疯狂 Java EE 面试题（《轻量级 Java EE 企业应用实战》附赠）

本大全每个月会定期更新，索取网址：http://www.fkit.org。

Java Web 部分

Java Web 部分的面试题，可能覆盖 Servlet、JSP、EL 表达式、JSTL 标签库、Servlet 3.1 等相关内容，这些知识基本都可通过《轻量级 Java EE 企业应用实战》一书找到详细解答。
这部分面试题大部分从网络收集、整理，也有部分题目来自疯狂软件学员面试之后的反馈。

1、Java Web 通常包含哪些常用的技术规范（组件）？

Java Web 通常包含如下技术规范：
JSP 和 Servlet：JSP 的本质其实就是 Servlet，所有 JSP 页面都需要先编译成 Servlet 后，然后才能处理用户请求、生成响应。Servlet 通常需要配置，既可在 web.xml 中使用 XML 元素配置，也可使用@WebServlet 注解配置。
Filter：Filter 的作用是对用户请求或响应进行过滤，它可以对用户请求进行预处理，也可以对响应进行后处理。使用 Filter 是典型的链式处理：Filter 对用户请求进行预处理，接着将请求交给 Servlet 进行处理并生成响应，最后 Filter 再对服务器响应进行后处理。。Servlet 通常需要配置，既可在 web.xml 中使用 XML 元素配置，也可使用@Filter 注解配置。
Listener：Listener 就是监听器。不同的 Listener 监听不同的事件，ServletContextListener 监听 Web 应用的启动和结束；HttpSessionListener 监听用户会话的开始和结束；ServletRequestListener 监听用户请求；还有 ServletContextAttributeListener、HttpSessionAttributeListener、ServletRequestAttributeListener 分别监听 application、session、request 范围的属性的改变。Listener 需要配置，既可在 web.xml 中使用 XML 元素配置，也可使用注解配置。
标签库：标签库是一种优秀的表现层技术。通过标签库，可以在简单的标签中封装复杂的数据展示逻辑。标签库通常需要先编写标签处理类，然后在*.tld 文件中配置标签。JSTL 就是 Java Web 官方提供的标准标签库。
EL 表达式语言：EL 表达式语言并不是真正的编程语言，它只是一种数据访问语言，通常用于访问 Java Web 应用各范围内的数据。

2、HTTP 请求的 GET 和 POST 的区别？

GET 方式的请求：直接在浏览器地址栏输入访问地址所发送的请求或提交表单发送请求时，该表单对应的 form 元素没有设置 method 属性，或设置 method 属性为 get，这几种请求都是 GET 方式的请求。
POST 方式的请求：这种方式通常使用提交表单（由 form HTML 元素表示）的方式来发送，且需要设置 form 元素的 method 属性为 post。
1. GET 方式的请求会将请求参数的名和值转换成字符串，并附加在原 URL 之后，如?username=fkjava&password=123，因此可以在地址栏中看到请求参数名和值。POST 请求使用请求正文提交请求数据，因此用户不可见。
2. 且 GET 请求传送的数据量较小，一般不能大于 2KB。POST 方式传送的数据量较大，通常认为 POST 请求参数的大小不受限制，只取决于服务器的限制。
3. Java Web 使用 request 对象获取请求信息，我们可以使用 request.setCharacterEncoding 方法修改请求的字符编码信息，但是不能修改地址栏字符编码。GET 请求从地址栏传递信息，因此不能使用 request.setCharacterEncoding 方法去修改字符编码。而 POST 从请求正文以 form 表单形式提交，所以可以使用 request.setCharacterEncoding 这个方法去修改字符编码。
4. 总结: 能够使用 post 提交尽量使用 post 提交。

3、常见的 Web 服务器有哪些？应用服务器有哪些？它们有什么区别？

常见的 Web 服务器有 Tomcat、Jetty、Resin 等。
常见的应用服务器有 WebLogic、WebSphere、JBoss、Glassfish 等。
Web 服务器通常只支持 Java Web 规范，比如只能支持 JSP、Servlet、Listener、Filter 等技术，通常也能支持 JNDI、数据源等。因此 Web 服务器支持的功能较少。
应用服务器则需要完整地支持 Java EE 规范，不仅需要支持 Java Web 的全部功能，还需要支持 JTA、EJB、JMS、JPA 等各种 Java EE 规范，因此应用服务器的功能比较强大。
但 WebLogic、WebSphere 的价格都很昂贵，一般中小型企业都用不起这么贵的产品，而且目前企业比较流行使用轻量级 Java EE 开发架构，这种轻量级 Java EE 通常会使用 Spring、Hibernate、MyBatis 等各种开源框架，这些开源框架并不需要应用服务器的支持，因此一般企业应用现在都不需要使用应用服务器。

4、在 Tomcat 上部署 Web 应用的方式有几种？

在 Tomcat 上部署 Web 应用很简单，实际上在所有 Web 服务器上部署 Web 应用都很简单。
在 Tomcat 中部署 Web 应用的方式通常有如下几种。

- 利用 Tomcat 的自动部署。
- 利用控制台部署。
- 增加自定义的 Web 部署文件。
- 修改 server.xml 文件部署 Web 应用。

如果在开发阶段向 Tomcat 部署 Web 应用则更加简单，比如使用 Eclipse EE 版本，只要选中该 Web 应用，通过右键菜单的“Run on Server”即可把该应用部署到指定的 Web 服务器上。

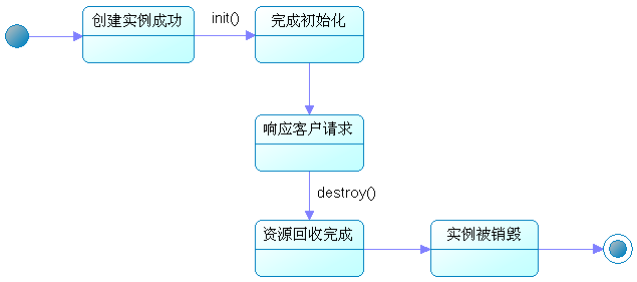
5、什么是 Servlet？

Servlet 是 Sun 公司（现已被 Oracle 收购）提供的一门用于开发动态 Web 站点的技术，是 Java 语言中编写 Web 服务器扩展功能的重要技术，同时它也是 JSP 技术的底层运行基础。
Servlet 是平台独立的 Java 类，编写一个 Servlet，实际上就是按照 Servlet 规范编写一个 Java 类。Servlet 被编译为平台中立的字节码，可以被动态的加载到支持 Java 技术的 Web 服务器中运行。
Servlet 是一个基于 Java 技术的 Web 组件，运行在服务器端，由 Servlet 容器管理，用于生成动态内容。

6、请说一说 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 方法代表。

Servlet 被 Web 服务器实例化后，容器运行其 init 方法，该方法在整个生命周期中只运行一次，用于做一些准备工作，当该方法结束时该 Servlet 可以处理客户端请求。
客户请求到达时运行其 service 方法，service 方法自动派遣运行与请求对应的 doXxx 方法（doGet、doPost）等，该方法在整个生命周期中运行多次，每个请求都使用 service 方法进行处理。
当服务器决定将实例销毁的时候调用其 destroy 方法，该方法在整个生命周期中只运行一次
，用于做一些清除工作。
其生命周期如下图所示。



7、forward 和 redirect 的区别？

1. forward 服务器内部跳转（在当前 webapp 中跳转），地址栏不显示跳转后的 URL；sendRedirect 是强制浏览器重新发请求，地址栏显示的是跳转后的 URL。
2. 执行 forward 后依然是上一次请求，执行 redirect 后会生成第二次请求。因此 forward 的目标页面可以访问原请求的请求参数，因为依然是同一次请求，所有原请求的请求参数、request 范围的属性全部存在；redirect 的目标页面不能访问原请求的请求参数，因为是第二次请求了，所有原请求的请求参数、request 范围的属性全部丢失。
3. forward 时前面是什么请求，跳转之后还是什么请求，而 sendRedirect 则一定是 get 请求。
4. 如果希望访问其他 webapp 的请求。由于 forward 是服务器内部跳转，只在当前 webapp 跳转，所以是完成不了的，因此只能用 sendRedirect 来重定向。

8、ServletConfig 对象和 ServletContext 对象有什么区别？

一个 Servlet 对应有一个 ServletConfig 对象，可以用来读取该 Servlet 的配置参数。
一个 Web App 对应一个 ServletContext 对象，因此 ServletContext 代表的整个 Web 应用，比如需要读取整个 Web 应用的配置参数，访问 application 范围的属性，都使用 ServletContext。
实际上，application 内置对象就是 ServletContext 的实例。
ServletContext 对象可以通过 context.getResourceAsStream("/images/tomcat.gif"); 或者 context.getRealPath("/")去获取 webapp 的资源文件。
ServletContext 对象的 setAttribute(String name,Object o)方法可以将对象存储在 Context 作用范围域又称为全局作用范围域，在整个 web 应用当中可以共享。
ServletContext 对象可以和服务器进行通讯，比如写信息到服务器的日志信息当中。

9、Servlet 的会话机制？

因为 HTTP 协议是无状态协议，又称为一次性连接，所以 webapp 必须有一种机制能够记住用户的一系列操作，并且唯一标示一个用户。
Cookie：就是使用一个短文本保存用户信息，在页面加载完毕是通过响应写回客户端进行保存。
Session：在服务器保存数据，Session 就是单个客户的一块内存，用以存储数据。
Tomcat 的 Session 是 HashMap 的实现。
当 request.getSession()时 Session 对象被服务器创建，并且产生一个唯一的 SessionID。
SessionID 会通过 Cookie 写回客户端进行保存。如果 Cookie 被客户端禁用，只能使用 URL 重写机制。
Session 对象可以使用 setAttribute(String name,Object o)将对象存储在 Session 作用范围域中。Session 作用范围域是在浏览器打开到浏览器关闭。

10、Filter 是什么？有什么作用？

Filter 是过滤器，它既可以对用户请求进行预处理，也也可以对响应进行后处理。Struts 2 的核心控制器就是使用一个 Filter 实现的。Filter 可负责拦截多个请求或响应；一个请求或响应也可被多个 Filter 拦截。
常用的 Filter 有如下：

- 用户授权的 Filter：Filter 负责检查用户请求，根据请求过滤用户非法请求。
- 日志 Filter：详细记录某些特殊的用户请求。
- 负责解码的 Filter：包括对非标准编码的请求解码。

11、Listener 是什么？有什么作用？

Listener 是指 Servlet 中的监听器。
Listener 可以对 ServletContext 对象、HttpSession 对象、ServletRequest 对象进行监听。既可监听 ServletContext 对象、HttpSession 对象、ServletRequest 对象的开始和结束，也可监听它们对应范围的属性的改变。常用的 Listener 有如下几个：
ServletContextListener：监听 Web 应用的启动和结束。

HttpSessionListener：监听用户 Session 的开始和结束。
ServletRequestListener：监听用户请求。
ServletContextAttributeListener：监听 application 范围的属性的改变。
HttpSessionAttributeListener：监听 session 范围的属性的改变。
ServletRequestAttributeListener：监听 request 范围的属性的改变。

12、Servlet 3.0 有哪些新特性？

Servlet 3.0 相对于 Servlet 2.0 来说最大的改变是引入了注解来取代 XML 配置，用于简化 Web 应用的开发和部署。最主要几项特性：
1. 新增的注解支持：该版本新增了若干注解，用于简化 Servlet、过滤器（Filter）和监听器（Listener）的配置，这使得 web.xml 文件从该版本开始不再是必选的了。
2. 异步处理支持：有了该特性，Servlet 线程不再需要一直阻塞，直到业务处理完毕才能再输出响应，最后才结束该 Servlet 线程。在接收到请求之后，Servlet 线程可以将耗时的操作委派给另一个线程来完成，自己在不生成响应的情况下返回至容器。针对业务处理较耗时的情况，这将大大减少服务器资源的占用，并且提高 并发处理速度。
3. 可插性支持：熟悉 Struts 2 的开发者一定会对其通过插件的方式与包括 Spring 在内的各种常用框架的整合特性记忆犹新。将相应的插件封装成 JAR 包并放在类路径下，Struts 2 运行时便能自动加载这些插件。现在 Servlet 3.0 提供了类似的特性，开发者可以通过插件的方式很方便的扩充已有 Web 应用的功能，而不需要修改原有的应用。
4. 改进的文件上传 API，以前 Web 应用要上传文件通常需要借助于 common-fileupload 等工具，但 Servlet 3.0 的 HttpServletRequest 可以直接处理文件上传，开发者只要使用@MultipartConfig 修饰该 Servlet，接下来就可在该 Servlet 中使用 HttpServletRequest 的 getPart 方法获取文件上传域。

13、JSP 是什么？有什么特点？

JSP 全称是 Java Server Pages，它和 Servlet 技术一样，都是 Sun 公司（已被 Oracle 收购）定义的一种用于开发动态 Web 站点的技术。JSP 通过在 HTML 页面中直接嵌入 Java 脚本，从而使得页面可以显示动态数据。
相比 Servlet 在 Java 代码中嵌入 HTML 标签，JSP 在页面显示上具有更方便的优势。
JSP 的本质还是 Servlet，JSP 会由 Web 服务器编译成 Servlet，因此 JSP 实际上还是以 Servlet 的形式在运行。
在实际开发中，通常会使用 MVC 模式开发，其中 Servlet 充当控制器，Servlet 负责处理请求、并调用 Service 等组件处理请求，并通过 Service 等组件获取数据，而 JSP 只是充当视图，负责数据的展现工作。

