JavaEE

JSP&Servlet部分

1. jsp 和 servlet 有什么区别?

- JSP是Java语言嵌入到Html代码中而来的，是Java和HTML组合成一个扩展名为.jsp的文件，其本质就是Servlet。但JVM不能识别JSP的代码，Web容器将JSP的代码编译成JVM能够识别的Java类。
- Jsp更擅长页面显示,Servlet更擅长逻辑控制。Servlet则是个完整的Java类，这个类的Service方法用于生成对客户端的响应。Servlet的应用逻辑是在Java文件中，并且完全从表示层中的HTML里分离开来。
- Servlet中没有内置对象，Jsp中的内置对象都是必须通过HttpServletRequest对象，HttpServletResponse对象以及HttpServlet对象得到.

2. jsp 有哪些内置对象？作用分别是什么?

JSP 共有以下9个内置的对象：

- request 用户端请求，此请求会包含来自GET/POST请求的参数，并且提供了几个用于获取cookie, header, 和session数据的有用的方法。
- response 网页传回用户端的回应
- pageContext 管理网页的属性，提供了对JSP页面内所有的对象及名字空间（就是四大作用域空间，如page空间、request空间、session空间、application空间）的访问。
- session 与请求有关的会话期，Session可以存贮用户的状态信息。
- application servlet 类似于系统的全局变量，用于实现Web应用中的资源共享。
- out 用来传送回应的输出
- config servlet 存放JSP编译后的初始数据。
- page JSP网页本身
- exception 针对错误网页，未捕捉的例外。表示JSP页面运行时产生的异常和错误信息，该对象只有在错误页面（page指令中设定isErrorPage为true的页面）中才能够使用。

3. 说一下 jsp 的 4 种作用域?

- JSP的四大作用域：page、request、session、application。
- 这四大作用域，其实就是其九大内置对象中的四个，为什么说他们也是JSP的四大作用域呢？因为这四个对象都能存储数据，比如request.setAttribute()注意和request.setParameter()区分开来，一个是存储在域中的、一个是请

求参数，session.setAttribute()、application其实就是SerlvetContext，自然也有setAttribute()方法。而page作用域的操作就需要依靠pageContext对象来进行了。在上面我们也有提到JSP的四大作用域，

- page作用域：代表变量只能在当前页面上生效
- request：代表变量能在一次请求中生效，一次请求可能包含一个页面，也可能包含多个页面，比如页面A请求转发到页面B
- session：代表变量能在一次会话中生效，基本上就是能在web项目下都有效，session的使用也跟cookie有很大的关系。一般来说，只要浏览器不关闭，cookie就会一直生效，cookie生效，session的使用就不会受到影响。
- application：代表变量能一个应用下(多个会话)，在服务器下的多个项目之间都能够使用。比如baidu、wenku等共享帐号。

4. session 和 cookie 有什么区别?

1）cookie以文本格式存储在浏览器上，存储量有限，只允许4KB；而会话存储在服务端，可以无限量存储多个变量并且比cookie更安全。
2）设置cookie时间可以使cookie过期。使用session-destory（），我们将会销毁会话。
3）cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗，考虑到安全应当使用session。
4）session会在一定时间内保存在服务器上。当访问增多，会比较占用服务器的性能，考虑到减轻服务器性能方面，应当使用COOKIE。

5. 说一下 session 的工作原理?

- session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。
- 当程序需要为某个客户端的请求创建一个session的时候，服务器首先检查这个客户端的请求里是否已包含了一个session标识 - 称为session id，如果已包含一个session id则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（如果检索不到，可能会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。 保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器。
- 由于cookie可以被人为的禁止，必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面，附加方式也有两种，一种是作为URL路径的附加信息，表现形式为http://...../xxx;jsessionid=ByOK ... 99zWpBng!-145788764 另一种是作为查询字符串附加在URL后面，表现形式为http://...../xxx?jsessionid=ByOK ... 99zWpBng!-145788764 这两种方式对于用户来说是没有区别的，只是服务器在解析的时候处理的方式不同，采用第一种方式也有利于把session id的信息和正常程序参数区分开来。 为了在整个交互过程中始终保持状态，就必须在每个客户端可能请求的路径后面都包含这个session id。
- 另一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。
- 在谈论session机制的时候，常常听到这样一种误解“只要关闭浏览器，session就消失了”。其实可以想象一下会员卡的例子，除非顾客主动 对店家提出销卡，否则店家绝对不会轻易删除顾客的资料。对session来说也是一样的，除非程序通知服务器删除一个session，否则服务器会一直保 留，程序一般都是在用户做log off的时候发个指令去删除session。然而浏览器从来不会主动在关闭之前通知服务器它将要关闭，因此服务器根本不会有机会知道浏览器已经关闭，之所以会有这种错觉，是大部分session机制都使用会话cookie来保存session id，而关闭浏览器后这个session id就消失了，再次连接服务器时也就无法找到原来的session。如果服务器设置的cookie被保存到硬盘上，或者使用某种手段改写浏览器发出的 HTTP请求头，把原来的session id发送给服务器，则再次打开浏览器仍然能够找到原来的session。
- 恰恰是由于关闭浏览器不会导致session被删除，迫使服务器为seesion设置了一个失效时间，当距离客户端上一次使用session的时间超过这个失效时间时，服务器就可以认为客户端已经停止了活动，才会把session删除以节

省存储空间。

6. 如果客户端禁止 cookie 能实现 session 还能用吗?

- 如果用户禁止cookie，服务器仍会将sessionId以cookie的方式发送给浏览器，但是，浏览器不再保存这个cookie(即sessionId)了。如果想继续使用session，需要采取其他方式来实现sessionId的跟踪。可以使用url重写来实现sessionId的跟踪。即：把会话ID附加在HTML页面中所有的URL上，这些页面作为响应发送给客户。这样，当用户单击URL时，会话ID被自动作为请求行的一部分发送回服务器。这种方法称为URL重写(URL rewriting)。
  使用URL重写应该注意下面几点：

1. 如果使用URL重写，应该在应用程序的所有页面中，对所有的URL编码，包括所有的超链接和表单的action属性值。
2. 应用程序的所有的页面都应该是动态的。因为不同的用户具有不同的会话ID，因此在静态HTML页面中无法在URL上附加会话ID。
3. 所有静态的HTML页面必须通过Servlet运行，在它将页面发送给客户时会重写URL。

7. 解释一下什么是 servlet?

- Servlet是一种服务端的Java技术，可以生成动态WEB页面。Servlet具有更好的可移植性，更强大的功能，更高的效率，更好的安全性。
- Servlet 主要用于处理客户端传来的Http 请求，并返回一个响应，通常来说Servlet就是指HttpServlet，用于处理Http请求，其能够处理的请求有doGet()，doPost()，service()等方法，开发servlet时直接继承javax.servlet,http.HttpServlet.
- Servlet需要在web.xml中进行描述，例如。映射执行servlet的名字，配置servlet类，初始化参数，进行安全配置，URL映射和设置启动优先权。Servlet不仅可以生成HTML脚本输出，也可以生成二进制表单输出。
- 现在许多流行框架都离不开Servlet的支持，比如Spring 容器启动的时候，要在web.xml中装载Spring容器和Actioncontext来初始化Spring的一些参数。如依赖注入，数据库表的映射，初始化系统的安全配置，设置read等属性进行一些相关的操作。

8. 说一说 Servlet 的生命周期?

- servlet 有良好的生存期的定义， 包括加载和实例化、 初始化、 处理请求以及服务结束。
- 这个生存期由 javax.servlet.Servlet 接口的 init,service 和 destroy 方法表达。
- web 容器加载 servlet， 生命周期开始。 通过调用 servlet 的 init()方法进行 servlet 的初始化。
- 通过调用 service()方法实现， 根据请求的不同调用不同的 do***()方法。 结束服务， web 容
- 器调用 servlet 的 destroy()方法。

9. servlrt API 中 forward()与 redirect()的区别?

