防止出现重复数据。这是最有效的防止重复提交数据的方法。

1. **JDBC**
2. **JDBC的核心API**



1. **Statement接口与PreparedStatement接口区别**

* Statement类的对象将SQL语句发送给数据库，如果SQL语句运行后产后结果集，Statement对象会将结果集返回给一个ResultSet对象。
* PreparedStatement可以接受一个带有占位符?的查询语句，并且PreparedStatement对象会将传入的SQL语句进行编译并暂时保存在内存中。
* 动态SQL语句：就是在SQL语句中可以提供参数。
* SQL 注入是用户利用某些系统没有对输入数据进行充分的检查，从而进行恶意破坏的行为。
* PreparedStatement可防止SQL注入现象发生

1. **为什么要使用 PreparedStatement**

一、代码的可读性和可维护性.Statement 要不断地拼接， PreparedStatement 不会。

二、PreparedStatement 尽最大可能提高性能.DB 有缓存机制，相同的预编译语句再次被调用不会再次需要编译。

三、最重要的一点是极大地提高了安全性.Statement 容易被 SQL 注入，而 PreparedStatementc 传入的内容不会和 sql 语句发生任何匹配关系。

1. **用JDBC来实现访问数据库记录可以采用下面的几个步骤**

第一步：Class.forName()加载数据库连接驱动；

第二步：DriverManager.getConnection()获取数据连接对象;

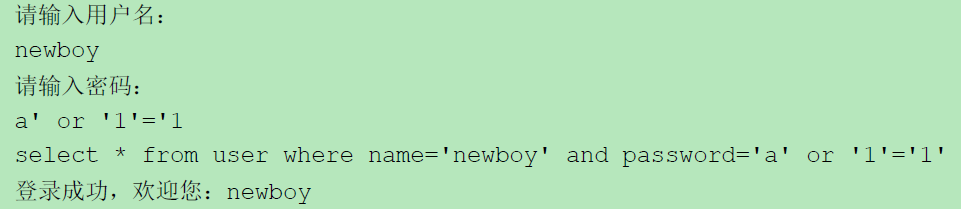
第三步：根据 SQL 获取 sql 会话对象，有 2 种方式 Statement、PreparedStatement ;

第四步：执行 SQL 处理结果集，执行 SQL 前如果有参数值就设置参数值 setXXX();

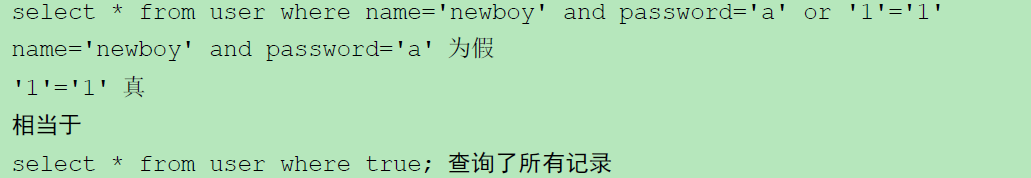
第五步：关闭结果集、关闭会话、关闭连接。

1. **SQL注入攻击**

当我们输入以下密码，我们发现我们账号和密码都不对竟然登录成功了



问题分析



我们让用户输入的密码和SQL语句进行字符串拼接。用户输入的内容作为了SQL语句语法的一部分，改变了原有SQL真正的意义，以上问题称为SQL注入。要解决SQL注入就不能让用户输入的密码和我们的SQL语句进行简单的字符串拼接。

解决：PreparedStatement

prepareStatement()会先将SQL语句发送给数据库预编译。PreparedStatement会引用着预编译后的结果。可以多次传入不同参数给PreparedStatement对象并执行。减少SQL编译次数，提高效率。2.安全性更高，没有SQL注入的隐患。

3.提高了程序的可读性

1. **为什么要使用连接池？**

•传统的数据库连接方式

一个连接对象对应一个物理连接

每次操作都打开一个物理连接，

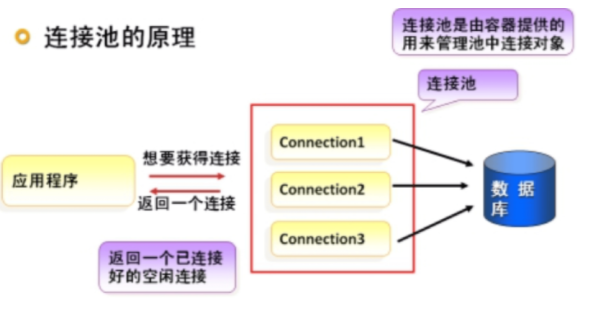
使用完都关闭连接，造成系统性能低下。

•连接池技术

客户程序得到的连接对象是连接池中物理连接的一个句柄，调用连接对象的close()方法,物理连接并没有关闭,数据源的实现只是删除了客户程序中的连接对象和池中的连接对象之间的联系.

•数据库连接的建立及关闭是耗费系统资源的操作，在大型应用中对系统的性能影响尤为明显。为了能重复利用数据库连接对象,缩短请求的响应时间和提高服务器的性能,支持更多的客户，应采用连接池技术.