14、JSP 页面包含哪些语法？它们具有什么特征？

JSP 页面可包含如下 4 种语法：
1. JSP 注释：它的语法格式为：<%-- 注释 --%>，JSP 注释与 HTML 注释的区别在于：JSP 注释是服务端注释，当 Web 服务器处理编译 JSP 页面时，它就会忽略这些 JSP 注释，JSP 编译生成的 Servlet 并不包含 JSP 注释，因此 JSP 注释不会输出到浏览器。而 HTML 注释则会输出到浏览器。

2. JSP 声明：它的语法是：<%! 声明 %>。JSP 声明对应于 Servlet 的成员变量部分，因此 JSP 声明语法内可定义成员变量、方法、内部类，而且这些成员变量、方法也可使用 private、protected、public、static 等修饰符修饰。由于 JSP 声明的内容属于该 Servlet 的成员，因此它们可以在所有脚本中使用。
3. JSP 脚本（也可叫 Java 脚本）：它的语法是：<% 声明 %>。JSP 脚本对应于 Servlet 的 service 方法中执行代码，因此 JSP 脚本中只能定义局部变量和放置可执行性语句，JSP 脚本中定义的局部变量不能使用 private、protected、public、static 等修饰符修饰，唯一可使用的修饰符是 final。
4. 表达式（也叫输出表达式）：它的语法是：<%= 表达式%>，这种语法比较简单，只能用于在页面上输出表达式的值。JSP 表达式对应于 JSP 脚本对应于 Servlet 的 service 方法中的一条 out.writeln(表达式)语句。

15、JSP 的编译指令有哪些？

JSP 使用编译指令的语法为：
<%@ 编译指令名 属性名="属性值"…%>
JSP 编译指令有如下 3 条。
1. page：该指令是针对当前页面的指令，通常用于对该页面指定某些设置信息，比如该页面的字符集、该页面需要导入的 Java 包。一个 JSP 页面可以包含多条 page 指令。
2. include：用于指定包含另一个页面。这种包含被称为静态包含。一个 JSP 页面可以包含多条 include 指令。
3. taglib：用于导入定义标签库。一个 JSP 页面可以包含多条 taglib 指令。

16、JSP 中动态 include 与静态 include 的区别？

动态 include 用<jsp:include.../>动作指令实现，适合用于包含动态页面，并且可以带参数。动态 include 不会导入被 include 页面的编译指令，仅仅将被导入页面的 body 内容插入本页面。
静态 include 用<%@ include file=included.htm %>编译指令实现。
归纳起来，静态导入和动态导入有如下三点区别：
1. 静态导入是将被导入页面的代码完全融入，两个页面融合成一个整体 Servlet；而动态导入则在 Servlet 中使用 include 方法来引入被导入页面的内容。
2. 静态导入时被导入页面的编译指令会起作用；而动态导入时被导入页面的编译指令则失去作用，只是插入被导入页面的 body 内容。
3. 动态包含还可以增加额外的参数。

17、JSP 有哪些动作指令？分别是什么？

JSP 动作指令主要有如下 7 个。
jsp:forward：执行页面转向，将请求的处理转发到下一个页面。
jsp:param：用于传递参数，必须与其他支持参数的标签一起使用。
jsp:include：用于动态包含一个 JSP 页面。
jsp:plugin：用于下载 JavaBean 或 Applet 到客户端执行。
jsp:useBean：创建一个 JavaBean 的实例。
jsp:setProperty：设置 JavaBean 实例的属性值。

jsp:getProperty：输出 JavaBean 实例的属性值。

18、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：
9 个内置对象依次如下。
1. application：javax.servlet.ServletContext 的实例，该实例代表 JSP 所属的 Web 应用本身，可用于 JSP 页面，或者在 Servlet 之间交换信息。常用的方法有 getAttribute(String attName)、setAttribute(String attName , String attValue)和 getInitParameter(String paramName)等。
2. config：javax.servlet.ServletConfig 的实例，该实例代表该 JSP 的配置信息。常用的方法有 getInitParameter(String paramName)和 getInitParameternames()等方法。事实上，JSP 页面通常无须配置，也就不存在配置信息。因此，该对象更多地在 Servlet 中有效。
3. exception：java.lang.Throwable 的实例，该实例代表其他页面中的异常和错误。只有当页面是错误处理页面，即编译指令 page 的 isErrorPage 属性为 true 时，该对象才可以使用。常用的方法有 getMessage()和 printStackTrace()等。
4. out：javax.servlet.jsp.JspWriter 的实例，该实例代表 JSP 页面的输出流，用于输出内容，形成 HTML 页面。
5. page：代表该页面本身，通常没有太大用处。也就是 Servlet 中的 this，其类型就是生成的 Servlet 类，能用 page 的地方就可用 this。
6. pageContext：javax.servlet.jsp.PageContext 的实例，该对象代表该 JSP 页面上下文，使用该对象可以访问页面中的共享数据。常用的方法有 getServletContext()和 getServletConfig()等。
7. request：javax.servlet.http.HttpServletRequest 的实例，该对象封装了一次请求，客户端的请求参数都被封装在该对象里。这是一个常用的对象，获取客户端请求参数必须使用该对象。常用的方法有 getParameter(String paramName) 、 getParameterValues(String paramName) 、 setAttribute(String attrName,Object attrValue)、getAttribute(String attrName)和 setCharacterEncoding(String env)等。
8. response：javax.servlet.http.HttpServletResponse 的实例，代表服务器对客户端的响应。通常很少使用该对象直接响应，而是使用 out 对象，除非需要生成非字符响应。而 response 对象常用于重定向，常用的方法有 getOutputStream()、sendRedirect(java.lang.String location)等。
9. session：javax.servlet.http.HttpSession 的实例，该对象代表一次会话。当客户端浏览器与站点建立连接时，会话开始；当客户端关闭浏览器时，会话结束。常用的方法有：getAttribute(String attrName)、setAttribute(String attrName, Object attrValue)等。

19、JSP 和 Servlet 有哪些相同点和不同点？

JSP 的本质就是 Servlet，JSP 经过 Web 服务器编译之后就变成了 Servlet，因此它们的生命周期、运行机制完全是相同的。
Servlet 和 JSP 最主要的不同点在于表现形式：Servlet 是一个继承 HttpServlet 的 Java 类，而 JSP 则是在 HTML 页面中嵌入 JSP 脚本，嵌入之后组合成一个扩展名为.jsp 的页面文件。
在实际项目开发当中，JSP 侧重于视图，Servlet 主要用于控制逻辑。

20、request.getParameter()和 request.getAttribute()的区别？

getParameter()用于获取请求参数，request.getAttribute()用于获取请求的属性。具体来说，他们存在如下区别。
1. getParameter()获取的类型是 String；getAttribute()获取的类型是 Object
2. getPrameter()获取的是 POST/GET 请求的参数值；而 getAttribute()获取的是 request 范围的属性——也就是 request.setAttribute()设置的属性。而 request 并没有类似的 request.setParamter 方法。
3. setAttribute()是让 Web 服务器把这个对象放在该页面所对应的一块内存中，当页面 forward 到另一个页面时（只要还是同一个请求），另一个页面可通过 getAttribute()获取 setAttribute 方法存入的属性。

21、JSTL 是什么？优点有哪些？

JSTL（JSP Standard Tag Library，JSP 标准标签库）是 Java 官方提供的开放源代码的标砖标签库标签库，主要由四个标准的标签库（core、format、xml、sql）组成。优点有：
1. 可以简化 Java Web 的开发，JSTL 已经提供平时开发所需的各种通用标签，因此开发者基本不需要编写自定义标签。
2. 通过 JSTL 可以避免在 JSP 页面中使用 Java 脚本来进行控制，在实际企业开发中，通常都会禁止在 JSP 页面上输出 Java 脚本。

22、如何避免 JSP 页面自动生成 session 对象？为什么要这么做？

在默认情况下，当浏览器对 JSP 页面发出请求时，如果 session 还没有建立，那么 Web 服务器就会自动为该请求创建一个 session 对象，但是 session 是比较消耗资源的，如果没有必要保持和使用 session，就不应该创建 session，例如一些只是做产品展示、公司简介网页，就没必要使用 session 来保存信息。可以在 JSP 中使用 page 指令设置来避免 JSP 页面为每个请求都自动创建 session。实例代码如下：
<%@page session="false"%>

23、如何防止表单重复提交？

使用 session 跟踪可实现：
1. 在显示表单页面（比如 regist.jsp）时生成一个随机值，将其保存到 session 中，同时将其保存为表单中的隐藏域的值。
2. 在处理表单求时，获取 session 中的值和隐藏域的值，比较两者是否相同，如果相同说明不是重复提交，请求通过同时删除 session 中保存的的值；如果不相同则是重复提交，不能通过。

24、session 和 application 的区别？

session 代表一次用户会话，application 则代表 Web 应用本身。具体来说，它们存在如下区别：
1. 两者的作用范围不同：session 是用户会话级别的，application 是 web 应用级别的。因此 application 是一个全局作用域。session 则对应于一个用户的单次会话，单个用户在同一次会话内访问多个页面共享同一个 session。而整个 Web 应用的所有用户、所有会话都共享同一个 application，只要 Web 应用

没有重启，application 内的数据一直有效。
2. 生命周期不同。
session：用户首次访问网站时创建，用户注销、离开网站或者关闭浏览器消亡。
application：启动 web 服务器创建，关闭 web 服务器销毁。

25、说说自动登录功能的实现机制。

自动登录是通过 Cookie 来实现的。
1. 当用户的第一次登录时，程序需要将登录状态存入 Session 之外，还将登录信息存入 Cookie，Cookie 保存在用户的电脑上。
2. 当用户再次访问站点时，直接读取用户电脑上的 Cookie 信息，如果 Cookie 已有登录信息，则将 Cookie 中的登录信息写入 Session，这就实现了自动登录。

26、EL 表达式的功能？为什么要用 EL 表达式？

EL 的全称是 Expression Language（表达式语言），它主要功能包括：
1. 从四个域对象中读取数据数据并显示。
2. 读取请求参数数据并显示。
3. 读取 Cookie 并显示
4. 支持丰富的运算符
EL 是 JSP 2.0 引入的功能，通过 EL 可以方便地访问、显示数据。传统的 JSP 脚本、JSP 表达式用于显示数据则比较麻烦，它们往往存在如下缺点：
1. 需要在 JSP 页面中嵌入 JSP 脚本
2. 往往需要判断
3. 有时候还需要进行类型转换

27、cookie 和 session 的作用、区别、应用范围？

1. cookie 数据保存在客户的电脑上，session 数据保存在服务端。
2. cookie 不是安全的，别人可以读取、甚至修改客户端电脑上的 cookie，甚至可进行 cookie 欺骗，因此 cookie 不适合保存安全性相关的数据，安全性要求较好的数据应使用 session 保存到服务端。
3. session 会在一定时间内保持在服务器上，但是会占用内存资源，当访问的用户过多，会加重服务器的负载，考虑到减轻服务器的压力，可以将不重要的数据放在 cookie 中持久的保存。
4. 单个 cookie 保存的数据不能超过 4k，很多浏览器都限制站点最多保存 20 个 cookie。

框架部分

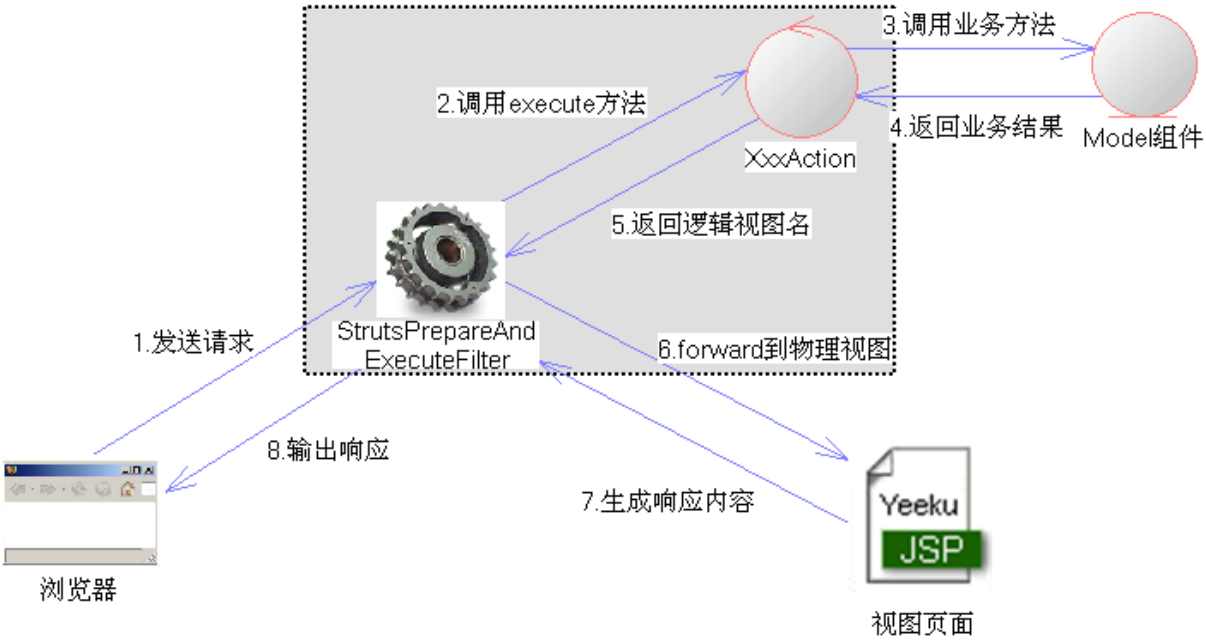
框架部分的面试题，可能覆盖 Struts、Hibernate、Spring 等相关内容，这些知识基本都可通过《轻量级 JavaEE 企业应用实战》一书找到详细解答。这部分面试题大部分从网络收集、整理，也有部分题目来自疯狂软件学员面试之后的反馈。