- forward是服务器端的转向也就是请求转发，而redirect是客户端的跳转也就是重定向
- 使用forward浏览器的地址不会发生改变，而redirect会发生改变。
- forward是一次请求中完成，而redirect是重新发起请求。
- forward是在服务器端完成，而不用客户端重新发起请求，效率较高。

1. 请求转发的特点： 只请求一次，而且属于内部跳转 。地址栏不会发生变化 。不允许访问外部资源 。绝对路径的/代表的是根目录之后 。效率偏高 。请求转发的语法：request.getRequestDispacher(地址).forward(请求对象,响应对象)

2. 重定向的特点： 整个过程发出两次请求 。地址栏会发生变化，并跳转到最新的页面，地址栏也是最新页面的地址 。允许访问外部资源，因为服务器已经响应回了浏览器，而且浏览器也发出了新的请求，由于HTTP是无状态的所以两次请求没有联系，第二次请求可以随意去任何网页 绝对路径的/代表的是端口号之后 。效率偏低，因为有两次请求，相对来说效率低 。重定向语法: response.sendRedirect(地址)

10.什么情况下调用 doGet()和 doPost()?

- 前端页面中的form是get就用doGet()，是post就是doPost() 。
- get和post是请求方式，一般get携带的信息量有限制，而且他的内容会在显示栏里面出现，不安全。post可携带大量数据，并且信息不回出现在显示栏里比较安全。当实现查询功能时适合用get，get效率比post高些。

11.Request 对象的主要方法

- setAttribute(String name,Object)：设置名字为name的request 的参数值
- getAttribute(String name)：返回由name指定的属性值
- getAttributeNames()：返回request 对象所有属性的名字集合，结果是一个枚举的实例
- getCookies()：返回客户端的所有 Cookie 对象，结果是一个Cookie 数组
- getCharacterEncoding()： 返回请求中的字符编码方式
- getContentLength()： 返回请求的 Body的长度
- getHeader(String name)： 获得HTTP协议定义的文件头信息
- getHeaders(String name)： 返回指定名字的request Header 的所有值，结果是一个枚举的实例
- getHeaderNames()： 返回所以request Header 的名字，结果是一个枚举的实例
- getInputStream()： 返回请求的输入流，用于获得请求中的数据
- getMethod()： 获得客户端向服务器端传送数据的方法
- getParameter(String name)： 获得客户端传送给服务器端的有 name指定的参数值
- getParameterNames()： 获得客户端传送给服务器端的所有参数的名字，结果是一个枚举的实例
- getParameterValues(String name)：获得有name指定的参数的所有值
- getProtocol()：获取客户端向服务器端传送数据所依据的协议名称
- getQueryString()： 获得查询字符串
- getRequestURI()： 获取发出请求字符串的客户端地址
- getRemoteAddr()：获取客户端的 IP 地址
- getRemoteHost()： 获取客户端的名字
- getSession([Boolean create])： 返回和请求相关 Session
- getServerName()： 获取服务器的名字
- getServletPath()：获取客户端所请求的脚本文件的路径
- getServerPort()：获取服务器的端口号
- removeAttribute(String name)：删除请求中的一个属性

12.request.getAttribute()和 request.getParameter()有何区别?

- getParameter 得到的都是 String 类型的，获取 POST/GET 传递的参数值，用于客户端重定向redirect时，即点击了链接或提交按扭时传值用。
- getAttribute 可获取到对象容器中的数据值，返回的是 Object，需进行转换,可用 setAttribute 设置成任意对象，使用很灵活，用于服务器端重定向forward。，只能收到setAttribute 传过来的值，获取的是session。

13.MVC 的各个部分都有那些技术来实现?如何实现?

- model：应用的业务逻辑（如：数据库的操作），通过JavaBean实现（hibernate、 mybatis、 ibatis）
- view：视图层，用于与用户的交互，主要由jsp页面产生。（jsp、 FreeMarker、 前端三件套、 Velocity ）
- controller：处理过程控制，一般是一个servlet。它可以分派用户的请求并选择恰当的视图以用于显示，，也可以解释用户的输入并将它们映射为模型层可执行的操作。（severlet、 struts、 spring）

14.说出数据连接池的工作机制是什么?

- 数据连接池是把数据库连接放到服务器上,比如tomcat上,那么每次操作数据库的时候就不需要再"连接"到数据库再进行相关操作,而是直接操作服务器上的"连接池"。
- 把数据库当做一条河,那么"连接池"就是一个"水池",这个水池里面的水是由事先架好的通向"小溪"的水管引进来的,所以,你想喝水的时候不必大老远地跑到小溪边上,而只要到这个水池。这样的话就可以提高"效率"，但是数据池一般是用在数据量比较大的项目，这样可以提高程序的效率，但就把相关的负荷加在了服务器上,因为这个"池"是在服务器上的，过于频繁的请求会使得服务器负载加重。

数据连接池怎么设置?

在数据库连接字段的时候可以设置
比如：“DataSource=Server;Initial;Catalog=db;UserID=test;Password=test;MaxPoolSize = 100;MinPoolSize= 15”这样连接池 最大为100，最小为15

- 一个项目对数据库连接的管理能显著影响到整个应用程序的伸缩性和健壮性，影响到程序的性能指标。数据库连接池正是针对这个问题提出来的。数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而再不是重新建立一个；释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。
- 数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由最小数据库连接数来设定的。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。连接池的最大数据库连接数量限定了这个连接池能占有的最大连接数，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。

数据库连接池的最小连接数和最大连接数的设置要考虑到下列几个因素:

1. 最小连接数是连接池一直保持的数据库连接，所以如果应用程序对数据库连接的使用量不大，将会有大量的数据库连接资源被浪费；
2. 最大连接数是连接池能申请的最大连接数，如果数据库连接请求超过此数，后面的数据库连接请求将被加入到等待队列中，这会影响之后的数据库操作。
3. 如果最小连接数与最大连接数相差太大，那么最先的连接请求将会获利，之后超过最小连接数量的连接请求等价于建立一个新的数据库连接。不过，这些大于最小连接数的数据库连接在使用完不会马上被释放，它将被放到连接池中等待重复使用或是空闲超时后被释放。

15.为什么要用 ORM? 和 JDBC 有何不一样?

ORM的作用是在关系型数据库和对象之间作一个映射，这样，我们在具体的操作数据库的时候，就不需要再去和复杂的SQL语句打交道，只要像平时操作对象一样操作它就可以了 。

1. 代码复杂性：

- 用JDBC的API访问数据库，代码量较大，特别是访问字段较多的表的时候，代码显得繁琐，容易出错，程序员需要耗费大量的时间、精力去编写具体的数据库访问的SQL语句，还要十分小心其中大量重复的源代码是否有疏漏，并不能集中精力于业务逻辑开发上面。

- ORM则建立了Java对象与数据库对象之间的影射关系，程序员不需要编写复杂的SQL语句，直接操作Java对象即可，从而大大降低了代码量，也使程序员更加专注于业务逻辑的实现。

2. 数据库对象连接问题

- 采用JDBC编程，必须保证各种关系数据之间不能出错，极其痛苦； ORM建立Java对象与数据库对象关系影射的同时，也自动根据数据库对象之间的关系创建Java对象的关系，并且提供了维持这些关系完整、有效的机制。

3. 系统架构问题

- 使用JDBC编程时，程序员必须知道用什么数据库、有哪些表、各个表字段、各个字段的类型是什么、表与表之间什么关系、创建了什么索引等等。
- 使用ORM技术，可以将数据库层完全隐蔽，程序员只需要根据业务逻辑的需要调用Java对象的Getter和 Setter方法，即可实现对后台数据库的操作。把ORM搭建好后，把Java对象交给程序员去实现业务逻辑，使数据访问层与数据库层清晰分界。

4. 性能问题

- 假设要向数据库写入1000条SQL语句，采用JDBC编程效率低下。
- 采用ORM技术，会自动延迟向后台数据库发送SQL请求，降低通讯量，提高运行效率，也可以根据实际情况，将数据库访问操作合成，尽量减少不必要的数据库操作请求。

网络部分

1. http 响应码 301 和 302 代表的是什么？有什么区别?

1. 对用户：

- 301，302对用户来说没有区别，他们看到效果只是一个跳转，浏览器中旧的URL变成了新的URL。页面跳到了这个新的url指向的地方。

2. 对开发人员：

- 当网页A用301重定向转到网页B时，搜索引擎可以肯定网页A永久的改变位置，或者说实际上不存在了，搜索引擎就会把网页B当作唯一有效目标。
- 302转向可能会有URL规范化及网址劫持的问题。可能被搜索引擎判为可疑转向，甚至认为是作弊。

302重定向和网址劫持（URL hijacking）有什么关系呢?