1. 连接池的原理



传统连接方式:

首先调用Class.forName()方法加载数据库驱动，

然后调用DriverManager.getConnection()方法建立连接.

连接池技术:

连接池解决方案是在应用程序启动时就预先建立多个数据库连接对象,然后将连接对象保存到连接池中。

当客户请求到来时,从池中取出一个连接对象为客户服务。

当请求完成时,客户程序调用close()方法,将连接对象放回池中.

对于多于连接池中连接数的请求，排队等待。

应用程序还可根据连接池中连接的使用率，动态增加或减少池中的连接数。

常见数据库连接池：DBCP、C3P0、Druid(最常用)

1. **servlet**
2. **HttpServlet中的方法**

**doGet()/doPost()方法**

通过覆盖HttpServlet类中的doGet()方法，可以处理浏览器端发送过来的GET请求

post是一种邮寄的方式，在浏览器的地址栏不显示提交的信息，但是这种方式传送的数据是没有限制的；

get是一种手把手的提交到服务器上，最大的信息量是1K，而且提交的信息显示在浏览器上。

**init()**

在第一次调用servlet时，会创建servlet实例，在创建这个实例时，会调用HttpServlet中的init()方法（这个方法只会被调用一次）

**service()**

当客户端有一个对servlet的请求发送过来，则服务器会产生一个新的线程，并让它调用servlet的service()方法，service()会根据收到的客户端请求类型，决定调用doGet()还是doPost()方法

**destroy()**

在删除servlet实例之前，服务器会先调用destroy()方法

1. **Servlet生命周期**

1、加载：在下列时刻加载 Servlet：（1）如果已配置自动加载选项，则在启动服务器时自动

2、加载 (web.xml中设置< load-on-start>)；（2）在服务器启动后，客户机首次向 Servlet 发出请求时;（3）重新加载 Servlet 时（只执行一次）

3、实例化：加载 Servlet 后，服务器创建一个 Servlet 实例。（只执行一次）

4、初始化：调用 Servlet 的 init() 方法。在初始化阶段，Servlet 初始化参数被传递给 Servlet 配置对象ServletConfig。 （只执行一次）

5、请求处理：对于到达服务器的客户机请求，服务器创建针对此次请求的一个“请求”对象和一个“响应”对象。服务器调用 Servlet 的 service() 方法，该方法用于传递“请求”和“响应”对象。service() 方法从“请求”对象获得请求信息、处理该请求并用“响应”对象的方法以将响应传回客户机。service() 方法可以调用其它方法来处理请求，例如 doGet()、doPost() 或其它的方法。（每次请求都执行该步骤）

6、销毁：当服务器不再需要 Servlet, 或重新装入 Servlet 的新实例时，服务器会调用 Servlet 的 destroy() 方法。（只执行一次）

1. **http 中重定向和请求转发的区别？**

1、从数据共享上

Forword是一个请求的延续，可以共享request的数据

Redirect开启一个新的请求，不可以共享request的数据

2、从地址栏

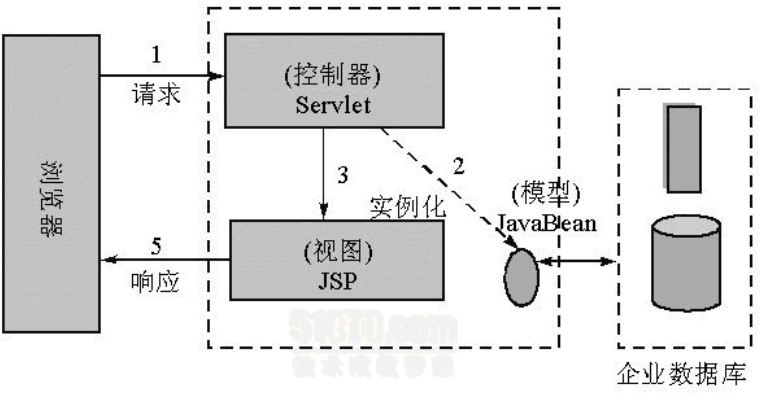
Forword转发地址栏不发生变化

Redirect转发地址栏发生变化

1. **MVC设计模式**

MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

最简单的、最经典就是Jsp(view) +Servlet(controller) + JavaBean(model)



1、当控制器收到来自用户的请求

2、控制器调用JavaBean完成业务

3、完成业务后通过控制器跳转JSP页面的方式给用户反馈信息

1. **Filter与listener**
2. **谈谈过滤器原理及其作用?**

**原理:**

过滤器是运行在服务器端的一个拦截作用的web组件，一个请求来到时，web容器会判断是否有过滤器与该信息资源相关联，如果有则交给过滤器处理，然后再交给目标资源，响应的时候则以相反的顺序交给过滤器处理，最后再返回给用户浏览器

一般用于日志记录、性能、安全、权限管理等公共模块。

**过滤器开发:**

过滤器是一个实现了javax.servlet.Filter接口的java类

主要业务代码放在doFilter方法中

业务代码完成后要将请求向后传递，即调用FilterChain对象的doFilter方法

**配置:**

在web.xml中增加如下代码

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>Filter完整类名</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<url-pattern>/\*(要过虑的url，此处\*表示过虑所有的url)</url-pattern>

</filter-mapping>

1. **Filter怎么执行的**

首先初始化过滤器，然后服务器组织过滤器链，所有的请求都必须需要先通过过滤器链，

过滤器链是一个栈，遵循先进后出的原则 ，所有的请求需要经过一个一个的过滤器，执行顺序要根据web.xml里配置的<filter-mapping>的位置前后执行，每个过滤器之间通过chain.doFilter连接， 最后抵达真正请求的资源，执行完后再从过滤器链退出

1. 谈谈监听器作用及其分类?