1、什么是 MVC 模式？

MVC (Model View Controller) 是一个设计模式，使用 MVC 应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。M 是指数据模型，V 是指用户界面，C 则是控制器。使用 MVC 的目的是将 M 和 V 的实现代码分离，从而使同一个程序可以应用于不同的表现形式。
Model：封装了所有的商业逻辑以及规则。通常被 JavaBean 或 EJB 实现。
View：使用商业逻辑处理后的结果并构建呈现给客户端的响应。通常被 JSP 实现。
Controller：管理和控制所有用户和应用程序间的交互。通常是一个 Servlet 接收用户的请求并把所有的输入转交给实际工作的 Model。最后调用 JSP 返回输出。
MVC 模式的好处：
1. 各司其职，互不干涉
在 MVC 模式中，三个层各司其职，所以如果一旦哪一层的需求发生了变化，就只需要更改相应的层中的代码而不会影响到其他层中的代码。
2. 有利于开发中的分工
在 MVC 模式中，如果按层把系统分开，那么就能更好地实现开发中的分工。网页设计人员可以进行开发视图层中的 JSP，对业务熟悉的开发人员可开发业务层，而其他开发人员可开发控制层。
3. 有利于组件的重用
分层后更有利于组件的重用。如控制层可独立成一个能用的组件，视图层也可做成通用的操作界面。
Struts 就是一个基于 MVC 模式的框架。

2、请说说 Struts 2 的工作流程。（也可以画图说明）

完整流程如下
1. 浏览器发送请求。
2. 请求被 Struts 2 的核心控制器：StrutsPrepareAndExecuteFilter 拦截。
3. StrutsPrepareAndExecuteFilter 在 struts.xml 文件中根据请求找到对应的 Action 类。
4. Struts 2 使用反射创建 Action 类的实例。
5. Struts 2 使用反射调用 Action 实例的处理方法，该方法将会返回一个 String 类型的逻辑视图名。
6. Struts 2 在 struts.xml 文件中根据逻辑视图名找到对应的视图页面，并将请求跳转到对应的视图页面。
7. Struts 2 调用视图页面向浏览器生成响应。
具体运行可参考下图



3、Struts 2 的功能扩展点有哪些？

Struts 2 最大的扩展点就是拦截器，拦截器完成了 Struts 2 框架 70%以上的工作，Struts 2 的核心控制器 StrutsPrepareAndExecuteFilter 拦截到用户请求之后，会调用各种拦截器对用户请求进行预处理，当 Action 处理完用户请求之外，各种拦截器也可对响应进行后处理。

开发者为 Struts 2 扩展拦截器也很简单，只要如下几步即可。

1. 开发自定义拦截器。
2. 配置自定义拦截器。
3. 在拦截器栈中添加自定义拦截器即可。

此外，Struts 2 还提供了以下扩展点。

1. Result 及其相关子类：通过扩展 Result，可以为 Struts 2 增加新的视图类型。
2. ObjectFactory 及其相关子类：通过扩展 ObjectFactory，可以改变 Struts 2 创建 Action 的方式。其中 Struts 2-Spring 的整合插件就是通过该扩展点来将 Action 的创建工作交给 Spring 容器完成。
3. TypeConverter 及其相关子类：通过扩展 TypeConverter，可实现自定义的类型转换器。
4. Validator 及其相关子类：通过扩展 Validator，可实现自定义的输入校验器。

4、Struts 2 拦截器和过滤器的区别？

1. 过滤器和拦截器都可对用户请求进行预处理，对服务器响应进行后处理。
2. 过滤器（Filter）是 Java Web 的规范，因此 Filter 由 Web 容器负责调用；拦截器是 Struts 2 的核心机制，因此必须由于 Struts 2 调用。
3. 过滤器依赖于 Web 容器，而拦截器不依赖于 Web 容器。
4. 拦截器是细粒度的，多个细粒度的拦截器可以组成粗粒度的拦截器栈；过滤器则通常是粗粒度的。
5. 拦截器只能对 Action 请求起作用，而过滤器则可以对几乎所有请求起作用。
6. 拦截器可以访问 Action 上下文、Stack Context 里的对象，而过滤器不能。

5、请介绍 Struts 2 的 Stack Context 和 Value Stack。

Stack Contex 是 Struts 2 的 OGNL 变量求值的重要概念。在 OGNL 表达式中主要存在如下两种变量求值的表达式。
#abc.xyz：获取 Stack Context 中 abc 对象的 xyz 属性值。
abc.xyz：获取 Stack Context 中 Root 对象的 abc 属性的 xyz 属性值。
由此可见，Stack Context 就相当于一个 Map 容器，它可以包含多个对象，而且其中还应该包含一个 root 对象。
实际上 Stack Context 中包含如下隐式对象。
parameters：用于访问请求参数
request：用于访问 request 访问内的属性
session：用于访问 session 范围内的属性
application：用于访问 application 范围内的属性
attr：用于访问各作用域内的属性，将依次搜索 page> request > session > application
action：代表当前 Action。
ValueStack：代表当前 Stack Context 中的值栈，该对象是 Stack Context 的 root 对象。
可见，ValueStack 只是 Stack Context 的 root 对象。

6、Struts 2 里面有哪些隐式对象？

Struts 2 的隐式对象，这些隐式对象都是 Map 类型
parameters：用于访问请求参数
request：用于访问 request 访问内的属性
session：用于访问 session 范围内的属性
application：用于访问 application 范围内的属性
attr：用于访问各作用域内的属性，将依次搜索 page> request > session > application
action：代表当前 Action。
ValueStack：代表当前 Stack Context 中的值栈，该对象是 Stack Context 的 root 对象。

7、Struts 2 的 Action 中获取 request 对象有几种方式？

一般来说，Struts 2 的 Action 并不推荐直接获取 request 对象，这样会导致 Action 与 Servlet API 耦合，造成 Action 不易测试的后果。
如果真的需要在 Action 中访问 Servlet API，通常有如下几种方式：
1. 使用 ActionContext 的 getXxx()方法，该方法可返回 parameters、session、application 对应的 Map，也就是这种方式并没有真正返回 Servlet API，它返回的 Map 对象，开发者可通过这些 Map 对象操作 parameters、session、application 范围的属性。
2. 通过 ServletActionContext 的静态 getXxx()获取，这些方法返回的是真正的 Servlet API。
3. 通过 XxxAware 接口注入，让 Action 类实现这些接口，Struts 2 就会把真正的 Servlet API 注入该 Action。
具体来说，如果只是访问 request 对象，可通过如下方式：
1. 使用 ActionContext 的 get、set 方法，用于操作 request 范围的属性。

2. 通过 ServletActionContext 的静态 getRequest()获取，这些方法返回的是真正的 HttpServletRequest 对象。
3. 通过 ServletRequestAware 接口诸如，让 Action 类实现 ServletRequestAware 接口，Struts 2 就会把 request 注入该 Action。

8、请说说你对 Struts 2 的拦截器的理解。

Struts 2 拦截器是在访问某个 Action 或 Action 的某个方法，字段之前或之后实施拦截，并且 Struts 2 拦截器是可插拔的，拦截器是 AOP 的一种实现.
拦截器栈（Interceptor Stack）就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，拦截器栈的拦截器就会按其之前定义的顺序被调用。
拦截器与拦截器栈又是高度统一的。也就是说，拦截器栈也可当成拦截器使用，区别只是拦截器是细粒度的，而拦截器栈则是粗粒度的。
Struts 2 的拦截器在执行 Action 的 execute 方法之前，Struts 2 会首先执行在 struts.xml 中引用的拦截器，完成一系列的功能，在执行完所有应用的拦截器的 intercept 方法后，会执行 Action 的 execute 方法。

9、在 Struts 2 中如何实现转发和重定向？

重定向分成两种：
1. 重定向到视图页面，可通过在 struts.xml 中配置 type="redirect"
2. 重定向到 Action，可通过在 struts.xml 中配置 type="redirectAction"
Struts 2 默认就是转发，因此不指定 type 属性值默认就是转发到 JSP 视图页面。
此外，如果要将请求转发到另外一个 Action，则可通过在 struts.xml 中配置 type="chain"。

10、谈谈 Struts 2 的国际化。

在 Struts 2 中是使用了拦截器来实现国际化，当然 Struts 2 的国际化依然也需要依赖于 Java 的国际化。所有 Java 相关的国际化都可按三步完成。
1. 编写国际化资源文件
国际化资源文件的命名规则为：basename_language_country.properties
2. 加载国际化资源文件
Struts 2 加载国际化资源文件有如下方式：
a. 页面范围的国际化资源文件，通过<s:i18.../>标签来加载。
b. 页面范围的国际化资源文件，让国际化资源文件的 basename 与该 Action 同名即可，并与该 Action 的源代码放在同一个路径下。
c. 包范围的国际化资源文件，让国际化资源文件的 basename 为 package，并将该国际化资源文件放在该包路径下。
d. 全局范围的国际化资源文件，可通过 Struts 2 的 struts.custom.i18n.resources 常量来加载
3. 根据 key 来访问国际化消息。在 JSP 页面中既可通过<s:text.../>标签来输出国际化消息；也可在 Struts 2 的表单标签中通过 key 输出国际化消息；在输入校验配置中也可通过 key 来访问国际化消息；在 Action 则可通过 getText()方法来获取国际化消息。

11、Action 为什么要继承 ActionSupport？

ActionSupport 实现了 Action 接口，提供了国际化，输入校验功能。
ActionSupport 还提供国际化支持：它提供了一个 getText(String key)方法实现国际化，该方法从国际化资源文件上获取国际化信息。
Action 接口提供了五个常量：SUCCES，ERROR、LOGIN，INPUT，NONE。

12、Struts 2 如何定位 action 中的方法？

主要有两种方式：
1. 在 struts.xml 文件中配置<action.../>元素时通过 method 指定该 Action 要调用的方法，如果不指定 method 属性，默认调用 execute 方法；method 属性支持形如{1}的表达式，这样该 method 属性值可动态改变。
2. 动态方法调用。在调用的 action 之后使用感叹号定位方法。
在 method 中使用{1}表达式或动态方法调用时，都需要为配置<allowed-method.../>元素，该元素指定允许动态调用的方法名，这样做是一种白名单的方式，只有在该元素中列出的方法才允许动态调用，因此可让 Struts 2 动态调用的方法更安全（避免恶意用户调用开发者不希望被调用的方法）。

13、模型驱动与属性驱动是什么？模型驱动使用时注意什么问题？

模型驱动与属性驱动都是用来封装数据的。
1. 模型驱动：让 Action 类中实现 ModelDriven<T>接口使用泛型把属性类封装起来，重写 getModel()方法，然后在实现类里创建封装所有请求参数的 Model 类，然后通过这个 Model 类封装的属性来获取请求参数。
2. 属性驱动：在 Action 类为所有请求参数定义成员变量，生成 get 与 set 方法，通过 Action 的属性来获取请求参数。
模型驱动是 WebWork 早期遗留的做法，现在已经都不推荐使用模型驱动，没有太大的意义——因此 Struts 2 的 Action 本身就支持定义复合类型的属性，这样完全可以把标量的请求参数转换为复合类型的对象。

14、Struts 2 是怎样进行值封装的？

Struts 2 的值封装实际上是采用了 OGNL 表达式.
Struts 2 的拦截器经过模型驱动时会先判断 Action 是否实现了 ModelDriven，如果是则拿到模型的实例放在了栈的顶部。
属性驱动则会从栈里面把栈顶的实例给取出来，从页面传进来的值放在一个 Map 集合当中，
通过 Map 集合进行迭代会通过 OGNL 技术把值封装到实例中。

15、Struts 2 如何进行输入校验？

A. 编程校验

1 继承 ActionSupport，重写 validate 方法(针对所有方法)(服务器端编程,不推荐)。
2 validateXxx 方法（Xxx 代表的是方法名，针对某个方法进行效验）。
3 如果有错误就把错误信息放在 FieldError 中，并且跳转到指定的错误业务类，没有就会进行 action 方法的调用。
B. 使用校验框架
每个 Action 类有一个校验文件，命名 Action 类名-validation.xml,且与 Action 类同目录，这是对 action 里面所有的方法进行校验。
对 Action 里面的指定方法做校验使用 Action 的类名-访问路径_方法名-validation.xml。
如:StudentAction-student_add-validation.xml
在效验文件里又分为两种:
字段校验：字段用什么校验器来校验。
非字段校验：是用校验器校验什么字段。
通俗点讲：字段校验：校验谁，用什么方法。
非字段校验：用什么校验，校验谁 。

16、项目中遇到什么问题？

1. 表单重复提交。
解决方法：在页面使用标签<s:token/>，使用检验表单重复提交的拦截器 tokenSession。。
2. 国际化必须经过 Action
Struts 2 的国际化是通过 i18n 拦截器来实现的，而拦截器是在访问 Action 的时候才执行。
解决方法：在对 jsp 的访问之前进行对 action 的访问。
3. 在页面使用转发会报 404 错。
原因：Struts 2 使用的是 Filter 机制，而<jsp:forward/>的转发机制是基于 Servlet 的。
解决方法：可以通过过滤器改变请求地址。
4. 当效验出错时，要跳转到相应的页面。
解决方法：使用通配符来解决。