- 这要从搜索引擎如何处理302转向说起。从定义来说，从网址A做一个302重定向到网址B时，意思是网址A随时有可能改主意，重新显示本身的内容或转向其他的地方。大部分的搜索引擎在大部分情况下，当收到302重定向时，一般只要去抓取目标网址就可以了，也就是说网址B。
- 实际上如果搜索引擎在遇到302转向时，百分之百的都抓取目标网址B的话，就不用担心网址URL劫持了。问题就在于，有的时候搜索引擎，并不能总是抓取目标网址。比如说Google，有的时候A网址很短，但是它做了一个302重定向到B网址，而B网址是一个很长的乱七八糟的URL网址，甚至还有可能包含一些问号之类的参数。很自然的，A网址更加用户友好，而B网址既难看，又不用户友好。这时Google很有可能会仍然显示网址A。这就造成了网址URL劫持的可能性。也就是说，一个不道德的人在他自己的网址A做一个302重定向到你的网址B，出

于某种原因， Google搜索结果所显示的仍然是网址A，但是所用的网页内容却是你的网址B上的内容，这种情况就叫做网址URL劫持。

2. 简述 tcp 和 udp的区别？tcp如何保证可靠传输？

1. TCP面向连接(发送数据要先连接)，可靠的数据传输流(不重复，不丢失)，传输单位为TCP报文段，全双工，占用资源多，每条TCP连接只能是点对点。TCP对应的协议有：FTP(文件传输协议)、telnet(远程登录）、SMTP(发送邮件协议)、POP3(接受邮件协议)
2. UDP面向非连接(发送数据不需要先连接)，不可靠的数据流传输，传输单位为用户数据包，实时性高，UDP支持一对一、一对多、多对多。UDP对应的协议有：DNS(域名解析协议)、SNMP(简单网络管理)、TFTP(简单文件传输)

   tcp如何保证可靠传输

   报文段确认：TCP发出一个报文段后，启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到确认，将重发这个报文段。

   排序：TCP给发送的每个包进行编号，接收方对数据包进行排序，有序传给应用层

   校验和：TCP将保持它首部与数据的校验和

   去重：TCP会丢掉重复数据

   流量控制：TCP连接的每一方都有一定空间的缓冲区，TCP的接收端只允许发送端发送接收端所能接纳的数据。当接收方不能及时处理数据时，则会提示发送方法降低发送速率，防止丢包，TCP使用的流量控制是可变大小的滑窗协议。

   拥塞控制：当网络拥塞时，减少数据发送

3. tcp 为什么要三次握手，两次不行吗？为什么？四次挥手又是什么？

采用三次握手是防止失效的连接请求报文段又传回服务器，从而产生错误。为了实现可靠数据传输， TCP 协议的通信双方， 都必须维护一个序列号， 以标识发送出去的数据包中， 哪些是已经被对方收到的。 三次握手的过程即是通信双方相互告知序列号起始值， 并确认对方已经收到了序列号起始值的必经步骤。如果只是两次握手， 至多只有连接发起方的起始序列号能被确认， 另一方选择的序列号则得不到确认。

失效的连接请求指的是：主机A发送的连接请求没有收到主机B的确认，于是经过一段时间，主机A又重新向B发送请求，并且连接成功，完成了数据传输。如果此时A的第一次的连接请求是因为网络延迟而又到达B，则B会认为A又发起了的连接，于是B会向A发送确认连接的信息，但A此时不会理会，B却一直等待A发送数据。

三次握手：

- 第一次握手：客户端发送同步序列编号包(syn，令此时的syn=x)到服务器，并进入SYN_SEND状态，等待服务器确认；
- 第二次握手：服务器收到syn包，将此syn包中的x进行加1并变为确认字符包ack，同时自己也发送一个syn包(令此包的syn=y)，因此服务器将发送ack+syn包
- 第三次握手：客户端收到服务器的ack+syn包后，只需要向服务器发送确认字符包ack(将服务器的syn+1)，此包发送完毕，客户端与服务器进入established状态，完成三次握手。

四次挥手：(连接终止协议)

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。(FIN表示关闭连接，是TCP层Flags字段中表示状态的标识)SYN, FIN, ACK, PSH(表示有 DATA数据传输)，, RSTRST(表示连接重置), URG.

- TCP客户端发送一个FIN，用来关闭客户到服务器的数据传送
- 服务器收到这个FIN，它发回一个ACK，确认序号为收到的序号加1。和SYN一样，一个FIN将占用一个序号。
- 服务器关闭客户端的连接，发送一个FIN给客户端。

- 客户端发回ACK确认，并将确认序号设置为收到序号加1。

4. 说一下 tcp 粘包是怎么产生的?

粘包是什么?

粘包发生在发送或接收缓冲区中；应用程序从缓冲区中取数据是整个缓冲区中有多少取多少；那么就有可能第一个数据的尾部和第二个数据的头部同时存在缓冲区，而TCP是流式的，数据无边界，这时发生粘包。

粘包的产生

1. 发送方产生粘包
   采用TCP协议传输数据的客户端与服务器经常是保持一个长连接的状态（一次连接发一次数据不存在粘包），双方在连接不断开的情况下，可以一直传输数据；但当发送的数据包过于的小时，那么TCP协议默认的会启用Nagle算法，将这些较小的数据包进行合并发送（缓冲区数据发送是一个堆压的过程）；这个合并过程就是在发送缓冲区中进行的，也就是说数据发送出来它已经是粘包的状态了；
2. 接收方产生粘包
   接收方采用TCP协议接收数据时的过程是这样的：数据到底接收方，从网络模型的下方传递至传输层，传输层的TCP协议处理是将其放置接收缓冲区，然后由应用层来主动获取（C语言用recv、read等函数）；这时会出现一个问题，就是我们在程序中调用的读取数据函数不能及时的把缓冲区中的数据拿出来，而下一个数据又到来并有一部分放入的缓冲区末尾，等我们读取数据时就是一个粘包；（放数据的速度 > 应用层拿数据速度）
   粘包的解决方案
3. 解决发送方粘包

- 发送产生是因为Nagle算法合并小数据包，那么可以禁用掉该算法；
- TCP提供了强制数据立即传送的操作指令push，当填入数据后调用操作指令就可以立即将数据发送，而不必等待发送缓冲区填充自动发送；
- 数据包中加头，头部信息为整个数据的长度（最广泛最常用）；

2. 解决接收方粘包

- 解析数据包头部信息，根据长度来接收；
- 自定义数据格式：在数据中放入开始、结束标识；解析时根据格式抓取数据，缺点是数据内不能含有开始或结束标识；
- 短连接传输，建立一次连接只传输一次数据就关闭；（不推荐）

5. OSI 的七层模型都有哪些？TCP/IP四层有哪些？五层协议又有哪些？七层模型中各层都有哪些协议?

OSI：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层
TCP/IP：网络接口层、网际层、运输层、应用层
五层：物理层、数据链路层、网络层、运输层、应用层