监听器也叫Listener，是一个实现特定接口的java类，使用时需要在web.xml中配置，它是web服务器端的一个组件，它们用于监听的事件源分别为SerlvetConext,HttpSession和ServletRequest这三个域对象

主要有以下三种操作:

–监听三个域对象创建和销毁的事件监听器

–监听域对象中属性的增加和删除的事件监听器

–监听绑定到HttpSession域中的某个对象的状态的时间监听器

配置：

< listener>< listener-class>实现以上任意接口的java类全名< /listener-class>

< /listener>

1. **Linux**
2. **常用的 Linux 命令**

列出文件列表：ls 【参数 -a -l】

创建目录和移除目录：mkdir rmdir

用于显示文件后几行内容：tail 打包：tar -xvf

打包并压缩：tar –zcvf

查找字符串：grep

显示当前所在目录：pwd 创建空文件：touch

编辑器：vim vi

列出文件列表：ls 【参数 -a -l】

创建目录和移除目录：mkdir rmdir

用于显示文件后几行内容：tail 打包：tar -xvf

打包并压缩：tar -zcvf

查找字符串：grep

显示当前所在目录：pwd 创建空文件：touch

编辑器：vim vi

1. **Linux 中如何查看日志**

动态打印日志信息：tail –f 日志文件

1. **Linux 怎么关闭进程**

通常用 ps 查看进程 PID ，用 kill 命令终止进程。

ps 命令用于查看当前正在运行的进程。

grep 是搜索

例如： ps -ef | grep java

表示查看所有进程里 CMD 是 java 的进程信息。

ps -aux | grep java

-aux 显示所有状态

kill 命令用于终止进程。例如： kill -9 [PID]

-9 表示强迫进程立即停止。

1. Git
2. **git命令**

新增文件的命令：git add file或者git add .

提交文件的命令：git commit –m或者git commit –a

查看工作区状况：git status –s

拉取合并远程分支的操作：git fetch/git merge或者git pull

查看提交记录命令：git reflog

1. **提交时发生冲突，你能解释冲突是如何产生的吗？你是如何解决的？**

开发过程中，我们都有自己的特性分支，所以冲突发生的并不多，但也碰到过。诸如公共类的公共方法，我和别人同时修改同一个文件，他提交后我再提交就会报冲突的错误。

发生冲突，在IDE里面一般都是对比本地文件和远程分支的文件，然后把远程分支上文件的内容手工修改到本地文件，然后再提交冲突的文件使其保证与远程分支的文件一致，这样才会消除冲突，然后再提交自己修改的部分。特别要注意下，修改本地冲突文件使其与远程仓库的文件保持一致后，需要提交后才能消除冲突，否则无法继续提交。必要时可与同事交流，消除冲突。

发生冲突，也可以使用命令。

通过git stash命令，把工作区的修改提交到栈区，目的是保存工作区的修改；

通过git pull命令，拉取远程分支上的代码并合并到本地分支，目的是消除冲突；

通过git stash pop命令，把保存在栈区的修改部分合并到最新的工作空间中；

1. **Spring（重点）**
2. **什么是 Spring 框架?**

Spring 是一种轻量级开发框架，旨在提高开发人员的开发效率以及系统的可维护性。

我们一般说 Spring 框架指的都是 Spring Framework，它是很多模块的集合，使用这些模块可以很方便地协助我们进行开发。这些模块是：核心容器、数据访问/集成,、Web、AOP（面向切面编程）、工具、消息和测试模块。比如：核心容器中的 Core 组件是Spring 所有组件的核心，Beans 组件和 Context 组件是实现IOC和依赖注入的基础，AOP组件用来实现面向切面编程。

1. **列举一些重要的Spring模块？**

Spring Core： 基础,可以说 Spring 其他所有的功能都需要依赖于该类库。主要提供 IoC 依赖注入功能。

Spring AOP ：提供了面向切面的编程实现。

Spring JDBC : Java数据库连接。

Spring JMS ：Java消息服务。

Spring ORM : 用于支持Hibernate等ORM工具。

Spring Web : 为创建Web应用程序提供支持。

Spring Test : 提供了对 JUnit 和 TestNG 测试的支持。

1. **Spring 中的设计模式**

工厂设计模式 : Spring使用工厂模式通过 BeanFactory、ApplicationContext 创建 bean 对象。

代理设计模式 : Spring AOP 功能的实现。

单例设计模式 : Spring 中的 Bean 默认都是单例的。

模板方法模式 : Spring 中 jdbcTemplate、hibernateTemplate 等以 Template 结尾的对数据库操作的类，它们就使用到了模板模式。