17、Struts 2 是如何实现 MVC 的？

MVC 对应于：M：模型，V：视图，C：控制器
在 Struts 2 中分别对应什么？
M（模型）：系统中实现业务的组件（Service 组件、Manager 组件与 DAO 组件等）。
V（视图）：由各种视图技术代表，通过 result 的 type 属性来指定视图的类型，可支持 JSP、Freemarker、Velocity、XSLT、报表、图表等各种视图技术。
C（控制器）：由核心控制器与业务控制器组成，核心控制器是 Struts 2 提供的 StrutsPrepareAndExecuteFilter，而业务控制器则是我们写的各种 Action 类。
MVC 框架的主要介绍了 Java Web 开发中通用的问题：数据封装，类型转换、输入校验、国际化、异常处理、控制器与视图的数据交换的通用问题，开发者使用 MVC 框架之后，则可专注于编写业务控制器，通用问题则交给 MVC 框架解决，从而让项目开发效率更高，开发的项目更稳定。

18、为什么要继承默认的包？

Struts 2 里面默认有很多的常量，拦截器，bean 定义、result 类型，这些内容都是定义在 struts-default 默认包中的。因此继承默认包即可继承这些常量、拦截器、bean 定义。

19、开发项目时 Struts 2 在页面怎样拿值？

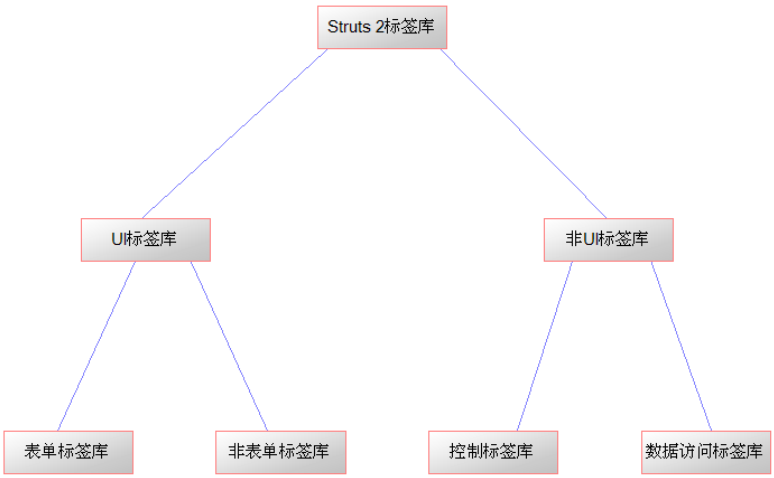
Action 与 JSP 的数据交换是通过 OGNL 的 Stack Context 实现的，Action 会将数据放在 Stack Context 中、或放入 Stack Context 的 root 对象：ValueStack 中，而 JSP 页面则可使用 OGNL 表达式从 Stack Context 或 ValueStack 中取值。

20、怎么样用 Struts 2 进行文件的上传或者下载？

Struts 2 已经提供了默认的拦截器来处理上传，因此只要经过如下几步即可
1. 在页面上将上传文件的表单的 enctype 属性设置为 multipart/form-data.
2. 在 Action 里为文件上传域定义三个字段 File file，String [file]ContentType，String [file]FileName，其中 file 代表文件上传域的 name 属性值；这三个字段分别代表上传的文件、上传文件的类型、上传文件的文件名
3. 在 Action 的处理方法（比如 execute）中通过 IO 流将文件内容写入服务器磁盘或数据库即可。
如果有多个文件上传，那么则使用 3 个 List 属性来封装多个文件上传域的文件内容、文件名、文件类型。
文件下载则使用 type 为 stream 的 result，这种 result 会生成被下载文件的二进制流作为响应。该 Action 默认的有个 InputStream 类型的 inputStream 属性，程序从硬盘上面读取下载文件对应的流，并赋值给 inputStream 属性即可。

21、简单讲讲 Struts 2 的标签，不少于 5 个。

Struts 2 的标签可分为如图



表单标签中常用的有<s:form.../>、<s:textfield.../>、<s:select.../>、<s:checkbox.../>等

非表单标签中常用的有<s:actionerror.../>、<s:actionmessage.../>、<s:fielderror.../>等
控制标签中常用的有<s:if.../>、<s:iterator.../>
数据访问标签中常用的有<s:text.../>、<s:i18n.../>、<s:set.../>、<s:bean.../>、<s:action.../>、<s:debug.../>等

22、Struts 2 常量的修改方式？

Struts 2 配置常量总共有三种方式。
1. 通过 struts.properties 文件。
struts.custom.i18n.resources=mess
2. 通过 struts.xml 配置文件，如
<constant name="struts.custom.i18n.resources" value="mess"/>
3. 通过 Web 应用的 web.xml 文件。