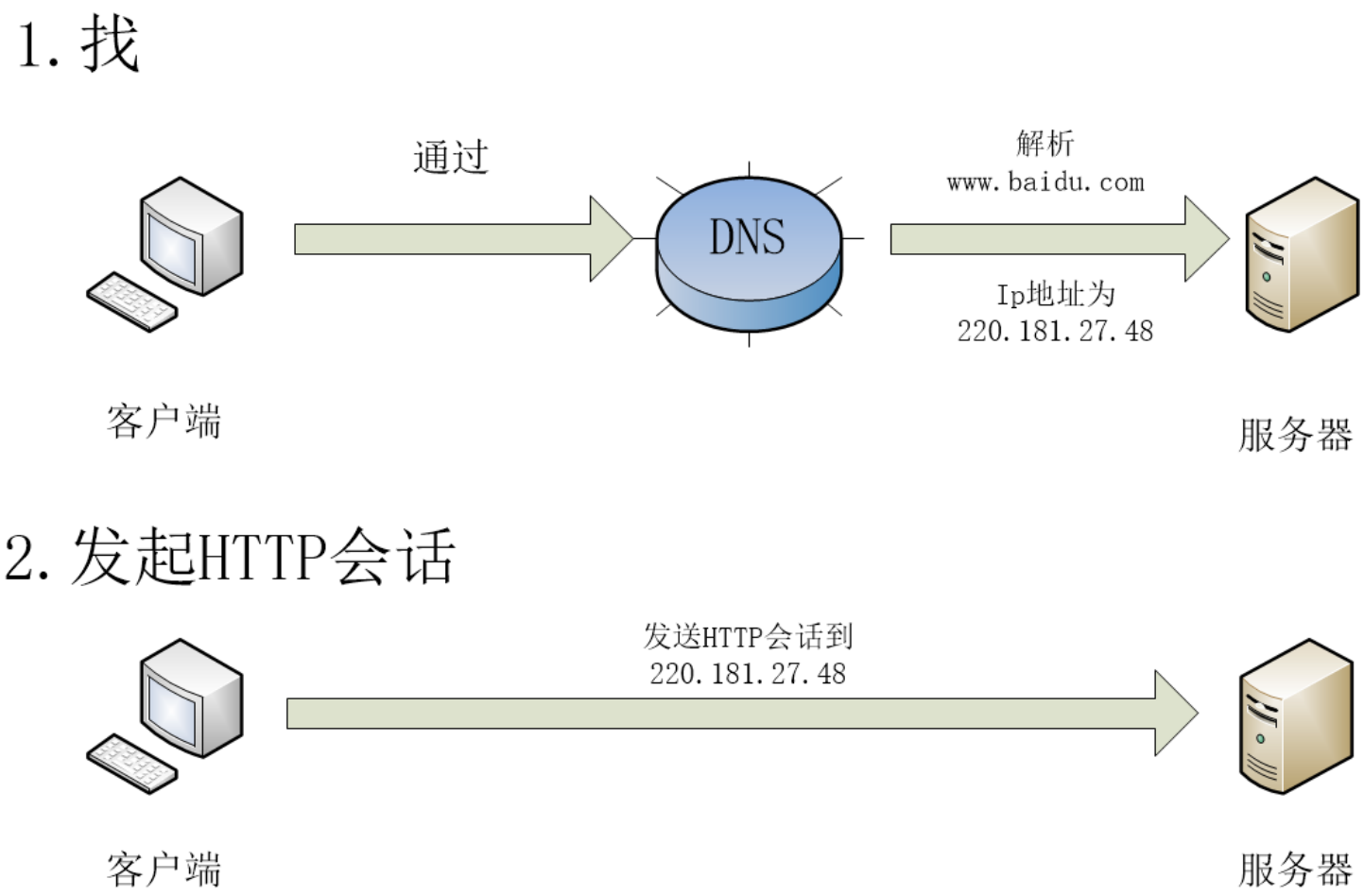
物理层：RJ45、CLOCK、IEEE802.3 比特传输
数据链路层：PPP、VLAN、MAC等 帧传输
网络层：IP、ICMP、ARP、OSPF等 包传输
传输层：TCP、UDP、SPX 段传输
会话层：NFS、SQL、RPC等 会话协议数据单元传输
表示层：JPEG、MPEG、ASII 等 表示协议数据单元传输
应用层：FTP、DNS、HTTP等 应用协议数据单元传输

6. 讲述ARP协议?

1. 每个主机都会在自己的ARP缓冲区中建立ARP列表，表示IP与MAC地址之间的关系

2. 当主机发送数据时，首先先检查ARP列表中是否存在对应IP地址目的主机的MAC地址，如果不存在，就向本网段的所有主机发送ARP数据包。该数据包包括：源主机IP地址，源主机MAC地址，目的主机IP地址。
3. 当本网段所有主机都收到了该ARP数据包，先检查数据包中IP地址是否是自己的IP，如果不是，则忽略。如果是，则先从数据包中取出源主机IP和MAC写入到ARP列表中，如果存在，则覆盖，然后将自己的MAC地址写入ARP响应中，告诉源主机自己是要找的MAC地址。
4. 源主机收到ARP响应后，将目的主机的IP和MAC地址写入ARP列表中，并利用此信息发送数据。如果源主机一直没有收到ARP相应数据包，则表示ARP查询失败。广播ARP请求，单播ARP响应。

7. 描述浏览器输入 www.baidu.com 访问的过程?



网络地址解析

发送HTTP会话到220.181.27.48过程拆解

1. 客户端通过TCP进行封装数据包，输入到网络层，网络层通过查找路由表确定如何达到服务器
2. 在客户端传输层，把HTTP请求分成报文段，添加源和目的端口。服务端使用80端口监听客户端请求，客户端由系统随机选择一个端口（假设5000）与服务器交互，服务器将相应的请求返回给客户端5000端口，然后使IP找到目的端。
3. 在客户端链路层，包通过链路层发送到路由器，通过邻居协议查找给定的IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换IP数据包，就可以传输了，然后发送IP数据包到服务器地址。

8. DNS系统原理?

当一个应用进程需要把某个域名解析为IP地址时，该应用进程会调用解析程序，并成为一个DNS用户，把待解析的域名放在DNS请求报文中，以UDP数据报的形式发送给本地域名服务器，本地域名服务器查找到相应域名的IP地址后，就将该域名IP地址信息放入应答报文中返回给客户端进程，如果本地域名服务器没有直接查找到对应IP地址，则向根域名服务器发出迭代查询，再将查询到的IP地址信息传回客户端程序。

Spring部分

1. 为什么要使用 spring？解释一下什么是 aop？解释一下什么是 ioc？

Spring是一个轻量级控制反转(IoC)和面向切面(AOP)的容器框架，为了解决软件开发的复杂性而创建的，使用基本的JavaBean代替EJB，并提供了更多的企业应用功能。可在任何Java应用中使用。

AOP：

- AOP，可以说是OOP的补充和完善。OOP引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。当我们需要为分散的对象引入公共行为的时候，OOP则显得无能为力。也就是说，**OOP允许你定义从上到下的关系，但并不适合定义从左到右的关系。**例如日志功能。日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。对于其他类型的代码，如安全性、异常处理和透明的持续性也是如此。这种散布在各处的无关的代码被称为横切代码，在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。
- 而AOP技术则恰恰相反，它利用一种称为“横切”的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其名为 “Aspect”，即方面。所谓“方面”，简单地说，就是将那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。AOP代表的是一个横向的关系，如果说“对象”是一个空心的圆柱体，其中封装的是对象的属性和行为； 那么面向方面编程的方法，就仿佛一把利刃，将这些空心圆柱体剖开，以获得其内部的消息。而剖开的切面，也就是所谓的“方面”了。然后它又以巧夺天功的妙手将这些剖开的切面复原，不留痕迹。
- 使用“横切”技术，AOP把软件系统分为两个部分：**核心关注点和横切关注点。**业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横 切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处都基本相似。比如权限认证、日志、事务处理。Aop 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。正如Avanade公司的高级方案构架师Adam Magee所说，AOP的核心思想就是“将应用程序中的商业逻辑同对其提供支持的通用服务进行分离。”
  AOP的实现：
  实现AOP的技术，主要分为两大类：一是采用**动态代理技术**，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；二是采用**静态织入**的 方式，引入特定的语法创建“方面”，从而使得编译器可以在编译期间织入有关“方面”的代码。然而殊途同归，实现AOP的技术特性却是相同的，分别为：
- join point（连接点）：是程序执行中的一个精确执行点，例如类中的一个方法。它是一个抽象的概念，在实现AOP时，并不需要去定义一个join point。
- point cut（切入点）：本质上是一个捕获连接点的结构。在AOP中，可以定义一个point cut，来捕获相关方法的调用。
- advice（通知）：是point cut的执行代码，是执行“方面”的具体逻辑。
- aspect（方面）：point cut和advice结合起来就是aspect，它类似于OOP中定义的一个类，但它代表的更多是对象间横向的关系。
- introduce（引入）：为对象引入附加的方法或属性，从而达到修改对象结构的目的。有的AOP工具又将其称为mixin。

IoC：

2. spring 有哪些主要模块?