包装器设计模式 : 我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。

适配器模式 :Spring AOP 的增强或通知(Advice)使用到了适配器模式、spring MVC 中也是用到了适配器模式适配Controller。

1. **IOC**
2. **什么是Ioc及其优点**

IoC（Inverse of Control:控制反转）是一种设计思想，就是 将原本在程序中手动创建对象的控制权，交由Spring框架来管理。IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个Map（key，value）,Map 中存放的是各种对象。

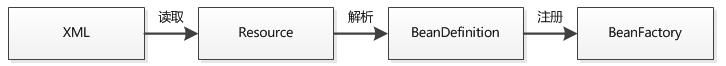
将对象之间的相互依赖关系交给 IoC 容器来管理，并由 IoC 容器完成对象的注入。这样可以很大程度上简化应用的开发，把应用从复杂的依赖关系中解放出来。 IoC 容器就像是一个工厂一样，当我们需要创建一个对象的时候，只需要配置好配置文件/注解即可，完全不用考虑对象是如何被创建出来的。

在实际项目中一个 Service 类可能有几百甚至上千个类作为它的底层，假如我们需要实例化这个 Service，你可能要每次都要搞清这个 Service 所有底层类的构造函数，这可能会把人逼疯。如果利用 IoC 的话，你只需要配置好，然后在需要的地方引用就行了，这大大增加了项目的可维护性且降低了开发难度。

Spring 时代我们一般通过 XML 文件来配置 Bean，后来开发人员觉得 XML 文件来配置不太好，于是 SpringBoot 注解配置就慢慢开始流行起来。

1. **IOC容器的初始化过程**

IOC容器的初始化主要包括Resource定位，载入和注册三个步骤，接下来我们依次介绍。



**Resource资源定位：**

Resouce定位是指BeanDefinition的资源定位，也就是IOC容器找数据的过程。Spring中使用外部资源来描述一个Bean对象，IOC容器第一步就是需要定位Resource外部资源。由ResourceLoader通过统一的Resource接口来完成定位。

**BeanDefinition的载入：**

载入过程就是把定义好的Bean表示成IOC容器内部的数据结构，即BeanDefinition。在配置文件中每一个Bean都对应着一个BeanDefinition对象。

通过BeanDefinitionReader读取，解析Resource定位的资源，将用户定义好的Bean表示成IOC容器的内部数据结构BeanDefinition。

在IOC容器内部维护着一个BeanDefinitionMap的数据结构，通过BeanDefinitionMap，IOC容器可以对Bean进行更好的管理。

**BeanDefinition的注册：**

注册是将前面的BeanDefition保存到Map中的过程，通过BeanDefinitionRegistry接口来实现注册。

1. **Spring 提供了多种依赖注入的方式**

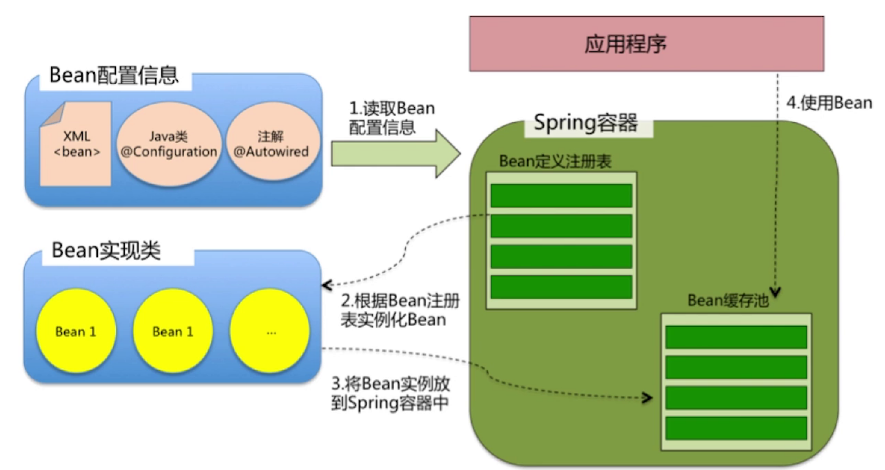
Set 注入

构造器注入

接口注入

注解注入

Spring启动时读取应用程序提供的Bean 配置信息，并在Spring 容器中生成一份相应的Bean 配置注册表，然后根据这张注册表实例化Bean，装配好Bean 之间的依赖关系，为上层应用提供准备就绪的运行环境。其中Bean 缓存池为HashMap 实现



1. **Spring中，如何给对象的属性赋值?**

1. 通过构造函数

2. 通过set方法给属性注入值

3. 自动装配

**byName** 通过参数名自动装配，如果一个bean的name 和另外一个bean的 property 相同就自动装配。

**byType** 通过参数的数据类型自动自动装配，如果一个bean的数据类型和另 外一个bean的property属性的数据类型兼容，就自动装配必须确保 该类型在IOC容器中只有一个对象；否则报错。

4.注解

使用注解步骤：

1）先引入context名称空间

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

2）开启注解扫描

<context:component-scan base-package="cn.itcast.e\_anno2"></context:component-scan>