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher
        .filter.StrutsPrepareAndExecuteFilter</filter-class>
    <init-param>
        <param-name>struts.custom.i18n.resources</param-name>
        <param-value>mess</param-value>
    </init-param>
</filter>
```

23、Struts 2 是如何管理 Action 的？这种管理方式有什么好处？

Struts 2 框架中使用包来管理 Action，Struts 2 的包用于管理一组业务功能相关的 Action，包不仅具有逻辑上的功能，还提供了命名空间的支持。比如将包的 namespace 指定了/fkjava，而 Action 的 name 指定为/foo，那么该 Action 实际处理的 URL 为/ fkjava/foo

24、Struts 2 如何对指定的方法进行验证？

从两方面回答。
A. 使用校验文件时，Action 类名-validation.xml 文件默认会校验所有与 execute 方法签名相同的方法。因此如果要校验指定文件，则需要将校验文件的文件名定义为<ActionClassName>-<ActionAliasName>-validation.xml。
B. 在 Action 中重写方法进行校验。validate()方法默认会校验 Action 中所有与 execute 方法签名相同的方法，因此如果要校验指定方法，则需要提供 validateXxx()方法实现， validateXxx()只会校验 Action 中方法名为 xxx 的方法。validateXxx 中 Xxx 的第一个字母要大写。

25、Struts 2 默认能解决 get 和 post 提交方式的乱码问题吗？

不能。struts.i18n.encoding=UTF-8 属性值只能解析 POST 提交下的乱码问题。

26、请你写出 Struts 2 中至少 5 个的默认拦截器。

fileUpload：负责处理文件上传功能的拦截器。
i18n：记录用户选择 locale、处理国际化的拦截器。
cookies：处理 Cookie 的拦截器。
conversionError：负责处理类型转换错误的拦截器。
validation：负责处理输入校验的拦截器。
params：负责处理参数的拦截器。
checkbox：添加了 checkbox 自动处理代码，将没有选中的 checkbox 的内容设定为 false，而 html 默认情况下不提交没有选中的 checkbox。
chain：让前一个 Action 的属性可以被后一个 Action 访问，现在和 chain 类型的 result 结合使用。
token：负责处理防重刷新的拦截器。

27、result 的 type 属性中有哪几种结果类型？

Struts 2 默认的结果类型是 dispatcher，这种结果类型也就是对应于 Java Web 的转发（forward），因此这种结果类型只能把请求转发给应用程序里的另一个资源，而不能把请求转发给一个外部资源。
此外，Struts 2 还支持如下结果类型。
chain 结果类型：Action 链式处理的结果类型。
freemarker 结果类型：用于指定使用 FreeMarker 模板作为视图的结果类型。
httpheader 结果类型：用于控制特殊的 HTTP 行为的结果类型。
redirect 结果类型：用于直接跳转到其他 URL 的结果类型。
redirectAction 结果类型：用于直接跳转到其他 Action 的结果类型。
stream 结果类型：用于向浏览器返回一个 InputStream（一般用于文件下载）。
velocity 结果类型：用于指定使用 Velocity 模板作为视图的结果类型。
xslt 结果类型：用于与 XML/XSLT 整合的结果类型。
plainText 结果类型：用于显示某个页面的原始代码的结果类型。

28、什么是 ORM 思想？常用的基于 ORM 的框架有哪些？各有什么特点？

ORM 的全称是 Object/Relation Mapping，即对象/关系数据库映射。ORM 可理解成一种规范，它概述了这类框架的基本特征：完成面向对象的编程语言到关系数据库的映射。当 ORM 框架完成映射后，既可利用面向对象程序设计语言的简单易用性，又可利用关系数据库的技术优势。因此可把 ORM 框架当成应用程序和数据库的桥梁。
ORM 框架的基本映射思想就是：
1. 持久化类映射数据表
2. 持久化对象映射数据行
3. 类的 field 映射数据列
目前流行的 ORM 框架有如下这些产品。
1. JPA：JPA 本身只是一种 ORM 规范，并不是 ORM 产品。它是 Java EE 规范制定者向开源世界学习

的结果。JPA 实体与 Hibernate PO 十分相似，甚至 JPA 实体完全可作为 Hibernate PO 类使用，因此很多地方也把 Hibernate PO 称为实体。相对于其他开源 ORM 框架，JPA 的最大优势在于它是官方标准，因此具有通用性，如果应用程序面向 JPA 编程，那么应用程序就可以在各种 ORM 框架之间自由切换：Hibernate？TopLink？OpenJPA？随你喜欢就行。
2. Hibernate：目前最流行的开源 ORM 框架，已经被选作 JBoss 的持久层解决方案。整个 Hibernate 项目也一并投入了 JBoss 的怀抱，而 JBoss 又加入了 Red Hat 组织。因此，Hibernate 属于 Red Hat 组织的一部分。Hibernate 灵巧的设计、优秀的性能，还有丰富的文档，都是其风靡全球的重要因素。
3. MyBatis（早期名称是 iBATIS）：Apache 软件基金组织的子项目。与其称它是一种 ORM 框架，不如称它是一种“SQL Mapping”框架。曾经在 Java EE 开发中扮演非常重要的角色，但因为并不支持纯粹的面向对象的操作，因此现在逐渐开始被取代。但在一些公司中依然占有一席之地。特别是一些对数据访问特别灵活的地方，MyBatis 更加灵活，它允许开发人员直接编写 SQL 语句。
4. TopLink：Oracle 公司的产品，早年单独作为 ORM 框架使用时一直没有赢得广泛的市场，现在主要作为 JPA 实现。GlassFish 服务器的 JPA 实现就是 TopLink。

29、请说说你对 Hibernate 的理解。JDBC 和 Hibernate 各有什么优势和劣势？

Hibernate 是一个轻量级的持久层开源框架，它是连接 Java 应用程序和关系数据库的中间件，负责 Java 对象和关系数据之间的映射，通过使用 Hibernate，可以让开发人员以面向对象的方式操作关系数据库，从而让开发人员更专注业务开发。
JDBC 和 Hibernate 都是用于数据持久化操作的。
JDBC 只是提供了执行 SQL 语句的接口，因此它的优势也是它的劣势。
优势：开发人员可以直接编写 SQL 语句、调用存储过程，因此可以充分利用各种 SQL 的特征、对 SQL 语句进行性能优化。
劣势：由于 JDBC 不支持面向对象，因此需要在持久化操作时将对象拆成标量值，然后才能将数据存入数据库；此外 JDBC 还需要开发人员编写复杂的 SQL 语句。

30、写出 Hibernate 框架核心接口/类的名称，并描述它们各自的作用。

Hibernate 的核心接口一共有 5 个，分别为：Configuration、SessionFactory 、Session、Transaction、Query、Criteria。这 5 个核心接口在任何开发中都会用到。它们的作用为：
Configuration 接口：Configuration 接口负责加载 Hibernate 配置文件，并负责启动 Hibernate 创建 SessionFactory 对象。在 Hibernate 的启动的过程中，Configuration 类首先读取配置文件，定位持久化映射类，然后创建 SessionFactory 对象。
SessionFactory 接口：SessionFactroy 是数据库编译后的内存镜像。SessionFactory 是线程安全的，每个数据库对应一个 SessionFactory。SessionFactory 持有一个可选的二级缓存。一般来说，一个项目通常只需要一个 SessionFactory 就够，当需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory。
Session 接口：Session 接口负责对被持久化对象执行 CRUD 操作，Session 持有必选的一级缓存，Session 是线程不安全的。

Transaction 接口：Transaction 接口负责事务相关的操作。它是可选的，开发人员也可以设计编写自己的底层事务处理代码。
Query 接口：代表 HQL 查询的接口。可通过 Session 的 createQuery 方法、传入 HQL 语句来创建 Query 对象，然后通过 Query 的 getResutList()方法返回查询结果集。
Criteria 接口：代表条件查询的接口。需要指出的是，从 Hibernate 5 开始，Hibernate 的整个条件查询体系已被标记为过时，Hibernate 的条件查询全面改为使用 JPA 条件查询。JPA 动态条件查询则由 CriteriaQuery 代表。
NativeQuery：代表原生 SQL 查询的接口。

31、Hibernate 的 session.load()和 session.get()有什么分别？

get()方法会立即发送查询语句提取数据，而 load()方法当对象使用时才去数据库查询。
因此 load()方法和 get()方法的主要区别在于是否延迟加载，使用 load()方法将具有延迟加载功能，load()方法不会立即访问数据库，当试图加载的记录不存在时，load()方法可能返回一个未初始化的代理对象；而 get()方法总是立即访问数据库，当试图加载的记录不存在时，get()方法将直接返回 null。
当试图加载的记录不存在时，由于 load()方法返回的只是未初始化的代理对象，因此使用该代理对象时会引发 ObjectNotFoundException 异常。

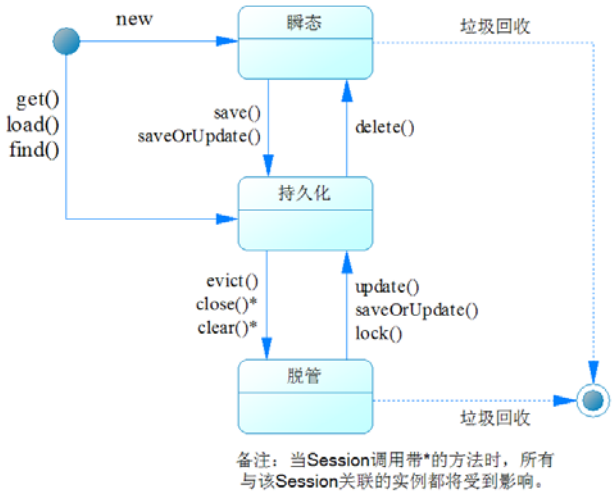
32、Session 的 save()和 persist()有什么分别？

Hibernate 之所以提供与 save()功能几乎完全类似的 persist()方法，一方面是为了照顾 JPA 的用法习惯。此外，save 和 persist 还存在如下区别：
使用 save()方法时，Hibernate 会立即将持久化对象对应的数据插入数据库，并返回该持久化对象的标识属性值（即对应记录的主键值）；
使用 persist()方法时，Hibernate 则不会立即转换成 insert 语句，因此该方法没有任何返回值。当程序需要封装一个长会话流程的时候，persist()方法很有用。

33、Hibernate 实体的三种状态是什么？各有什么特点？

Hibernate 持久化对象支持如下几种对象状态。
1. 瞬态：对象由 new 操作符创建，且尚未与 Hibernate Session 关联的对象被认为处于瞬态。瞬态对象不会被持久化到数据库中，也不会被赋予持久化标识。如果程序中失去了瞬态对象的引用，瞬态对象将被垃圾回收机制销毁。使用 Hibernate Session 可以将其变为持久化状态。
2. 持久化：持久化实例在数据库中有对应的记录，并拥有一个持久化标识（identifier）。持久化的实例可以是刚刚保存的，也可以是刚刚被加载的。无论哪一种，持久化对象都必须与指定的 Hibernate Session 关联。Hibernate 会检测到处于持久化状态对象的改动，在当前操作执行完成时将对象数据写回数据库。开发者不需要手动执行 update。
3. 脱管：某个实例曾经处于持久化状态，但随着与之关联的 Session 被关闭，该对象就变成脱管状态。脱管对象的引用依然有效，对象可继续被修改。如果重新让脱管对象与某个 Session 关联，这个脱管对象会重新转换为持久化状态，而脱管期间的改动不会丢失，也可被写入数据库。正是因为这个功能，逻辑上的长事务成为可能，它被称为应用程序事务，即事务可以跨越用户的思考，因为当对象处于脱管状态时，对该对象的操作无须锁定数据库，不会造成性能的下降。

下图显示了 Hibernate 持久化对象的状态演化图。



34、如何对 Hibernate 进行性能优化？

大体上，对于 HIBERNATE 性能调优的主要考虑点如下:

1. 数据库设计调整
2. HQL 优化
3. API 的正确使用（如根据不同的业务类型选用不同的集合及查询 API）
4. 通过调整配置参数（日志，二级缓存，查询缓存，fetch_size，batch_size 等）
5. 映射优化（调整主键生成策略，延迟加载，关联优化）
6. 一级缓存的管理
7. 针对二级缓存，还有许多特有的策略
8. 事务控制策略。

1、数据库设计调整方面可能涉及如下方面。

1. 降低关联的复杂性
2. 尽量不使用联合主键
3. ID 的生成机制，不同的数据库所提供的机制并不完全一样
4. 适当的冗余数据，不过分追求高范式

2、HQL 优化

HQL 底层同样是转换成 SQL 语句执行查询的，因此可以在调试时将 Hibernate 的 show_sql 选项打开，这样能看到 HQL 语句所对应的 SQL 语句，然后再针对 SQL 语句执行优化。

3、API 的正确使用

比如获取查询结果集时，调用 getResultList()方法会返回多条记录组成的 List 集合，当一次抓取数据量太大时，该方法可能导致内存溢出的问题，而如果用基于（ScrollableResults）或 iterator()方法来获取结果集，就可避免内存溢出的问题（但 iterator()方法来处理少量数据时性能较差）

4、调整配置参数

1. 建议开启二级缓存，全局的二级缓存可有效地提升性能。
2. 谨慎地开启查询缓存，查询缓存针对 HQL 语句进行缓存：即完全一样的 HQL 语句再次执行时可以利用缓存数据。但是，查询缓存在数据变更频繁，查询条件相同的概率并不大的系统中可能会起反作用：它会耗费大量内存空间，却难以复用缓存的数据。
3. fetch_size：用于控制 JDBC 每次抓取记录的数量，该参数并不是越大越好，而应根据业务条件动态调整。

4. batch_size：用于控制 JDBC 批量操作的数量。该参数同样需要根据业务条件动态调整。
5. 生产系统中，要关掉 show_sql 选项。

5、映射优化

关联映射时候使用双向一对多关联，不使用单向一对多
尽量避免使用一对一，用多对一取代
继承映射时使用显式多态

35、比较 HQL、Criteria、Native-SQL 这三者做查询的区别，以及应该如何进行选择。

HQL 查询、Criteria（Hibernate 5 中被 JPA 的动态条件查询取代）和 Native-SQL 查询所使用的查询 API 存在区别，但最终底层都是转换为 SQL 执行。
HQL 查询是一种简单、方便的面向对象的查询方式，Hibernate 查询基本上主要都使用这种查询方式。但 HQL 查询的缺点是构建动态条件查询很不方便。
Hibernate 原来的 Criteria 适合构建动态查询，但不太适合统计查询。但 Hibernate 5 改为使用 JPA 的动态条件查询之后，CriteriaQuery 的功能被大大加强了（可参考《轻量级 Java EE 企业应用实战》一书的讲解），现在主要的弱点是在定义查询条件时不如 HQL 灵活。
Native-SQL 使用原生 SQL 查询，因此可以让开发人员直接编写原生 SQL 语句，因此可以充分利用原生 SQL 语句的各种有点。但这种方式首先是编程比较复杂，其实原生 SQL 可能会利用特定数据库功能，因此可移植性并不好。
在实际项目开发中，建议以 HQL 查询为主；如果查询中涉及大量动态查询条件，此时可考虑使用 CriteriaQuery（取代了 Criteria）来动态构建查询条件；如果项目是从早期 SQL 项目移植到 Hibernate 的，或需要使用某些数据库的特性的时候，建议选择 Native-SQL 了。
另外需要说明的是，如果开发人员开始时真的需要大量自己编写 SQL 语句、直接利用 SQL 语句进行优化，则建议采用 MyBatis 框架来代替 Hibernate

36、Hibernate 的一级缓存和二级缓存有什么区别？

一级缓存由 Session 实例维护，它是必选的，其中保持了 Session 当前所有关联实体的数据，也称为内部缓存。
二级缓存则存在于 SessionFactory 层次，它是可选的全局缓存，所有 Session 都会共享 SessionFactory 的二级缓存。
一级缓存只能缓存当前会话的实体，无法给整个应用共享；二级缓存可为整个应用提供缓存支持。二级缓存可以开启查询缓存，而一级缓存不行。

37、Hibernate 中的命名 SQL 查询指的是什么？

命令 SQL 查询指的是将 SQL 查询语句放在映射文件或注解中配置，这样可避免将 SQL 字符串硬编码地写在 Java 代码中。
在映射文件中通过<sql-query>标签在映射文档中定义命名 SQL 查询，也可使用@NamedNativeQuery 注解映射命名 SQL 查询，使@NamedNativeQuery 定义命名 SQL 查询时，还需要使用

@SqlResultSetMapping 定义结果映射。
命名 SQL 定义完成之后，接下来即可通过使用 Session.getNamedQuery()方法对它进行调用。
因此：命名 SQL 查询与原生 SQL 查询的本质是一样的，区别只是定义 SQL 语句的位置有区别而已。

38、Hibernate 中的 SessionFactory 有什么作用？SessionFactory 是线程安全的吗？

SessionFactory 是整个数据库映射关系的内存镜像，也是创建 Session 对象的工厂，SessionFactory 底层通常封装了 DataSource。SessionFactory 通常是在应用启动时创建完成，应用代码用它来获得 Session 对象。SessionFactory 是线程安全的，所以多个线程可以共享同一个 SessionFactory。
一般来说，一个数据库对应一个 SessionFactory，因此普通的 Java EE 应用一般只有一个 SessionFactory，但如果该 Java EE 需要使用多个数据库，则需要配置多个 SessionFactory。另外，SessionFactory 的内部状态包含着同对象关系映射的所有元数据，因此它是不可变的，一旦创建好后就不能对其进行修改了。