七个定义明确的模块组成:

核心容器

这是Spring框架最基础的部分，它提供了依赖注入（DependencyInjection）特征来实现容器对Bean的管理。这里最基本的概念是BeanFactory，它是任何Spring应用的核心。BeanFactory是工厂模式的一个实现，它使用IoC将应用配置和依赖说明从实际的应用代码中分离出来。

应用上下文（Context）模块

核心模块的BeanFactory使Spring成为一个容器，而上下文模块使它成为一个框架。这个模块扩展了BeanFactory的概念，增加了对国际化（I18N）消息、事件传播以及验证的支持。另外，这个模块提供了许多企业服务，例如电子邮件、JNDI访问、EJB集成、远程以及时序调度（scheduling）服务。也包括了对模版框架例如Velocity和FreeMarker集成的支持。

Spring的AOP模块

Spring在它的AOP模块中提供了对面向切面编程的丰富支持。这个模块是在Spring应用中实现切面编程的基础。为了确保Spring与其它AOP框架的互用性，Spring的AOP支持基于AOP联盟定义的API。AOP联盟是一个开源项目，它的目标是通过定义一组共同的接口和组件来促进AOP的使用以及不同的AOP实现之间的互用性。通过访问他们的站点，你可以找到关于AOP联盟的更多内容。Spring的AOP模块也将元数据编程引入了Spring。使用Spring的元数据支持，你可以为你的源代码增加注释，指示Spring在何处以及如何应用切面函数。

JDBC抽象和DAO模块

使用JDBC经常导致大量的重复代码，取得连接、创建语句、处理结果集，然后关闭连接。Spring的JDBC和DAO模块抽取了这些重复代码，因此你可以保持你的数据库访问代码干净简洁，并且可以防止因关闭数据库资源失败而引起的问题。这个模块还在几种数据库服务器给出的错误消息之上建立了一个有意义的异常层。使你不用再试图破译神秘的私有的SQL错误消息！另外，这个模块还使用了Spring的AOP模块为Spring应用中的对象提供了事务管理服务。

对象/关系映射集成模块

对那些更喜欢使用对象/关系映射工具而不是直接使用JDBC的人，Spring提供了ORM模块。Spring并不试图实现它自己的ORM解决方案，而是为几种流行的ORM框架提供了集成方案，包括Hibernate、JDO和iBATIS SQL映射。Spring的事务管理支持这些ORM框架中的每一个也包括JDBC。

Spring的Web模块

Web上下文模块建立于应用上下文模块之上，提供了一个适合于Web应用的上下文。另外，这个模块还提供了一些面向服务支持。例如：实现文件上传的multipart请求，它也提供了Spring和其它Web框架的集成，比如Struts、WebWork。

Spring的MVC框架

Spring为构建Web应用提供了一个功能全面的MVC框架。虽然Spring可以很容易地与其它MVC框架集成，例如Struts，但Spring的MVC框架使用IoC对控制逻辑和业务对象提供了完全的分离。它也允许你声明性地将请求参数绑定到你的业务对象中，此外，Spring的MVC框架还可以利用Spring的任何其它服务，例如国际化信息与验证。

3. spring 常用的注入方式有哪些?

常用的注入方式主要有三种：构造方法注入，setter注入，基于注解的注入，通常使用基于注解注入。

基于注解的注入

在介绍注解注入的方式前，先简单了解bean的一个属性autowire，autowire主要有三个属性值：constructor，byName，byType。

- constructor：通过构造方法进行自动注入，spring会匹配与构造方法参数类型一致的bean进行注入，如果有一个多参数的构造方法，一个只有一个参数的构造方法，在容器中查找到多个匹配多参数构造方法的bean，那么spring会优先将bean注入到多参数的构造方法中。
- byName：被注入bean的id名必须与set方法后半截匹配，并且id名称的第一个单词首字母必须小写，这一点与手动set注入有点不同。

- byType：查找所有的set方法，将符合符合参数类型的bean注入。

下面进入正题：注解方式注册bean，注入依赖 。主要有四种注解可以注册bean，每种注解可以任意使用，只是语义上有所差异：

1. @Component：可以用于注册所有bean
2. @Repository：主要用于注册dao层的bean
3. @Controller：主要用于注册控制层的bean
4. @Service：主要用于注册服务层的bean

描述依赖关系主要有两种：

- @Resource：java的注解，默认以byName的方式去匹配与属性名相同的bean的id，如果没有找到就会以byType的方式查找，如果byType查找到多个的话，使用@Qualifier注解（spring注解）指定某个具体名称的bean。

```
@Resource
@Qualifier("userDaoMyBatis")
private IUserDao userDao;

public UserService(){
```

- @Autowired：spring注解，默认是以byType的方式去匹配与属性名相同的bean的id，如果没有找到，就通过byName的方式去查找，

```
@Autowired
@Qualifier("userDaoJdbc")
private IUserDao userDao;
```

4. spring 中的 bean 是线程安全的吗？spring 支持几种 bean 的作用域?

Spring容器中的Bean是否线程安全，容器本身并没有提供Bean的线程安全策略，因此可以说spring容器中的Bean本身不具备线程安全的特性，但是具体还是要结合具体scope的Bean去研究。

spring 的 bean 有哪些scope[作用域]

1. singleton:单例。【默认】
2. prototype:原型，每次创建一个新对象
3. request:请求，每次Http请求创建一个新对象，适用于WebApplicationContext环境下。
4. session:会话，同一个会话共享一个实例。不同会话使用不用的实例。
5. global-session:全局会话，所有会话共享一个实例。

- 单例Bean,所有线程都共享一个单例实例Bean,因此是存在资源的竞争。如果单例Bean,是一个无状态Bean，那么线程安全。
- 对于原型Bean,线程安全。
- 比如spring mvc 的 Controller、 Service、 Dao等，这些Bean大多是无状态的，只关注于方法本身。

- 对于有状态的bean，spring官方提供的bean，一般提供了通过ThreadLocal去解决线程安全的方法。比如RequestContextHolder、TransactionSynchronizationManager、LocaleContextHolder等。使用ThreadLocal的好处是使得多线程场景下，多个线程对这个单例Bean的成员变量并不存在资源的竞争，因为ThreadLocal为每个线程保存线程私有的数据。这是一种以空间换时间的方式。当然也可以通过加锁的方法来解决线程安全，这种以时间换空间的场景在高并发场景下显然是不实际的。

5. spring 自动装配 bean 有哪些方式?

6. spring 事务实现方式有哪些? 说一下 spring 的事务隔离?

7. 说一下 spring mvc 运行流程?

8. spring mvc 有哪些组件?

9. @RequestMapping 的作用是什么?

10. @Autowired 的作用是什么?

11.Spring框架Web页面乱码问题的解决?

可以采用Spring框架自带的过滤器CharacterEncodingFilter，这样可以大大减轻了我们的工作量，即简单方便又容易理解，配置方式如下：在web.xml文件中filter的位置加上如下内容：

```
<filter>
    <filter-name>encodingFilter</filter-name >
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>*</url-pattern>
</filter-mapping>
```

Hibernate部分

1. hibernate 中如何在控制台查看打印的 sql 语句?

- 可以在Hibernate.cfg.xml文件中配置
  <property name="show_sql">true</property>
- 也可以在Spring配置文件中设置
  <prop key="hibernate.show_sql">true</prop>

2. hibernate 有几种查询方式?

hibernate的查询方式常见的主要分为三种: HQL, QBC（命名查询）, 以及使用原生SQL查询（SqlQuery）

- HQL在语法结构上和SQL语句十分的相同，所以可以很快的上手进行使用。使用HQL需要用到Hibernate中的Query对象，该对象专门执行HQL方式的操作。
- QBC（Query By Criteria）查询，Criteria对象提供了一种面向对象的方式查询数据库。Criteria对象需要使用Session对象来获得。一个Criteria对象表示对一个持久化类的查询。

3. hibernate 实体类可以被定义为 final 吗?

- 可以将Hibernate的实体类定义为final类，但这种做法并不好。因为Hibernate会使用代理模式在延迟关联的情况下提高性能，如果你把实体类定义成final类之后，因为 Java不允许对final类进行扩展，所以Hibernate就无法再使用代理了，如此一来就限制了使用可以提升性能的手段。
- 不过，如果持久化类实现了一个接口而且在该接口中声明了所有定义于实体类中的所有public的方法轮到话，就能够避免出现前面所说的不利后果。

4. 在 hibernate 中使用 Integer 和 int 做映射有什么区别?