3）使用注解

通过注解的方式，把对象加入ioc容器。

创建对象以及处理对象依赖关系，相关的注解：

@Component 指定把一个对象加入IOC容器

@Repository 作用同@Component； 在持久层使用

@Service 作用同@Component； 在业务逻辑层使用

@Controller 作用同@Component； 在控制层使用

@Resource或@Autowired 属性注入

@Qualifier指定注入bean的名称，防止在容器中有重名的bean

总结：

1） 使用注解，可以简化配置，且可以把对象加入IOC容器,及处理依赖关系(DI)  
 2） 注解可以和XML配置一起使用。

3） @Resource的作用相当于@Autowired，只不过@Autowired按byType自动 注入，而 @Resource默认按 byName自动注入

1. **bean对象创建的细节**

1) 对象创建： 单例/多例

scope="singleton", 默认值， 即 默认是单例 【service/dao/工具类】

scope="prototype", 多例； 【Action对象】

Spring的单例bean不是线程安全的（可以用局部变量使线程安全）

2) 什么时候创建?

scope="prototype" 在用到对象的时候，才创建对象。

scope="singleton" 在启动(容器初始化之前)， 就已经创建了bean，且整个应用只有一个。

3)是否延迟创建

lazy-init="false" 默认为false, 不延迟创建，即在启动时候就创建对象

lazy-init="true" 延迟初始化， 在用到对象的时候才创建对象（只对单例有效）

4) 创建对象之后，初始化/销毁

init-method="init\_user" 【对应对象的init\_user方法，对象创建之后执行】 destroy-method="destroy\_user" 【在调用容器对象的destriy方法时候执行，(容器用实现类)】

1. **Bean的作用域？**

singleton : 唯一 bean 实例，Spring 中的 bean 默认都是单例的。

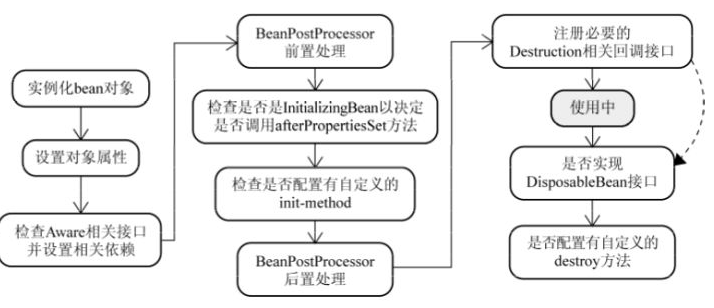
prototype : 每次请求都会创建一个新的 bean 实例。

request : 每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有效。

session : 每一次HTTP请求都会产生一个新的 bean，该bean仅在当前 HTTP session 内有效。

global-session： 全局session作用域，仅仅在基于portlet的web应用中才有意义，Spring5 已经没有了。Portlet是能够生成语义代码(例如：HTML)片段的小型Java Web插件。它们基于 portlet容器，可以像servlet一样处理HTTP请求。但是，与 servlet 不同，每个 portlet 都 有不同的会话

1. **Bean的生命周期**



Bean 容器找到配置文件中 Spring Bean 的定义。

Bean 容器利用 Java Reflection API 创建一个Bean的实例。

如果涉及到一些属性值利用 set()方法设置一些属性值。

如果 Bean 实现了 BeanNameAware 接口，调用 setBeanName()方法，传入Bean的名字。

如果 Bean 实现了 BeanClassLoaderAware 接口，调用 setBeanClassLoader()方法，传入 ClassLoader对象的实例。

与上面的类似，如果实现了其他 \*.Aware接口，就调用相应的方法。

如果有和加载这个 Bean 的 Spring 容器相关的 BeanPostProcessor 对象，执行postProcessBeforeInitialization() 方法

如果Bean实现了InitializingBean接口，执行afterPropertiesSet()方法。

如果 Bean 在配置文件中的定义包含 init-method 属性，执行指定的方法。

如果有和加载这个 Bean的 Spring 容器相关的 BeanPostProcessor 对象，执行postProcessAfterInitialization() 方法

当要销毁 Bean 的时候，如果 Bean 实现了 DisposableBean 接口，执行 destroy() 方法。

当要销毁 Bean 的时候，如果 Bean 在配置文件中的定义包含 destroy-method 属性，执行指定的方法。

1. **BeanFactory和ApplicationContext的区别？**

BeanFactory是Spring里面最顶层的接口，包含了各种Bean的定义，读取Bean配置文档，管理Bean的加载、实例化，控制Bean的生命周期，维护Bean之间的依赖关系。

ApplicationContext接口是BeanFactory的派生，除了提供BeanFactory所具有的功能外，还提供了更完整的框架功能：

1. **AOP**

答： AOP，面向切面编程是指当需要在某一个方法之前或者之后做一些额外的操作，比如说日志记录，权限判断，异常统计等，可以利用AOP将功能代码从业务逻辑代码中分离出来。