39、Hibernate 中的 Session 指的是什么？Session 能否在多个线程间进行共享？

Session 代表 Hibernate 执行持久化操作的关键 API，它底层负责维护与数据库的连接，Session 是线程不安全的，也就是说，Hibernate 的 Session 不能在多个线程间进行共享。虽然 Session 会尽量延迟去获取数据库连接，但是 Session 最好还是在用完之后立即将其关闭。

40、Hibernate 中 sorted collection 和 ordered collection 有什么不同？

sorted collection 是通过使用 Java 的 Comparator 在内存中进行排序的，ordered collection 中的排序用的是数据库的 order by 子句。对于比较大的数据集，为了避免在内存中对它们进行排序而出现 Java 中的 OutOfMemoryError，最好使用 ordered collection。

41、Hibernate 中 transient、persistent、detached 对象三者之间有什么区别？

持久化对象有以下三种状态：
1. transient（瞬态）：对象由 new 操作符创建，且尚未与 Hibernate Session 关联的对象被认为处于瞬态。瞬态对象不会被持久化到数据库中，也不会被赋予持久化标识。如果程序中失去了瞬态对象的引用，瞬态对象将被垃圾回收机制销毁。使用 Hibernate Session 可以将其变为持久化状态。
2. persistent（持久化）：持久化实例在数据库中有对应的记录，并拥有一个持久化标识（identifier）。持久化的实例可以是刚刚保存的，也可以是刚刚被加载的。无论哪一种，持久化对象都必须与指定的

Hibernate Session 关联。Hibernate 会检测到处于持久化状态对象的改动，在当前操作执行完成时将对象数据写回数据库。开发者不需要手动执行 update。
3. detached（脱管）：某个实例曾经处于持久化状态，但随着与之关联的 Session 被关闭，该对象就变成脱管状态。脱管对象的引用依然有效，对象可继续被修改。如果重新让脱管对象与某个 Session 关联，这个脱管对象会重新转换为持久化状态，而脱管期间的改动不会丢失，也可被写入数据库。
需要说明的是，此处的 transient 可不是 Java 中的 transient 关键字，二者风马牛不相及。

42、Hibernate 中 Session 的 lock()方法有什么作用？

Session 的 lock()方法重建了实体与 Session 的关联关系却并没有将实体的状态同步、更新到底层数据库。因此，使用 lock()方法时一定要多加小心。当程序调用 Session 的 lock()方法来锁定某个实体时，Hibernate 只是从数据库读取该实体对应的数据库记录（使用 LockMode.NONE 锁时不读取记录），并对该记录执行对应的锁定。
此外，Session 的 update()方法也可用于重建脱管对象与 Session 的关联关系，但 update()方法会将实体的状态同步、更新到底层数据库。

43、Hibernate 中二级缓存指的是什么？

二级缓存则存在于 SessionFactory 层次，它是可选的全局缓存，所有 Session 都会共享 SessionFactory 的二级缓存。二级缓存可为整个应用提供缓存支持。二级缓存可以开启查询缓存，而一级缓存不行。

44、Hibernate 中的查询缓存指的是什么？

查询缓存就是根据查询的 HQL 语句来缓存它的查询结果：即完全一样的 HQL 语句再次执行时可以利用缓存数据，从而避免重复查询数据。
但由于查询缓存只是缓存查询实体的 ID，因此查询缓存必须与二级缓存结合使用。否则，即使程序拿到了缓存实体的 ID，程序依然需要根据 ID 去数据库中抓取相应的记录。
另外在数据变更频繁、查询条件经常变化的系统中，查询缓存被复用的机会并不大，因此查询缓存往往可能会起反作用：它会耗费大量内存空间，却难以复用缓存的数据。

45、Hibernate 的实体类为什么要提供一个无参数的构造器？

Hibernate 实体类必须包含一个无参数的构造器, 这是因为 Hibernate 框架要使用反射 API，会通过反射的 Class.newInstance()来创建这些实体类的实例。如果在实体类中找不到无参数的构造器，这个方法就会抛出一个 InstantiationException 异常。

46、可不可以将 Hibernate 的实体类定义为 final 类？

可以将 Hibernate 的实体类定义为 final 类，但这种做法并不好。
因为 Hibernate 的延迟加载都是基于代理模式来实现的，Hibernate 使用 Javassist 工具来为实体类生成代理，这些代理类是实体类的子类。

如果你把实体类定义成 final 类，由于 Java 不允许对 final 类派生子类，所以 Hibernate 就无法为实体类创建代理，这样就没办法使用延迟加载了，Hibernate 就没办法利用延迟加载来提升性能了。

47、介绍使用 Hibernate 操作数据库的基本步骤。

Hibernate 持久化操作的核心 API 是 Configuration、SessionFactory、Session、Transaction、Query、CriteriaQuery 等。使用 Hibernate 执行持久化操作的步骤如下。

1. 创建 Configuration 对象，它负责读取并解析配置文件
2. 调用 Configuration 创建 SessionFactory，该方法会负责读取并解析映射信息
3. 调用 SessionFactory 打开 Sesssion
4. 开启 Transation
5. 持久化操作，CRUD 操作分别对应 save/persist，load/get，update，delete 方法；如果要执行 HQL 查询则需要创建 Query 对象；如果要执行动态条件查询，则需要创建 CriteriaQuery 对象。
6. 提交事务
7. 关闭 Session
8. 关闭 SesstionFactory

48、什么是延迟加载？

延迟加载指的是只有当真正需要数据的时候，才去底层数据库加载数据记录。Hibernate 提供了对实体对象的延迟加载以及对集合属性、关联实体的延迟加载。

49、Hibernate 是如何实现延迟加载的？

延迟加载指的是当 Hibernate 加载实体、集合属性或关联实体时，Hibernate 并未真正去数据库中取出数据来创建实体，只是创建了这些实体的代理对象，这些代理对象并持有真正的数据。
只有当程序实际访问实体、集合属性或关联实体时，Hibernate 才会生成 SQL 语句去底层数据库获取数据记录、创建真正的实体对象。通过使用延迟加载，可以让数据延迟到真正需要时才去获取，这样可以降低服务器的内存开销，从而提高应用的性能。

50、Hibernate 中怎样实现类之间的关系？（如：一对多、多对多的关系）

实体类之间的关联关系在底层数据库中体现为表与表之间的关联关系。Hibernate 使用@OneToMany、@ManyToOne、@OneToOne、@ManyToMany 来映射实体之间的关联关系。这些注解通常可指定如下属性：

1. targetEntity：该属性指定关联实体的类型。
2. fetch：该属性指定是否需要延迟加载关联实体。通常来说，如果关联实体是 Many（比如 OneToMany、ManyToMany），这意味着关联实体有个 N 个，因此该属性默认是 FetchType.LAZY；如果关联实体是 One（比如@OneToOne、OneToOne），这意味着关联实体只有 1 个，因此该属性默认是 FetchType.EAGER。

3. cascade：该属性用于指定该 Hibernate 对该实体的持久化操作是否级联到关联实体。
4. mappedBy：该属性指定当前实体不控制关联关系，而是由关联实体来控制关联关系。
这些注解可能在需要与如下注解结合使用。
@JoinColumn：映射外键列。
@JoinTable：映射连接表。

51、介绍 Hibernate 的缓存机制。

Hibernate 一共支持 3 种缓存：
1. Session 级别的、必选的一级缓存。
2. SessionFactory 级别的、全局的、可选的二级缓存。
3. SessionFactory 级别的、全局的、可选的查询缓存。
Session 级别的一级缓存只能被当前会话使用，当 Session 对缓存中的实体执行 CRUD 操作时，Hibernate 并不会立即将这种操作写入底层数据库，它会缓存程序对实体的修改，只有当程序调用 Session 的 flush()或 close()方法时才会将实体的状态写入底层数据库；此外，当程序通过 Session 获取实体时，如果 Session 的一级缓存中已有该实体，程序会直接使用缓存中的实体，不会去数据库中抓取记录。
SessionFactory 级别的二级缓存是全局的，可以被所有 Session 共享，这个级别的缓存必须由开发人员配置打开。当程序使用 Session 加载实体时，Hibernate 的加载顺序是：一级缓存 -> 二级缓存 -> 数据库。
查询缓存的根据 HQL 语句进行缓存，如果查询所使用的 HQL 语句、参数与缓存中 HQL 语句、参数完全相同时，Hibernate 不会重新查询，而是直接使用缓存中的查询结果。查询结果只缓存查询实体的 ID，因此查询缓存必须与二级缓存结合使用，否则性能反而会下降。

52、inverse 的好处？

inverse 用于在映射文件中设置关联关系由哪一端控制，该属性与关联注解中的 mappedBy 属性的作用大致相同。
对于 OneToMany 关联，应该避免由于 1 的一端的控制关联关系，比如系统中有 Customer 和 Order 存在 OneToMany 的关系，如果由 Customer 来控制关联关系，假设程序需要添加一个 Order，则程序需要先取 Customer 实体， 然后调用 getOrders()得到所有的 Order 实体集合，然后往集合里面多加入新建的 Order 实体，然后才 save(Customer 实体)，这样开销太大，而且底层会生成额外的 update 语句，因此性能较差；如果改为使用 Order 来控制关联关系，此时要添加一个 Order，则只要 new 一个 Order 对象，然后通过 Order 设置与 Customer 的关联，然后 save(Order 实体)即可，因此开销较小，而且底层性能也好。
而 inverse 和 mappedBy 属性则可设置让 OneToMany 关联中 1 的一端不再控制关联关系。

53、merge 的含义？

merge()方法与 update()方法功能比较相似。
merge()方法也可将程序对脱管对象所做的修改保存到数据库，但 merge()与 update()方法的区别在于：merge()方法不会持久化给定的对象。举例来说，当程序执行 session.update(a)代码后，a 对象将会变成持久化状态；而执行 session.merge(a)代码后，a 对象依然不是持久化状态，a 对象依然不会被关联到

Session 上，merge()方法会返回 a 对象的副本——该副本处于持久化状态。
当程序使用 merge()方法来保存程序对脱管对象所做的修改时，如果 Session 中存在相同持久化标识的持久化对象，merge()方法里提供的对象状态将覆盖原有持久化实例的状态。如果 Session 中没有相应的持久化实例，则尝试从数据库中加载，或者创建新的持久化实例，最后返回该持久化实例。

54、介绍 cascade 属性的作用。

Hibernate 的关联映射的 XML 元素、关联注解都可指定 cascade 属性，该属性用于控制程序对主实体的操作是否级联到关联实体。
需要说明的是：如果实体的属性、集合属性持有的对象只是普通组件，不是关联实体，那么程序对主实体的持久化操作总会级联到关联组件——因为这些组件不是实体，Hibernate 没法单独操作它们。
cascade 属性支持如下级联行为：
1. ALL：程序对主实体的操作，都会级联到关联实体。但解除父子关系时不会自动删除子对象（即没有 orphanRemoval 行为）。
2. DETACH：程序对主实体的脱管操作，会级联到关联实体。也就是将主实体 evict Session 管理时，关联实体也会随之脱离 Session 的管理。
3. MERGE：程序对主实体的 merge 操作，会级联到关联实体。
4. PERSIST：程序对主实体的持久化操作，会级联到关联实体。也就是持久化主实体时，关联实体也会被自动持久化。
5. REFRESH：程序对主实体的刷新操作，会级联到关联实体。也就是刷新主实体时，关联实体也会被自动刷新。
6. REMOVE：程序对主实体的删除操作，会级联到关联实体。也就是删除主实体时，关联实体也会被自动删除。
上面这些级联行为可以组合使用，比如将 cascade 指定为 PERSIST 和 REMOVE 组合，这样将同时支持久化和删除两种级联行为。
此外，对于@OneToOne、@OneToMany 两个注解还支持将 orphanRemoval 指定为 true，用于启用 orphanRemoval 行为：当主实体与关联实体解除关联关系时，这些关联实体就变成了孤儿记录，Hibernate 会自动删除它们。

55、Session 的 commit()和 flush()的区别？

首先要说的，commit()并不是 Session 的方法，而是 Transaction 的提交事务的方法。
Transaction 的 commit()会先调用 flush()将数据刷入底层数据库，然后才提交事务事务；而 flush()只是将数据刷入底层数据库，但不一定提交事务。

56、主键生成策略有哪些？

早期 Hibernate 支持 increment、identity、sequence、uuid、hilo、assigned 和 foreign 等主键生成策略，其中 foreign 主要在一对一主键关联的情况下使用。
但 Hibernate 5 已经不再建议使用这些特殊的主键生成策略，而是建议使用如下 4 种标准的主键生成策略。
1. AUTO：让 Hibernate 根据数据库特征来选择合适的主键生成策略。

2. IDENTITY：利用数据库的自增长主键来作为主键生成策略。
3. SEQUENCE：利用数据库的 sequence 特征作为主键生成策略。这种策略往往要与@SequenceGenerator 注解结合使用。
4. TABLE：利用额外的数据表来作为主键生成策略。这种策略要结合@TableGenerator 注解使用。

57、Hibernate 中使用 Integer 做映射和使用 int 做映射之间有什么差别？

Integer 是 int 的包装类，因此 Integer 可以赋值为 null，而 int 是基本类型，不可赋值为 null。
由于数据库底层的数据列通常都支持使用 null 值（只要没有非空约束），为了让持久化类与底层数据库保持一致，建议使用 Integer 做映射。

58、SQL 和 HQL 有什么区别？

HQL（Hibernate Query Language）的语法看上去与 SQL 非常相像，但它们的本质是不同：HQL 是面向对象的查询语言，因此它操作的是持久化类、对象、属性；而 SQL 是结构化查询语句，它操作的是表、行、列。
此外，它们存在如下细节差别：
1. HQL 允许省略 select 子句，SQL 不能。
2. 由于 HQL 是面向对象的，因此它操作的类、属性、别名是区分大小写的，但 SQL 是不区分大小写的。
3. HQL 支持隐式关联，但 SQL 不支持。
4. HQL 的移植性较好（由 Hibernate 根据方言转换为特定数据库的 SQL），但 SQL 的移植性较差。

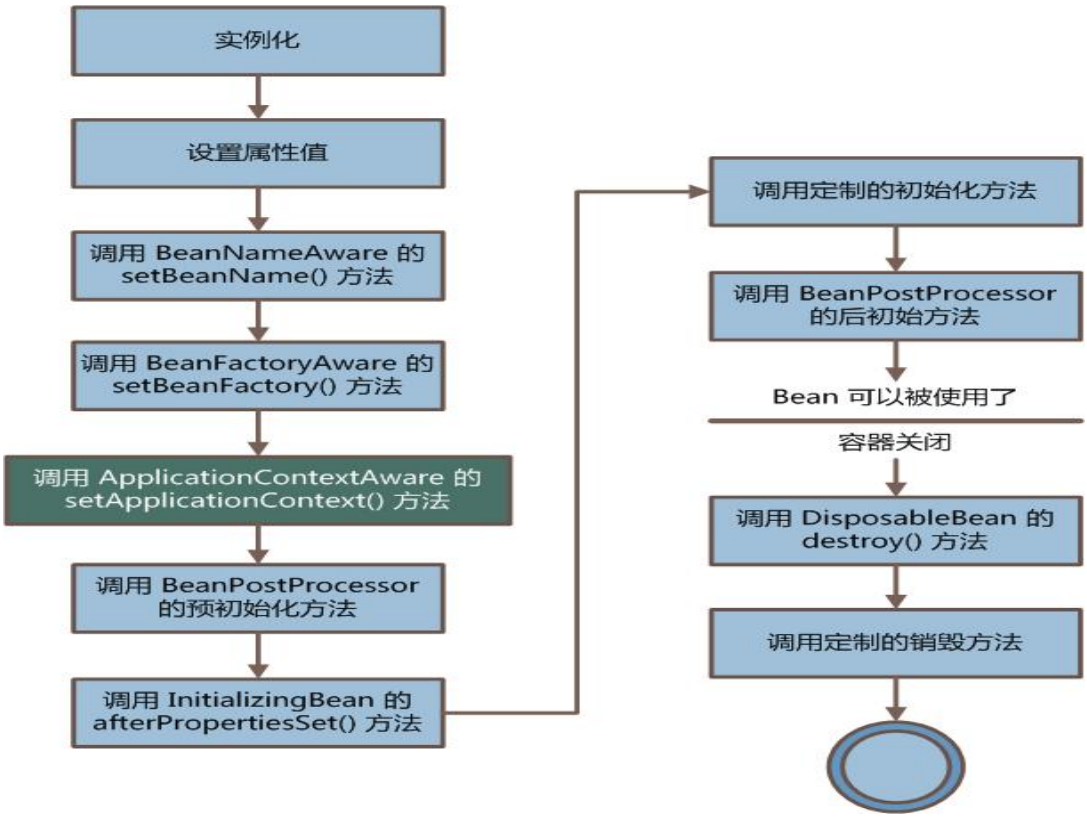
59、你是怎么看 Spring 框架的？

Spring 是一个轻量级的、非侵入性的开源框架，最重要的核心概念是 IoC 和 AOP。以此为基础，Spring 上可以与各种主流的 MVC 框架整合（Spring 本身也提供了 Spring MVC 框架），下可以与各种主流的 ORM 框架、JDBC 整合，并对持久层、提供了优秀的事务支持
通过使用 Spring，我们能够减少类之间的依赖性和程序之间的耦合度，最大程度的实现松耦合，使程序更加灵活，具有更好的可扩展性。使用 Spring，可让开发人员真正的专注于业务逻辑实现，而不是代码。

60、请描述一下 Spring 中一个 Bean 的完整的生命周期。

首先要说明的是，prototype Bean 是没有生命周期的。对于 prototype Bean 而言，当程序请求该 Bean 时，Spring 容器仅仅负责创建，当容器创建了 Bean 实例之后，Bean 实例完全交给客户端代码管理，容器不再跟踪其生命周期。
其他作用域的 Bean 才有生命周期行为，其生命周期完整描述如下
1. Spring 容器读取 XML 文件中 Bean 的定义并实例化 Bean。
2. Spring 根据 Bean 的定义设置属性值。

3. 如果该 Bean 实现了 BeanNameAware 接口，Spring 将 Bean 的 id 传递给 setBeanName()方法。
4. 如果该 Bean 实现了 BeanFactoryAware 接口，Spring 将 beanFactory 传递给 setBeanFactory()方法。
5. 如果该 Bean 实现了 ApplicationContextAware 接口，Spring 将 applicationContext 传递给 setApplicationContext()方法。
6. 如果容器中配置了 Bean 后处理器（实现了 BeanPostProcessor 接口的 Bean），调用 Bean 后处理器的 postProcessBeforeInitialization()方法处理该 Bean
7. 如果该 Bean 实现了 InitializingBean 接口，调用 Bean 中的 afterPropertiesSet 方法。如果配置 Bean 时指定了 init-method 属性，调用该属性指定的初始化方法。
8. 如果容器中配置了 Bean 后处理器（实现了 BeanPostProcessor 接口的 Bean），调用 Bean 后处理器的 postProcessAfterInitialization()方法处理该 Bean