1. 返回数据库字段值是null的话，int类型会报错。int是基本数据类型，其声明的是变量，而null则是对象。所以hibernate实体建议用integer；
2. 通过jdbc将实体存储到数据库的操作通过sql语句，基本数据类型可以直接存储，对象需要序列化存储。
3. 在很多应用中，需要对某些对象进行序列化，让它们离开内存空间，入住物理硬盘，以便长期保存。比如最常见的是Web服务器中的Session对象，当有 10万用户并发访问，就有可能出现10万个Session对象，内存可能吃不消，于是Web容器就会把一些seesion先序列化到硬盘中，等要用了，再把保存在硬盘中的对象还原到内存中。使用ObjectOutputStream和ObjectInputStream来实现序列化。

5. hibernate 是如何工作的? 为什么用hibernate?

hibernate核心接口

- session：负责被持久化对象CRUD操作
- sessionFactory:负责初始化hibernate，创建session对象
- configuration:负责配置并启动hibernate，创建SessionFactory
- Transaction:负责事物相关的操作
- Query和Criteria接口：负责执行各种数据库查询

hibernate工作原理:

1. 通过Configuration config = new Configuration().configure();//读取并解析hibernate.cfg.xml配置文件
2. 由hibernate.cfg.xml中的<mapping resource="com/xx/User.hbm.xml"/>读取并解析映射信息
3. 通过SessionFactory sf = config.buildSessionFactory();//创建SessionFactory
4. Session session = sf.openSession();//打开Sesssion
5. Transaction tx = session.beginTransaction();//创建并启动事务Transation
6. persistent operate操作数据，持久化操作
7. tx.commit();//提交事务
8. 关闭Session
9. 关闭SesstionFactory

6. get()和 load()的区别?

1. 对于Hibernate get方法，Hibernate会确认一下该id对应的数据是否存在，首先在session缓存中查找，然后在二级缓存中查找，还没有就查询数据库，数据 库中没有就返回null。这个相对比较简单，也没有太大的争议。主要要说明的一点就是在这个版本(bibernate3.2以上)中get方法也会查找二级缓存!
2. Hibernate load方法加载实体对象的时候，根据映射文件上类级别的lazy属性的配置(默认为true)，分情况讨论:

- 若为true,则首先在Session缓存中查找，看看该id对应的对象是否存在，不存在则使用延迟加载，返回实体的代理类对象(该代理类为实体类的子类，由CGLIB动态生成)。等到具体使用该对象(除获取OID以外)的时候，再查询二级缓存和数据库，若仍没发现符合条件的记录，则会抛出一个ObjectNotFoundException。
- 若为false,就跟Hibernateget方法查找顺序一样，只是最终若没发现符合条件的记录，则会抛出一个ObjectNotFoundException。

_这里get和load有两个重要区别: _

- 如果未能发现符合条件的记录，Hibernate get方法返回null，而load方法会抛出一个ObjectNotFoundException。
- load方法可返回没有加载实体数据的代 理类实例，而get方法永远返回有实体数据的对象。

总之对于get和load的根本区别，hibernate对于 load方法认为该数据在数据库中一定存在，可以放心的使用代理来延迟加载，如果在使用过程中发现了问题，只能抛异常；而对于get方 法，hibernate一定要获取到真实的数据，否则返回null。

7. 说一下 hibernate 的缓存机制?

为什么要用 Hibernate缓存?

Hibernate是一个持久层框架，经常访问物理数据库。为了降低应用程序对物理数据源访问的频次，从而提高应用程序的运行性能。缓存内的数据是对物理数据源中的数据的复制，应用程序在运行时从缓存读写数据，在特定的时刻或事件会同步缓存和物理数据源的数据。为了提供访问速度，把磁盘或数据库访问变成内存访问。

Hibernate缓存原理是怎样的?

Hibernate缓存包括两大类：Hibernate一级缓存和 Hibernate二级缓存。

1. Hibernate一级缓存又称为“Session的缓存”。Session缓存内置不能被卸载，Session的缓存是事务范围的缓存（Session对象的生命周期通常对应一个数据库事务或者一个应用事务）。一级缓存中，持久化类的每个实例都具有唯一的 OID。
2. Hibernate二级缓存又称为“SessionFactory的缓存”。由于 SessionFactory 对象的生命周期和应用程序的整个过程对应，因此 Hibernate 二级缓存是进程范围或者集群范围的缓存，有可能出现并发问题，因此需要采用适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。第二级缓存是可选的，是一个可配置的插件，默认下 SessionFactory不会启用这个插件。Hibernate提供了 org.hibernate.cache.CacheProvider接口,它充当缓存插件与 Hibernate之间的适配器。

总结一下:

Hibernate中的缓存分一级缓存和二级缓存。

- 一级缓存就是 Session 级别的缓存，在事务范围内有效是,内置的不能被卸载。二级缓存是 SesionFactory级别的缓存，从应用启动到应用结束有效。是可选的，默认没有二级缓存，需要手动开启。保存数据库后，缓存在内存中保存一份，如果更新了数据库就要同步更新。

什么样的数据适合存放到第二级缓存中?

1. 很少被修改的数据 帖子的最后回复时间
2. 经常被查询的数据 电商的地点

3. 不是很重要的数据，允许出现偶尔并发的数据
4. 不会被并发访问的数据
5. 常量数据

8. hibernate 对象有哪些状态?

hibernate里对象有三种状态:

1. Transient 瞬时 ： 对象刚new出来，还没设id，设了其他值。
2. Persistent 持久：调用了save()、saveOrUpdate()，就变成Persistent，有id
3. Detached 脱管 ： 当session close()完之后，变成Detached。

9. 在 hibernate 中 getCurrentSession 和 openSession 的区别是什么?

1. openSession 从字面上可以看得出来，是打开一个新的session对象，而且每次使用都是打开一个新的session，假如连续使用多次，则获得的session不是同一个对象，并且使用完需要调用close方法关闭session。
2. getCurrentSession ，从字面上可以看得出来，是获取当前上下文一个session对象，当第一次使用此方法时，会自动产生一个session对象，并且连续使用多次时，得到的session都是同一个对象，这就是与openSession的区别之一，简单而言，getCurrentSession 就是：如果有已经使用的，用旧的，如果没有，建新的。

- 在实际开发中，往往使用getCurrentSession多，因为一般是处理同一个事务（即是使用一个数据库的情况），所以在一般情况下比较少使用openSession或者说openSession是比较老旧的一套接口了;

getCurrentSession有以下一些特点:

1. 用途，界定事务边界
2. 事务提交会自动close，不需要像openSession一样自己调用close方法关闭session
3. 上下文配置即在hibernate.cfg.xml中，需要配置:

```
<property name="current_session_context_class">thread</property>
(需要注意，这里的current_session_context_class属性有几个属性值：jta 、 thread 常用 ， custom、m
```

- thread使用connection 单数据库连接管理事务
- jta Java transaction api 分布式事务管理 （多数据库访问），jta 由中间件提供（JBoss WebLogic 等， 但是tomcat 不支持）

10. hibernate 实体类必须要有无参构造函数吗? 为什么?

因为hibernate框架会调用这个默认构造方法来构造实例对象。即Class类的newInstance方法 这个方法就是通过调用默认构造方法来创建实例对象的 。如果没有提供任何构造方法，虚拟机会自动提供默认构造方法（无参构造器），但是如果你提供了其他有参数的构造方法的话，虚拟机就不再为你提供默认构造方法，这时必须手动把无参构造器写在代码里，否则new Xxxx()是会报错的，所以默认的构造方法不是必须的，只在有多个构造方法时才是必须的，这里“必须”指的是“必须手动写出来”。

MyBatis部分

1. mybatis 中 #{}和 ${}的区别是什么?