**Spring中的AOP主要有两种实现方式：**

使用JDK动态代理实现，使用java.lang.reflection.Proxy类来处理

使用cglib来实现

两种实现方式的不同之处：

JDK动态代理，只能对实现了接口的类生成代理，而不是针对类，该目标类型实现的接口都将被代理。原理是通过在运行期间创建一个接口的实现类来完成对目标对象的代理。

cglib主要针对类实现代理，对是否实现接口无要求。原理是对指定的类生成一个子类，覆盖其中的方法，因为是继承，所以被代理的类或方法不可以声明为final类型。

**Spring AOP对这两种代理方式的选择：**

如果目标对象实现了接口，默认情况下会采用JDK的动态代理实现AOP，也可以强制使用cglib实现AOP；

如果目标对象没有实现接口，必须采用cglib库，Spring会自动在JDK动态代理和cglib之间转换。

1. **事务**

Spring支持编程式事务(不用)和声明式事务两种方式：

编程式事务管理：使用TransactionTemplate实现。

声明式事务管理：建立在AOP之上的。其本质是通过AOP功能，对方法前后进行拦截，将事务处理的功能编织到拦截的方法中，也就是在目标方法开始之前加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。

声明式事务的优点：

就是不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明或通过@Transactional注解的方式，便可以将事务规则应用到业务逻辑中。

**事务选择：**

声明式事务管理要优于编程式事务管理，这正是Spring倡导的非侵入式的开发方式，使业务代码不受污染，只要加上注解就可以获得完全的事务支持。唯一不足之处是声明式事务的最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。

当多个Spring事务存在的时候，Spring定义了下边的7个传播行为来处理这些事务行为：

PROPAGATION\_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

PROPAGATION\_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。

spring事务对于查询的find()是只读，对保存save()是读写？

1. **Mybatis（重点）**
2. MyBatis

MyBatis是一个半ORM（对象关系映射）框架，内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。通过直接编写原生态SQL，可以严格控制SQL语句的执行性能，灵活度高(支持动态SQL语句)。

MyBatis 使用XML或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了JDBC代码手动设置参数以及获取结果集的繁琐步骤。

MyBatis通过xml文件或注解的方式将要执行的各种 statement 配置起来，并通过Java对象和 statement中SQL的动态参数进行映射生成最终执行的SQL语句，最后由MyBatis框架执行SQL语句，并将结果映射为Java对象并返回。

1. **Mybatis架构或工作流程**

每一个Mybatis的应用程序都以一个SqlSessionFactory(法克特润)对象的实例为核心。首先用字节流通过Resource将配置文件读入，然后通过SqlSessionFactoryBuilder().build方法创建SqlSessionFactory，然后再通过SqlSessionFactory.openSession()方法创建一个SqlSession为每一个数据库事务服务。

经历了Mybatis初始化 –>创建SqlSession –>运行SQL语句，返回结果三个过程

1. **Mapper.xml中statement中属性含义**
2. id:sql语句唯一标识
3. parameterType：指定输入参数类型，mybatis通过ognl从输入对象中获取参数值拼 接在sql中。
4. resultType：指定输出结果类型，mybatis将sql查询结果的一行记录数据映射为resultType指定类型的对象。
5. resultMap：resultType可以指定pojo将查询结果映射为pojo，但需要pojo的属性名和sql查询的列名一致方可映射成功。如果sql查询字段名和pojo的属性名不一致，可以通过resultMap将字段名和属性名作一个对应关系 resultMap实质上还需要将查询结果映射到pojo对象中。

resultMap可以实现将查询结果映射为复杂类型的pojo，比如在查询结果映射对象中包括pojo 和list实现一对一查询和一对多查询。

1. #{}表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换，#{}可以有效防止sql注入。 #{}可以接收简单类型值或pojo属性值。 如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。一般能用#的就别用$.
2. ${}表示拼接sql串，通过${}可以将parameterType 传入的内容拼接在sql中且不进 行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中只能是value。
3. **简述MyBatis的单个参数、多个参数如何传递及如何取值。**

MyBatis传递单个参数，如果是普通类型(String+8个基本)的，取值时在#{}中可以任意指定，如果是对象类型的，则在#{}中使用对象的属性名来取值

MyBatis传递多个参数，默认情况下，MyBatis会对多个参数进行封装Map. 取值时在#{}可以使用0 1 2 .. 或者是param1 param2..

1. **Mapper接口开发方法**

通常Mybatis开发Dao方法有两种。即原始Dao开发方法和Mapper接口开发方法。

Mapper接口开发方法只需要程序员编写Mapper接口（相当于Dao接口），由Mybatis框架根据接口定义创建接口的动态代理对象，代理对象的方法体同上边Dao接口实现类方法。

Mapper接口开发需要遵循以下规范：

1. Mapper.xml文件中的namespace与mapper接口的类路径相同。
2. Mapper接口方法名和Mapper.xml中定义的每个statement的id相同
3. Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql的parameterType类型相同
4. Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同

接口绑定有两种实现方式,一种是通过注解绑定,就是在接口的方法上面加上@Select@Update等注解里面包含Sql语句来绑定,另外一种就是通过xml里面写SQL来绑定,在这种情况下,要指定xml映射文件里面的namespace必须为接口的全路径名.