9. Bean 处于容器中运行，对外提供服务——Bean 将一直处于该状态。
10. Bean 销毁之前，如果该 Bean 实现了 DisposableBean，调用 destroy()方法。如果配置 Bean 时指定了 destroy-method 属性，调用该属性指定的销毁方法。
它的完整的生命周期如下图。



61、什么是 AOP?

AOP 为 Aspect Oriented Programming 的缩写，意为：面向切面编程（也叫面向方面），可以通过预编译方式和运行期动态代理实现在不修改源代码的情况下给程序动态地、统一添加特定功能的一种技术。
AOP 是目前软件开发中的一个热点，也是 Spring 框架中的一个重要内容。利用 AOP 可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。
AOP 是 OOP 的延续，主要的功能是：日志记录，性能统计，安全控制，事务处理，异常处理等等。

在 Spring 中提供了 AOP 的丰富支持，允许通过分离应用的业务逻辑与系统级服务（例如审计（auditing）和事务（transaction）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们并不负责（甚至是意识）其他的系统级关注点，例如日志或事务支持。

62、在 Spring 的事务体系中，事务传播特性：Required 和 RequiresNew 有何不同？

Required
如果当前存在一个事务，则加入当前事务。如果不存在任何事务，则创建一个新的事务。总之，要至少保证在一个事务中运行。PROPAGATION_REQUIRED 通常作为默认的事务传播行为。
RequiresNew
不管当前是否存在事务，都会创建新的事务。如果当前存在事务，会将当前的事务挂起（Suspend）。如果某个业务对象所做的事情不想影响到外层事务，PROPAGATION_REQUIRES_NEW 应该是合适的选择。

63、什么是 Spring？

Spring 是一个轻量级的、非侵入性的开源框架，最重要的核心概念是 IoC 和 AOP。以此为基础，Spring 上可以与各种主流的 MVC 框架整合（Spring 本身也提供了 Spring MVC 框架），下可以与各种主流的 ORM 框架、JDBC 整合，并对持久层、提供了优秀的事务支持
通过使用 Spring，我们能够减少类之间的依赖性和程序之间的耦合度，最大程度的实现松耦合，使程序更加灵活，具有更好的可扩展性。使用 Spring，可让开发人员真正的专注于业务逻辑实现，而不是代码。

64、Spring 有哪些优点？

总结起来，Spring 具有如下优点
1. 轻量级，此处的轻量级并不是说 Spring 框架的 JAR 包有多大，而是说 Spring 框架无需依赖任何特定平台和容器。Spring 既可在 Java SE 环境下运行，也可以在 Tomcat 中运行。
2. 低侵入式设计，代码的污染极低。
3. 控制反转（IoC）：Spring 的 IoC 容器降低了业务对象替换的复杂性，提高了组件之间的解耦。
4. 面向切面编程（AOP）：Spring 的 AOP 支持允许将一些通用任务如安全、事务、日志等进行集中式处理，从而提供了更好的复用。
5. Spring 的 ORM 和 DAO 支持提供了与第三方持久层框架的良好整合，并简化了底层的数据库访问。
6. Spring 框架可以与各种 Java EE 框架整合，Spring 的高度包容性使得它成为 Java EE 开发中的必选框架。
7. Spring 的高度开放性，并不强制应用完全依赖于 Spring，开发者可自由选用 Spring 框架的部分或全部。

65、Spring IoC 容器是什么？IoC 有什么优点？

Spring IoC 容器就相当于一个大工厂，Spring 容器中中所有 Bean 都由该工厂负责创建、管理。
IoC 容器负责创建 Bean、管理 Bean 之间的依赖关系（通过依赖注入）、整合 Bean、管理 Bean 的生命周期。
IOC 或依赖注入减少了应用程序的代码量。它使得应用程序的测试很简单，因为在单元测试中不再需要单例或 JNDI 查找机制。简单的实现以及较少的干扰机制使得松耦合得以实现。IOC 容器支持勤性单例及延迟加载服务。

66、Bean Factory 和 ApplicationContext 有什么区别？

Spring 容器最基本的接口就是 BeanFactory。BeanFactory 负责配置、创建、管理 Bean，它有一个子接口：ApplictionContext，因此也被称为 Spring 上下文。ApplicationContext 是 BeanFactory 的子接口，因此功能更强大。对于大部分 Java EE 应用而言，使用它作为 Spring 容器更方便。常用实现类有：
1. FileSystemXmlApplicationContext
2. ClassPathXmlApplicationContext
3. AnnotationConfigApplicationContext。
除了提供 BeanFactory 所支持的全部功能外，ApplicationContext 还有如下额外的功能。
1. ApplicationContext 默认会预初始化所有的 singleton Bean，也可通过配置取消预初始化。
2. ApplicationContext 继承 MessageSource 接口，因此提供国际化支持。
3. 资源访问，比如访问 URL 和文件。
4. 事件机制。
5. 同时加载多个配置文件。
6. 以声明式方式启动并创建 Spring 容器。

67、Spring 中的依赖注入是什么？

当 A 组件需要调用 B 组件的方法时，我们称为 A 组件依赖 B 组件。A 组件称为调用者，B 组件成为被依赖对象。
在使用依赖注入之前，调用者要么显式 new 一个被依赖对象，要么通过工厂获取被依赖对象。
使用 Spring 框架之后，调用者无须主动获取被依赖对象，调用者只要被动接受 Spring 容器为调用者的成员变量赋值即可（只要配置一个<property.../>子元素，Spring 就会执行对应的 setter 方法为调用者的成员变量赋值）。由此可见，使用 Spring 框架之后，调用者获取被依赖对象的方式由原来的主动获取，变成了被动接受——于是 Rod Johnson 将这种方式称为控制反转。
从 Spring 容器的角度来看，Spring 容器负责将被依赖对象赋值给调用者的成员变量——相当于为调用者注入它依赖的实例，因此 Martine Fowler 将这种方式称为依赖注入。

68、什么是 IoC？什么又是 DI？它们有什么区别？

关于 IoC 和 DI 的介绍可参考前一个问题。
IoC 和 DI 所介绍的其实是同一件事情，只是命名的角度不同，给这个技术命名的作者不同。

IoC 是从调用者的角度来说的，以前调用者需要**主动**去获取被依赖对象，使用 IoC 容器之后，调用者只要**被动**地接收容器为之注入被依赖对象，控制关系反过来了，因此称为控制反转（IoC）。
DI 是从 IoC 容器的角度来说的，IoC 容器负责将被依赖对象注入调用者，因此被称为依赖注入（DI）。

69、有哪些不同类型的依赖注入（IoC）？

主要有 3 种
1. 设值注入：IoC 容器使用成员变量的 setter 方法来注入被依赖对象。
2. 构造器注入：IoC 容器使用构造器来注入被依赖对象。
3. 接口注入：让调用者 Bean 实现特定接口，IoC 容器检测到该 Bean 实现该接口后会自动调用该 Bean 特定的 setter 方法，调用 setter 方法时将被依赖对象注入调用者 Bean。典型的，Bean 获取 Spring 容器、Bean 获取自身的配置 ID 分别要实现 ApplicationContextAware、BeanNameAware 接口，这就是接口注入。
对于普通开发来说，通常使用的是设值注入和构造器注入。

70、你推荐哪种依赖注入？

对于普通的项目开发来说，最常使用的是设值注入或构造器注入，接口注入往往需要对 IoC 容器进行改造，因此较少使用。
相比之下，设值注入具有如下的优点。
1. 与传统的 JavaBean 的写法更相似，程序开发人员更容易理解、接受。通过 setter 方法设定依赖关系显得更加直观、自然。
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，难以阅读。Spring 在创建 Bean 实例时，需要同时实例化其依赖的全部实例，因而导致性能下降。而使用设值注入，则能避免这些问题。
3. 尤其是在某些成员变量可选的情况下，多参数的构造器更加笨重。
但构造器注入也有其优点：构造注入可以在构造器中决定依赖关系的注入顺序，优先依赖的优先注入。
一般建议设值注入为主，构造器注入为辅。

71、什么是 Spring Bean？

Spring 的 Bean 就是 Spring 容器搜管理的 Java 对象，因此 Spring 容器的 Bean 并不是 Java Bean。
传统的 JavaBean 和 Spring 中的 Bean 存在如下区别。
1. 用处不同：传统的 JavaBean 更多是作为值对象传递参数；Spring 的 Bean 用处几乎无所不包，任何 Java 对象都可被称为 Bean。
2. 写法不同：传统的 JavaBean 作为值对象，要求每个属性都提供 getter 和 setter 方法；但 Spring 的 Bean 只需为接受设值注入的属性提供 setter 方法即可。
3. 生命周期不同：传统的 JavaBean 作为值对象传递，不接受任何容器管理其生命周期；Spring 中的 Bean 由 Spring 管理其生命周期行为。

72、如何向 Spring 容器提供配置元数据？

有如下方式向 Spring 容器提供配置元数据：
1. 使用 XML 配置文件：这是最主流的配置文件的方式。这种方式采用一个以<beans.../>为根元素的 XML 文件为 Spring 配置元数据。
2. 基于注解的配置：这种方式也被称为“零配置”，主要通过使用@Component、@Controller、@Serviced 等注解来修饰 Bean 类完成配置。
3. 使用 Groovy 配置：这种方式是当前 Spring 引入的配置方式，采用 Groovy 语法来完成 Bean 配置，这种方式可以充分利用 Groovy 语言的灵活性。
4. 使用 Java 的配置：这种方式使用 Java 代码作为配置文件，这种方式采用一个@Configuration 修饰 Java 类来完成配置。这种方式还需要使用@Bean 注解来定义 Bean，这种方式可以利用 Java 语言的灵活性。
实际开发中，往往还是使用 XML 配置文件和基于注解的“零配置”比较常用。

73、说一下 Spring 中支持的 Bean 作用域。

在 Spring 配置文件中定义 Bean 时，允许通过 scope 指定它的作用域，也可通过@Scope 注解来指定 Bean 的作用域。现在 Spring 容器中 Bean 共有 6 个作用域。
1. singleton：单例模式，在整个 Spring IoC 容器中，singleton 作用域的 Bean 将只生成一个实例。这是默认的作用域。
2. prototype：每次通过容器的 getBean()方法获取 prototype 作用域的 Bean 时，都将产生一个新的 Bean 实例。
3. request：对于一次 HTTP 请求，request 作用域的 Bean 将只生成一个实例，这意味着，在同一次 HTTP 请求内，程序每次请求该 Bean，得到的总是同一个实例。只有在 Web 应用中使用 Spring 时，该作用域才真正有效。
4. session：对于一次 HTTP 会话，session 作用域的 Bean 将只生成一个实例，这意味着，在同一次 HTTP 会话内，程序每次请求该 Bean，得到的总是同一个实例。只有在 Web 应用中使用 Spring 时，该作用域才真正有效。
5. application：对应整个 Web 应用，该 Bean 只生成一个实例。这意味着，在整个 Web 应用内，程序每次请求该 Bean，得到的总是同一个实例。只有在 Web 应用中使用 Spring 时，该作用域才真正有效。
6. websocket：在整个 WebSocket 的通信过程中，该 Bean 只生成一个实例。只有在 Web 应用中使用 Spring 时，该作用域才真正有效。
需要说明的是，早期 Spring 为 Portlet 提供的 global session 作用域已经被删除了，而是添加 application、websocke 两个新的作用域。

74、Spring 框架中单例 Bean 是线程安全的吗？

Spring 框架并不保证 Bean 的线程安全。因此 Spring 框架中的单例 Bean 通常不是线程安全的。
如果开发者希望单例 Bean 在多线程环境下使用，则建议自己通过代码来保证该 Bean 是线程安全的。

75、哪些是最重要的 Bean 生命周期方法？能重写它们吗？

当 Bean 实现了 InitializingBean 接口时，该接口中 afterPropertiesSet 方法就变成了 Bean 的生命周期方法，该方法会在 Bean 的所有属性被设置之后被调用。开发者可以重写这个方法。
当 Bean 实现了 DisposableBean 接口时，该接口中 destroy 方法就变成了 Bean 的生命周期方法，该方法会在 Bean 销毁之前被调用。开发者可以重写这个方法。
此外，还可以在配置文件中通过 init-method 属性（对应于@PostConstruct）来指定生命周期方法，该方法也会在 Bean 的所有属性被设置之后被调用；也可在配置文件中通过 destroy-method 属性（对应于@PreDestroy）来指定生命周期方法，该方法也会在 Bean 销毁之前被调用。

76、什么是 Spring 的 nested Bean（嵌套 Bean）？

如果把<bean.../>配置成<property.../>、<constructor-args.../>或<list.../>、<set.../>、<map.../>等元素的子元素，那么该<bean.../>元素配置的 Bean 仅仅作为 setter 注入、构造注入的参数、集合属性的元素，这种 Bean 就是嵌套 Bean。由于容器不能获取嵌套 Bean，因此它不需要指定 id 属性。

77、如何在 Spring 中注入 Java 集合属性？

Spring 使用<list.../>、<set.../>、<map.../>、<props.../>等元素来配置集合属性。使用它们时，还需要为集合配置元素。由于集合元素又可以是基本类型值、引用容器中的其他 Bean、嵌套 Bean 或集合属性等，所以<list.../>、<key.../>和<set.../>元素又可接受如下子元素。
1. value：指定集合元素是基本数据类型值或字符串类型值。
2. ref：指定集合元素是容器中的另一个 Bean 实例。
3. bean：指定集合元素是一个嵌套 Bean。
4. list、set、map 及 props：指定集合元素又是集合。
由于<props.../>元素用于配置 Properties 类型的参数值，Properties 类型是一种特殊的类型，其 key 和 value 都只能是字符串，故 Spring 配置 Properties 类型的参数值比较简单：每个 key-value 对只要分别给出 key 和 value 就足够了。

78、什么是 Bean wiring（Bean 装配）？

所谓 Bean 装配，说的就是管理 Bean 之间的依赖关系：当 A 组件需要调用 Bean 组件时，IoC 容器自动将被依赖的 B 组件注入 A 组件，这就完成了 A、B 组件的装配。

79、什么是 Bean 自动装配？解释各种自动装配模式。

所谓自动装配指的是使用 ref 显式指定依赖 Bean（也不使用@Resource 注解），而是由 Spring 容器检查 XML 配置文件内容，根据某种规则，为调用者 Bean 注入被依赖的 Bean。
自动装配可通过<beans.../>元素的 default-autowire 属性指定，该属性对配置文件中所有的 Bean 起作用；也可通过<bean.../>元素的 autowire 属性指定，该属性只对该 Bean 起作用。
自动装配支持如下模式（也就是 autowire 属性支持的值）

1。 no：不使用自动装配。Bean 依赖必须通过 ref 元素定义。这是默认的配置，在较大的部署环境中不鼓励改变这个配置，显式配置合作者能够得到更清晰的依赖关系。
2. byName：根据 setter 方法名进行自动装配。Spring 容器查找容器中的全部 Bean，找出其 id 与 setter 方法名去掉 set 前缀、并小写首字母后同名的 Bean 来完成注入。如果没有找到匹配的 Bean 实例，则 Spring 不会进行任何注入。
3. byType：根据 setter 方法的形参类型来自动装配。Spring 容器查找容器中的全部 Bean，如果正好有一个 Bean 类型与 setter 方法的形参类型匹配，就自动注入这个 Bean；如果找到多个这样的 Bean，就抛出一个异常；如果没找到这样的 Bean，则什么都不会发生，setter 方法不会被调用。
4. constructor：与 byType 类似，区别是用于自动匹配构造器的参数。如果容器不能恰好找到一个与构造器参数类型匹配的 Bean，则会抛出一个异常。
5. autodetect：Spring 容器根据 Bean 内部结构，自行决定使用 constructor 或 byType 策略。如果找到一个默认的构造函数，那么就会应用 byType 策略。

80、自动装配有哪些局限性？

自动装配有如下局限性：
1. 自动装配不支持标量类型：如果属性类型是 8 个基本类型及其包装类、String 类型，自动装配都不能工作。
2. 模糊特性：自动装配不如显式指定依赖精确。
可见：自动装配可以减少配置文件的工作量，但降低了依赖关系的透明性和清晰性。

81、你可以在 Spring 中注入 null 或空字符串吗？

当然可以。Spring 的依赖注入只负责通过反射调用 setter 方法或构造器——因此主要被注入的属性本身允许被赋值为 null，Spring 自然可以为它注入 null 值或空字符串。
需要说明的是，如果将被依赖对象注入为 null，程序后面调用被依赖对象时肯定会引发 NPE 异常。

82、如何开启基于注解的“零配置”装配？

需要在 Spring 配置文件中使用<context:component-scan.../>元素来指定搜索哪些包及其子包下的 Bean 类。例如如下配置文件：