1. #是将传入的值当做字符串的形式，eg:select id,name,age from student where id =#{id},当前端把id值1，传入到后台的时候，就相当于 select id,name,age from student where id ='1'.
2. $是将传入的数据直接显示生成sql语句，eg:select id,name,age from student where id =${id},当前端把id值1，传入到后台的时候，就相当于 select id,name,age from student where id = 1.
3. 使用#可以很大程度上防止sql注入。(语句的拼接)
4. 但是如果使用在order by 中就需要使用 $.
5. 在大多数情况下还是经常使用#，但在不同情况下必须使用$.
6. #与的区别最大在于：#{} 传入值时，sql解析时，参数是带引号的，而{}传入值，sql解析时，参数是不带引号的。

在mybatis中的$与#都是在sql中动态的传入参数。

```
eg:select id,name,age from student where name=#{name}
```

这个name是动态的，可变的。当你传入什么样的值，就会根据你传入的值执行sql语句。

- #{}: 解析为一个 JDBC 预编译语句（prepared statement）的参数标记符，一个 #{ } 被解析为一个参数占位符 。
- ${}: 仅仅为一个纯碎的 string 替换，在动态 SQL 解析阶段将会进行变量替换。
  name-->cy

```
select id,name,age from student where name=${name}    -- name=cy
```

2. mybatis 有几种分页方式?

1. 数组分页
   查询出全部数据，然后再list中截取需要的部分。

- *mybatis接口*

```
List<Student> queryStudentsByArray();
```

- *xml配置文件*

```
<select id="queryStudentsByArray" resultMap="studentmapper"> select * from student </select
```

- *service*

接口

```
List<Student> queryStudentsByArray(int currPage, int pageSize);
```

实现接口

```
 @Override
    public List<Student> queryStudentsByArray(int currPage, int pageSize) {
        //查询全部数据
        List<Student> students = studentMapper.queryStudentsByArray();
        //从第几条数据开始
        int firstIndex = (currPage - 1) * pageSize;
        //到第几条数据结束
        int lastIndex = currPage * pageSize;
        return students.subList(firstIndex, lastIndex); //直接在list中截取
    }
```

- *controller*

```
@ResponseBody
    @RequestMapping("/student/array/{currPage}/{pageSize}") public List<Student> getStudent
        List<Student> student = StuServiceIml.queryStudentsByArray(currPage, pageSize); ret
    }
```

2. sql分页

mybatis接口

```
List<Student> queryStudentsBySql(Map<String,Object> data);
```

xml文件

```
<select id="queryStudentsBySql" parameterType="map" resultMap="studentmapper"> select * fro
```

service

接口

```
List<Student> queryStudentsBySql(int currPage, int pageSize);
```

实现类

```
public List<Student> queryStudentsBySql(int currPage, int pageSize) {
        Map<String, Object> data = new HashedMap();
        data.put("currIndex", (currPage-1)*pageSize);
        data.put("pageSize", pageSize);
```

```
        return studentMapper.queryStudentsBySql(data);
    }
```

3. 拦截器分页
   创建拦截器，拦截mybatis接口方法id以ByPage结束的语句

```
package com.autumn.interceptor;

import org.apache.ibatis.executor.Executor;
import org.apache.ibatis.executor.parameter.ParameterHandler;
import org.apache.ibatis.executor.resultset.ResultSetHandler;
import org.apache.ibatis.executor.statement.StatementHandler;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.plugin.*;
import org.apache.ibatis.reflection.MetaObject;
import org.apache.ibatis.reflection.SystemMetaObject;

import java.sql.Connection;
import java.util.Map;
import java.util.Properties;

/**
 * @Intercepts 说明是一个拦截器
 * @Signature 拦截器的签名
 * type 拦截的类型 四大对象之一( Executor,ResultSetHandler,ParameterHandler,StatementHandler)
 * method 拦截的方法
 * args 参数,高版本需要加个Integer.class参数,不然会报错
 */
@Intercepts({@Signature(type = StatementHandler.class, method = "prepare", args = {Connecti
public class MyPageInterceptor implements Interceptor {

    //每页显示的条目数
    private int pageSize;
    //当前现实的页数
    private int currPage;
    //数据库类型
    private String dbType;


    @Override
    public Object intercept(Invocation invocation) throws Throwable {
        //获取StatementHandler, 默认是RoutingStatementHandler
        StatementHandler statementHandler = (StatementHandler) invocation.getTarget();
        //获取statementHandler包装类
        MetaObject MetaObjectHandler = SystemMetaObject.forObject(statementHandler);

        //分离代理对象链
```

```
        while (MetaObjectHandler.hasGetter("h")) {
            Object obj = MetaObjectHandler.getValue("h");
            MetaObjectHandler = SystemMetaObject.forObject(obj);
        }

        while (MetaObjectHandler.hasGetter("target")) {
            Object obj = MetaObjectHandler.getValue("target");
            MetaObjectHandler = SystemMetaObject.forObject(obj);
        }

        //获取连接对象
        //Connection connection = (Connection) invocation.getArgs()[0];


        //object.getValue("delegate");  获取StatementHandler的实现类

        //获取查询接口映射的相关信息
        MappedStatement mappedStatement = (MappedStatement) MetaObjectHandler.getValue("del
        String mapId = mappedStatement.getId();

        //statementHandler.getBoundSql().getParameterObject();

        //拦截以.ByPage结尾的请求，分页功能的统一实现
        if (mapId.matches(".+ByPage$")) {
            //获取进行数据库操作时管理参数的handler
            ParameterHandler parameterHandler = (ParameterHandler) MetaObjectHandler.getVal
            //获取请求时的参数
            Map<String, Object> paraObject = (Map<String, Object>) parameterHandler.getPara
            //也可以这样获取
            //paraObject = (Map<String, Object>) statementHandler.getBoundSql().getParamete

            //参数名称和在service中设置到map中的名称一致
            currPage = (int) paraObject.get("currPage");
            pageSize = (int) paraObject.get("pageSize");

            String sql = (String) MetaObjectHandler.getValue("delegate.boundSql.sql");
            //也可以通过statementHandler直接获取
            //sql = statementHandler.getBoundSql().getSql();

            //构建分页功能的sql语句
            String limitSql;
            sql = sql.trim();
            limitSql = sql + " limit " + (currPage - 1) * pageSize + "," + pageSize;

            //将构建完成的分页sql语句赋值个体'delegate.boundSql.sql'，偷天换日
            MetaObjectHandler.setValue("delegate.boundSql.sql", limitSql);
        }
        //调用原对象的方法，进入责任链的下一级
        return invocation.proceed();
```

```
    }


    //获取代理对象
    @Override
    public Object plugin(Object o) {
        //生成object对象的动态代理对象
        return Plugin.wrap(o, this);
    }

    //设置代理对象的参数
    @Override
    public void setProperties(Properties properties) {
        //如果项目中分页的pageSize是统一的，也可以在这里统一配置和获取，这样就不用每次请求都传递pa
        String limit1 = properties.getProperty("limit", "10");
        this.pageSize = Integer.valueOf(limit1);
        this.dbType = properties.getProperty("dbType", "mysql");
    }
}
```

_ 配置文件SqlMapConfig.xml_

```
<configuration>

    <plugins>
        <plugin interceptor="com.autumn.interceptor.MyPageInterceptor">
            <property name="limit" value="10"/>
            <property name="dbType" value="mysql"/>
        </plugin>
    </plugins>

</configuration>
```

mybatis配置

```
<!--接口-->
List<AccountExt> getAllBookByPage(@Param("currPage")Integer pageNo,@Param("pageSize")Intege
<!--xml配置文件-->
  <sql id="getAllBooksql" >
    acc.id, acc.cateCode, cate_name, user_id,u.name as user_name, money, remark, time
  </sql>
  <select id="getAllBook" resultType="com.autumn.pojo.AccountExt" >
    select
    <include refid="getAllBooksql" />
```

```
    from account as acc
  </select>
```

service

```
public List<AccountExt> getAllBookByPage(String pageNo,String pageSize) {
        return accountMapper.getAllBookByPage(Integer.parseInt(pageNo),Integer.parseInt(pag
    }
```

controller

```
    @RequestMapping("/getAllBook")
    @ResponseBody public Page getAllBook(String pageNo,String pageSize,HttpServletRequest r
        pageNo=pageNo==null?"1":pageNo;   //当前页码
        pageSize=pageSize==null?"5":pageSize;   //页面大小 //获取当前页数据
        List<AccountExt> list = bookService.getAllBookByPage(pageNo,pageSize); //获取总数据大
        int totals = bookService.getAllBook(); //封装返回结果
        Page page = new Page();
        page.setTotal(totals+"");
        page.setRows(list); return page;
    }
```

Page实体类

```
package com.autumn.pojo;

import java.util.List;

/**
 * Created by Autumn on 2018/6/21.
 */
public class Page {
    private String pageNo = null;
    private String pageSize = null;
    private String total = null;
    private List rows = null;

    public String getTotal() {
        return total;
    }

    public void setTotal(String total) {
        this.total = total;
```

```
    }

    public List getRows() {
        return rows;
    }

    public void setRows(List rows) {
        this.rows = rows;
    }

    public String getPageNo() {
        return pageNo;
    }

    public void setPageNo(String pageNo) {
        this.pageNo = pageNo;
    }

    public String getPageSize() {
        return pageSize;
    }

    public void setPageSize(String pageSize) {
        this.pageSize = pageSize;
    }

}
```

4. RowBounds分页

数据量小时，RowBounds不失为一种好办法。但是数据量大时，实现拦截器就很有必要了。

mybatis接口加入RowBounds参数

```
public List<UserBean> queryUsersByPage(String userName, RowBounds rowBounds);
```

service

```
    @Override
    @Transactional(isolation = Isolation.READ_COMMITTED, propagation = Propagation.SUPPORTS
    public List<RoleBean> queryRolesByPage(String roleName, int start, int limit) {
        return roleDao.queryRolesByPage(roleName, new RowBounds(start, limit));
    }
```

4. mybatis 逻辑分页和物理分页的区别是什么?