Dao 接口的工作原理是 JDK 动态代理，MyBatis 运行时会使用 JDK 动态代理为 Dao 接口生成代理 proxy 对象，代理对象 proxy 会拦截接口方法，转而执行MappedStatement所代表的 sql，然后将 sql 执行结果返回。

1. **动态sql**

动态 SQL 是指可以根据不同的参数信息来动态拼接的不确定的SQL叫做动态 SQL。

MyBatis 动态SQL 的主要元素有：if、choose/when/otherwise、trim、where、set、foreach 等。

通过mybatis提供的各种标签方法实现动态拼接sql

foreach标签:循环传入的集合参数

collection:传入的集合的变量名称

item:每次循环将循环出的数据放入这个变量中

open:循环开始拼接的字符串

close:循环结束拼接的字符串

separator:循环中拼接的分隔符

where标签作用:会自动向sql语句中添加where关键字,会去掉第一个条件的and关键字。

include标签：调用sql条件

1. **Mybatis 如何完成MySQL的批量操作,举例说明**

MyBatis完成MySQL的批量操作主要是通过<foreache>标签来拼装相应的SQL语句.

例如:

<insert id="insertBatch" >

insert into tbl\_employee(last\_name,email,gender,d\_id) values

<foreach collection="emps" item="curr\_emp" separator=",">

(#{curr\_emp.lastName},#{curr\_emp.email},#{curr\_emp.gender},#{curr\_emp.dept.id})

</foreach>

</insert>

1. **Mybatis缓存机制**

mybatis是自动开启一级缓存的,sqlsession级别

mybatis的二级缓存需要手动开启的,mapper级别(application应用),

通常项目中,对于字典表(大量查询,很少修改)的数据,可以使用mybatis的二级缓存机制,

提高查询效率。比如省市县、汽车品牌、配件类别、企业信息等

1. **如何开启 MyBatis 的延迟加载？**

只需要在 mybatis-config.xml 设置即可打开延迟缓存功能，完整配置文件如下

1. **Mybatis是如何进行分页的？分页插件的原理是什么？**

逻辑分页，使用MyBatis自带的RowBounds进行分页，它是一次性查询很多数据，然后在数据中再进行检索；

物理分页，手写 SQL 分页或使用分页插件 PageHelper，去数据库查询指定条数的分页数据形式。

1）Mybatis使用RowBounds对象进行分页，可以直接编写sql实现分页，也可以使用Mybatis的分页插件PageHelper。

2）分页插件的原理：实现Mybatis提供的接口Interceptor，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql。

举例：

select \* from student，拦截sql后重写为：select t.\* from （select \* from student）t limit 0，10

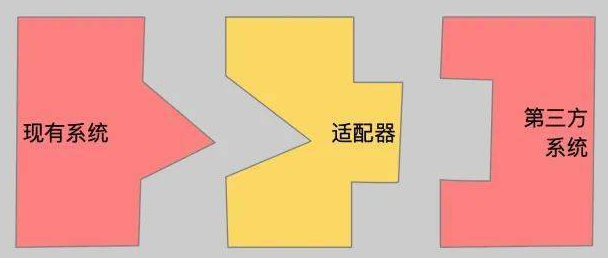
1. **SpringMVC（重点）**
2. **适配器模式**

假设我们在做一套股票看盘系统，数据提供方给我们提供 XML 格式数据，我们获取数据用来显示，随着系统的迭代，我们要整合一些第三方系统的对外数据，但是他们只提供获取 JSON 格式的数据接口。

在不想改变原有代码逻辑的情况下，如何解决呢？

这时候我们就可以创建一个「适配器」。这是一个特殊的对象， 能够转换对象接口， 使其能与其他对象进行交互。

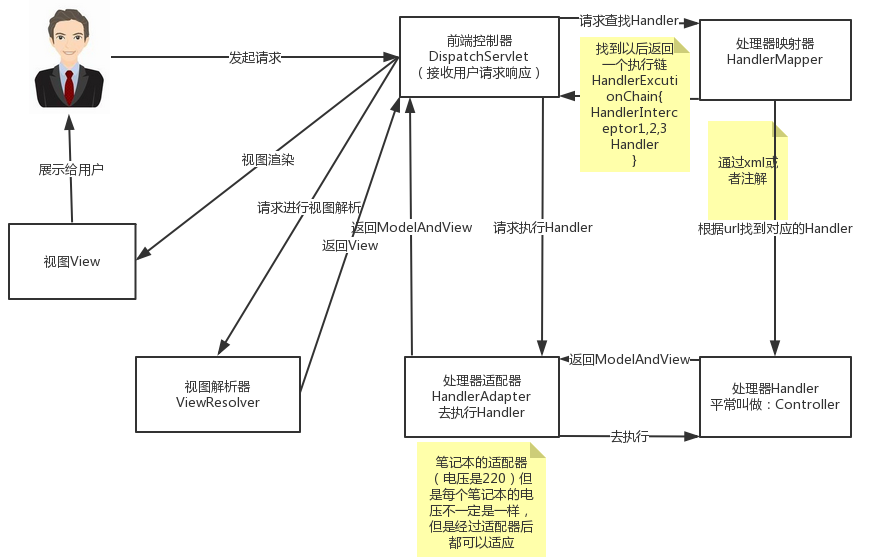
适配器模式通过封装对象将复杂的转换过程隐藏于幕后。被封装的对象甚至察觉不到适配器的存在。



1. **SpringMVC运行原理**

SpringMVC是一种轻量级的Web层框架，是一个基于请求驱动的Web框架，使用了“前端控制器（DispatcherServlet）”模型来进行设计，再根据请求映射规则分发给相应的页面控制器进行处理。消息处理依次经过的组件如下：

DispatcherServlet、HandlerMapping、HandlerAdapter，返回一个ModelAndView逻辑视图名、ViewResolver、View



客户端（浏览器）发送请求，直接请求到 DispatcherServlet。

DispatcherServlet 根据请求信息调用 HandlerMapping，解析请求对应的 Handler。

解析到对应的 Handler（也就是我们平常说的 Controller 控制器）后，开始由 HandlerAdapter 适配器处理。

HandlerAdapter 会根据 Handler 来调用真正的处理器来处理请求，并处理相应的业务逻辑。

处理器处理完业务后，会返回一个 ModelAndView 对象，Model 是返回的数据对象，View 是个逻辑上的 View。

ViewResolver 会根据逻辑 View 查找实际的 View。

DispaterServlet 把返回的 Model 传给 View（视图渲染）。

把 View 返回给请求者（浏览器）

1. **Spring MVC 常用的注解**

springMVC中用到过的注解有

@RequestParam它的作用是接受前台传递的参数并且可以通过defaultValue属性对其设置默认值；在SpringMVC进行文件上传的时候也会通过@RequestParam和MultipartFile结合使用。

@Autowired注解和@Resource注解的作用都是为了进行属性注入，但@Autowired默认是按照类型进行匹配，它是Spring提供的注解，@Resource默认按照名字进行匹配，它是java提供的注解。

在进行restful接口编程的时候我们还会用到@pathvariable注解从路径中获取参数信息以及用到

@ResponseBody注解将实体类自动转换为指定的json格式,

@RequestBody将前台传递过来的json格式的数据转换为对应的javabean。

除此之外还有@Controller，@Service,@Repository分别在ss层，业务逻辑层和持久层的实现类型添加。

@RequestMapping注解在控制层的方法上添加从而将指定url和方法对应起来。

@RestController：相当于 @Controller 加 @ResponseBody 的组合效果

@Configuration：用于定义配置类

@PathVariable用于获取路径参数

@RequestParam用于获取查询参数。

@Value读取比较简单的配置信息：

1. **如何使用SpringMVC 完成 JSON 操作**

配置 MappingJacksonHttpMessageConverter

如果使用mvc:annotation-driven就不需要配置。

使用 @RequestBody 注解作为请求参数或 ResponseBody作为返回值

还有常用的fastjson,jackson等第三方组件。

1. **SpringMVC怎么样设定重定向和转发的？**

（1）转发：在返回值前面加"forward:"，譬如"forward:user.do?name=method4"

（2）重定向：在返回值前面加"redirect:"，譬如"redirect:http://www.baidu.com"

1. **restful**

一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。（强调以资源为导向）

Url格式：http://blog.csdn.net/beat\_the\_world/article/details/45621673

（使用restful后，utl中不能用？传参或.action）

资源操作：使用put、delete、post、get，使用不同方法对资源进行操作。分别对应添加、删除、修改、查询。一般使用时还是post和get。Put和Delete几乎不使用。

客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。

它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

1. **SpringMvc里面拦截器是怎么实现的？**

定义一个类,实现了handlerInterceptor接口,重写 preHandle postHandle afterCompletion 三个方法,之后在springmvc的配置文件当中使用<mvc:interceptors>里面可以配置多个interceptor

**preHandle**

此方法的执行时机是在控制器方法执行之前，所以我们通常是使用此方法对请求部分进行 增强。同时由于结果视图还没有创建生成，所以此时我们可以指定响应的视图。

**postHandle**

此方法的执行时机是在控制器方法执行之后，结果视图创建生成之前。所以通常是使用此 方法对响应部分进行增强。因为结果视图没有生成，所以我们此时仍然可以控制响应结果。

**afterCompletion**

此方法的执行时机是在结果视图创建生成之后，展示到浏览器之前。所以此方法执行时， 本次请求要准备的数据具已生成完毕，且结果视图也已创建完成，所以我们可以利用此方 法进行清理操作。同时，我们也无法控制响应结果集内容。

**使用场景：**

用户登录判断，在执行Action的前面判断是否已经登录，如果没有登录的跳转到登录页面。

用户权限判断，在执行Action的前面判断是否具有，如果没有权限就给出提示信息。