- 逻辑分页 内存开销比较大,在数据量比较小的情况下效率比物理分页高;在数据量很大的情况下,内存开销过大,容易内存溢出,不建议使用。
- 物理分页 内存开销比较小,在数据量比较小的情况下效率比逻辑分页还是低,在数据量很大的情况下,建议使用物理分页。

5. mybatis 是否支持延迟加载?延迟加载的原理是什么?

1. 什么是延迟加载
   举个例子:如果查询订单并且关联查询用户信息。如果先查询订单信息即可满足要求,当我们需要查询用户信息时再查询用户信息。把对用户信息的按需去查询就是延迟加载。 所以延迟加载即先从单表查询、需要时再从关联表去关联查询,大大提高数据库性能,因为查询单表要比关联查询多张表速度要快。
2. 延迟加载对主对象都是直接加载,只有对关联对象是延迟加载。延迟加载可以减轻数据库的压力,延迟加载不可是一条SQL查询多表信息,这样构不成延迟加载,会形成直接加载。

MyBatis延迟加载分为三种类型:

1. 直接加载。执行完主对象之后,直接执行关联对象。
2. 侵入式加载。在执行主对象详情的时候,执行关联对象。
3. 深度延迟加载。执行完主对象或主对象详情不会执行关联对象,只有用到关联对象数据的时候才走深度延迟加载。
   延迟加载默认情况下是关闭状态(false) 延迟加载下的侵入式加载默认情况下是开启状态(true)在这种情况下延迟加载不生效, 比如说延迟加载是一个大的水龙头总闸,如果总闸没开,那么总闸里面的小的闸肯定没有水。如果想要使用延迟加载必须写成true。

6. 说一下 mybatis 的一级缓存和二级缓存?

1. 一级缓存,Mybatis对缓存提供支持,但是在没有配置的默认情况下,它只开启一级缓存,一级缓存只是相对于同一个SqlSession而言。所以在参数和SQL完全一样的情况下,我们使用同一个SqlSession对象调用一个Mapper方法,往往只执行一次SQL,因为使用SelSession第一次查询后,MyBatis会将其放在缓存中,以后再查询的时候,如果没有声明需要刷新,并且缓存没有超时的情况下,SqlSession都会取出当前缓存的数据,而不会再次发送SQL到数据库。
   一级缓存的生命周期有多长?

- MyBatis在开启一个数据库会话时,会 创建一个新的SqlSession对象,SqlSession对象中会有一个新的Executor对象。Executor对象中持有一个新的PerpetualCache对象;当会话结束时,SqlSession对象及其内部的Executor对象还有PerpetualCache对象也一并释放掉。
- 如果SqlSession调用了close()方法,会释放掉一级缓存PerpetualCache对象,一级缓存将不可用。
- 如果SqlSession调用了clearCache(),会清空PerpetualCache对象中的数据,但是该对象仍可使用。
- SqlSession中执行了任何一个update操作(update()、delete()、insert()) ,都会清空PerpetualCache对象的数据,但是该对象可以继续使用

2. MyBatis的二级缓存是Application级别的缓存,它可以提高对数据库查询的效率,以提高应用的性能。
   SqlSessionFactory层面上的二级缓存默认是不开启的,二级缓存的开席需要进行配置,实现二级缓存的时候,MyBatis要求返回的POJO必须是可序列化的。 也就是要求实现Serializable接口,配置方法很简单,只需要在映射XML文件配置就可以开启缓存了<cache/>,如果我们配置了二级缓存就意味着:

- 映射语句文件中的所有select语句将会被缓存。

- 映射语句文件中的所欲insert、update和delete语句会刷新缓存。
- 缓存会使用默认的Least Recently Used（LRU，最近最少使用的）算法来收回。
- 根据时间表，比如No Flush Interval，（CNFI没有刷新间隔），缓存不会以任何时间顺序来刷新。
- 缓存会存储列表集合或对象(无论查询方法返回什么)的1024个引用
- 缓存会被视为是read/write(可读/可写)的缓存，意味着对象检索不是共享的，而且可以安全的被调用者修改，不干扰其他调用者或线程所做的潜在修改。

7. mybatis 和 hibernate 的区别有哪些?

1. Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句，不过mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。
2. Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。
3. Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的缺点是学习门槛高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。

8. mybatis 有哪些执行器（Executor）？

Mybatis有三种基本的Executor执行器:SimpleExecutor、ReuseExecutor、BatchExecutor。

- SimpleExecutor：每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象。
- ReuseExecutor：执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map内，供下一次使用。简言之，就是重复使用Statement对象。
- BatchExecutor：执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，等待逐一执行executeBatch()批处理。与JDBC批处理相同。
- 作用范围：Executor的这些特点，都严格限制在SqlSession生命周期范围内。

9. mybatis 分页插件的实现原理是什么？ 如何编写一个自定义插件?

为什么要有插件
可以在映射语句执行前后加一些自定义的操作，比如缓存、分页等
可以拦截哪些方法
默认情况下，Mybatis允许使用插件来拦截的方法有：

- Executor：update、query、flushStatements、commit、rollback、getTransaction、close、isClosed。 实现类：SimpleExecutor/BatchExecutor/ReuseExecutor/CachingExecutor
- ParameterHandler：getParameterObject、setParameters。 实现类：DefaultParameterHandler
- ResultSetHandler：handleResultSets、handleOutputParameters。 实现类：DefaultResultSetHandler
- StatementHandler：prepare、parameterize、batch、update、query。 实现类：CallableStatementHandler/PreparedStatementHandler/SimpleStatementHandler/RoutingStatementHandler
  如何自定义插件
  只需实现Interceptor接口，并指定要拦截的方法签名。

```
    @Intercepts({
    @Signature(
        type=Executor.class,method="update",args={ MappedStatement.class,Object.class })
})
public class ExamplePlugin implements Interceptor {
    public Object intercept(Invocation invocation) throws Throwable {
       //自定义实现
       return invocation.proceed();
    }
    public Object plugin(Object target){
        return Plugin.wrap(target,this)
    }
    public void setProperties(Properties properties){
      //传入配置项
      String size = properties.getProperty("size");
    }
}
<!-- mybatis-config.xml -->
<plugins>
    <plugin interceptor="org.mybatis.example.ExamplePlugin">
        <!-- 这里的配置项就传入setProperties方法中 -->
        <property name="size" value="100">
    </plugin>
</plugins>
```

1. 拦截器实现
   Interceptor接口供插件实现，@Intercepts注解在插件实现上，表示这是一个插件类并配置将要拦截哪些方法，@Signature定义将要拦截的方法信息,如名称/类型/形参列表，Plugin类实现了InvocationHandler接口，是动态代理的具体实现，Invocation类包装了拦截的目标实例，InterceptorChain保存所有拦截器。
2. 如何实现拦截
   创建目标实例，比如A a = new A();
   Interceptor interceptor = new LogInterceptor();//如果拦截a中的save方法
   将A b = (A)interceptor.plugin(a);这里b就是a的代理实例，在调用a中的save方法时，实际将调用interceptor的intercept方法，在该方法中一定要调用Invocation的proceed方法并将返回值返回。