```
<?xml version="1.0" encoding="GBK"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
    <context:component-scan
        base-package="org.crazyit.app.service"/>
</beans>
```

接下来即可使用注解来修饰 Spring Bean 类，使之成为 Spring 容器中的 Bean。

83、@Required 注解？

@Required 表明 Bean 的属性必须在配置时设置，为了避免开发者的疏忽，可以让 Spring 在创建容器时就执行检查，此时需要为 setter 方法添加@Required 修饰，此时 Spring 会检查该 setter 方法：如果开发者既没有显式通过<property.../>配置依赖注入；也没有使用自动装配执行依赖注入，Spring 容器会报异常：BeanInitializationException。

84、@Autowired 注解？

Spring 提供了@Autowired 注解来指定自动装配，@Autowired 可以修饰 setter 方法、普通方法、实例变量和构造器等。当使用@Autowired 标注 setter 方法时，默认采用 byType 自动装配策略。

85、@Qualifier 注解？

为了实现精确的自动装配，Spring 提供了@Qualifier 注解，通过使用@Qualifier，允许根据 Bean 的 id 来执行自动装配。

86、Spring 框架中有哪些不同类型的事件？

ApplicationContext 提供了事件支持，并允许在代码中监听器容器事件的功能。
如果容器中某个 Bean 实现了 ApplicationListener 接口，那么该 Bean 将作为容器的事件监听器，当容器中一个 ApplicationEvent 被发布以后，该 Bean 会被自动激发。
Spring 提供了以下 5 中标准的事件：
1. 容器更新事件（ContextRefreshedEvent）：该事件会在 ApplicationContext 被初始化或者更新时发布。也可以在调用 ConfigurableApplicationContext 接口中的 refresh()方法时被触发。
2. 容器开始事件（ContextStartedEvent）：当容器调用 ConfigurableApplicationContext 的 start()方法开始/重新开始容器时触发该事件。
3. 容器停止事件（ContextStoppedEvent）：当容器调用 ConfigurableApplicationContext 的 stop()方法停止容器时触发该事件。
4. 容器关闭事件（ContextClosedEvent）：当 ApplicationContext 被关闭时触发该事件。容器被关闭时，其管理的所有单例 Bean 都被销毁。
5. 请求处理事件（RequestHandledEvent）：在 Web 应用中，当一个 HTTP 请求（request）结束触发该事件。
除了上面介绍的事件以外，还可以通过继承 ApplicationEvent 类来开发自定义的事件。
开发自定义事件之后，程序可调用 ApplicationContext 的 publishEvent 方法来发布自定义事件。

87、Spring 框架中都用到了哪些设计模式？

Spring 框架中使用到了大量的设计模式，下面列举了比较有代表性的：
1. 工厂模式：Spring 本身就是一个超级大工厂，Spring 容器负责创建所有 Bean，因此 Spring 框架本身就是从工厂模式扩展而来的。
2. 单例模式。容器中 singleton Bean 都是单例的。
3. 代理模式。Spring AOP 就是基于代理模式的，由于所有 AOP 代理和目标对象之间的关系就是代理和目标之间的关系。
4. 策略模式。Spring 的事务控制、资源访问都是策略模式的典型应用。其中事务控制的策略接口就是 PlatformTransactionManager，该接口定义了事务控制的 3 个核心方法，Spring 为该策略接口提供了大量的策略实现类；资源访问的策略接口是 Resource 接口，Spring 为该接口提供了大量的策略实现类。

88、在 Spring 框架中如何更有效地使用 JDBC？

Spring 为 JDBC 访问提供了 JdbcDaoSupport 和 JdbcTemplate 两个工具类，其中 JdbcDaoSupport 可作为所有基于 JBDC 的 DAO 组件的基类，JdbcDaoSupport 提供了一个 setDataSource 方法，因此可让 Spring 容器为继承了 JdbcDaoSupport 的 DAO 组件注入 DataSource。
JdbcDaoSupport 还提供另一个 getJdbcTemplate()方法，该方法返回一个 JdbcTemplate 对象。JdbcTempalate 把绝大部分通用的数据库访问操作定义成了一个模板方法，开发者只要调用 JdbcTempalate 的方法即可完成大部分 CRUD 操作；如果开发者需要完成更复杂的数据库访问时，也可调用 JdbcTemplate 的 execute()方法实现。

89、使用 Spring 可以通过什么方式访问 Hibernate？

使用 Spring 有两种方式使用 Hibernate：
1. 传统方式是使用 HibernateDaoSupport 和 HibernateTemplate，在这种方式下，所有基于 Hibernate 的 DAO 组件都应该继承 HibernateDaoSupport；继承了 HibernateDaoSupport 的 DAO 组件只要容器为之注入 SessionFactory，然后该 DAO 组件即可调用 getHibernateTemplate()返回 HibernateTemplate 对象，HibernateTemplate 把绝大部分通用的数据库访问操作定义成了一个模板方法，开发者只要调用 HibernateTemplate 的方法即可完成大部分 CRUD 操作；如果开发者需要完成更复杂的数据库访问时，也可调用 HibernateTemplate 的 execute()方法实现。
需要说明的是，从 Hibernate 4 开始，Spring 已经不再推荐使用这种方式。因此目前在企业开发中推荐使用下面方式。
2. Spring 容器负责管理 Hibernate 的 SessionFactory，Spring 容器还负责将 SessionFactory 注入 DAO 组件，这些 DAO 组件可通过 SessionFactory 的 getCurrentSession()方法来获取 Session，然后通过 Session 来完成 CRUD 操作。

90、Spring 支持哪些 ORM？

早期 Spring 主动支持 Hibernate、iBatis（现改名为 MyBatis）、TopLink、JDO、JPA 等各种 ORM 技术。现在 Spring 主要支持 Hibernate 和 JPA 这两种持久化技术。
但实际上由于 Spring 现在已经是 Java EE 领域最流行的框架，因此基本上所有的 ORM 技术都可以与 Spring 整合。

91、Spring 的事务管理有哪些优点？

为不同的事务 API（如 JTA、JDBC、Hibernate、JPA）提供了统一的编程模型。
支持声明式事务管理。
为编程式事务提供了简化的、统一的编程 API。
可以和 Spring 的多种数据访问技术很好地整合。

92、Spring 支持的事务管理类型？

Spring 支持如下两种方式的事务管理方式：
1. 编程式事务，通过开发者自行编写程序来控制事务。通常可选择面向 PlatformTransactionManager 或 TransactionTemplate 进行编程。这种方式带来了很大的灵活性，但很难维护。
2. 声明式事务，在配置文件或注解中配置事务（推荐使用）。这种方式可将事务管理代码和业务代码分离。只需通过注解或者 XML 配置即可管理事务。因此具有很好的可维护性。
声明式事务又分为两种：
1. 基于 XML 的声明式事务
2. 基于注解的声明式事务

93、你更推荐哪种类型的事务管理？

我更推荐使用声明式事务，这种方式可将事务管理代码和业务代码分离。只需通过注解或者 XML 配置即可管理事务。因此具有很好的可维护性。
但在一些有特殊要求的事务控制场景下，也可以通过少量的编程式事务进行辅助。

94、Spring 中 AOP 的应用场景、AOP 原理、好处？

AOP 专门用于处理系统中分布于各个模块（不同方法）中的交叉关注点的问题，在 Java EE 应用中，常常通过 AOP 来处理一些具有横切性质的系统级服务，如事务管理、安全检查、缓存、对象池管理等，AOP 已经成为一种非常常用的解决方案。
通俗来说，当我们需要为项目中某一批方法（分布于各个模块）增加某种通用的功能。此时 AOP 可达到的效果是，保证在程序员不修改源代码的前提下，为系统中业务组件的多个业务方法添加某种通用功能。
AOP 框架的本质是，依然要去修改业务组件的多个业务方法的源代码——只是这个修改由 AOP 框架完成，程序员不需要修改！
AOP 实现可分为两类（按 AOP 框架修改源代码的时机）。
1. 静态 AOP 实现：AOP 框架在编译阶段对程序进行修改，即实现对目标类的增强，生成静态的 AOP 代理类（生成的*.class 文件已经被改掉了，需要使用特定的编译器）。以 AspectJ 为代表。
2. 动态 AOP 实现：AOP 框架在运行阶段动态生成 AOP 代理（在内存中以 JDK 动态代理或 cglib 动态地生成 AOP 代理类），以实现对目标对象的增强。以 Spring AOP 为代表。
AOP 编程的优点：
1. 各步骤之间的良好隔离性，大大降低耦合度。

2. 源代码无关性，为项目扩展新功能时不需要修改源代码。

95、介绍 Spring 支持的 5 种 Advice（增强处理）。

Spring 可以支持以下五种类型的 Advice：
1. before Advice（前置增强处理）：在目标方法执行之前织入的 Advice。
2. after Advice（最终增强处理）：在目标方法执行结束后（不论是正常返回还是异常退出）织入的 Advice。
3. after-returning Advice（后置增强处理）：在目标方法正常完成之后织入的 Advice。
4. after-throwing Advice（异常增强处理）：在目标方法由于未捕获异常结束之后织入的 Advice。
5. around Advice（环绕增强处理）：在目标方法执行前后都织入的 Advice。

96、引入（Introduction）和织入（Weaving）的区别？

引入和织入都是 AOP 框架对目标对象的修改方式。
而引入指的是为被处理的类添加方法或字段，Spring 允许将新的接口引入到任何被处理的对象。
织入指的是将 Advice 添加到目标对象中，并创建一个被增强的对象（AOP 代理）的过程。织入有两种实现方式——编译时增强（如 AspectJ）和运行时增强（如 Spring AOP）。Spring 和其他纯 Java AOP 框架一样，在运行时完成织入。

97、什么是目标对象？

目标对象就是还没有被 AOP 框架所修改、增加的对象。目标对象将会被 AOP 框架进行修改、增强处理，因此也被称为“被增强的对象”。如果 AOP 框架采用的是动态 AOP 实现，那么该对象就是一个被代理的对象。

98、什么是代理？

代理就是 AOP 框架所创建的对象，简单地说，代理就是对目标对象的加强。Spring 中的 AOP 代理可以是 JDK 动态代理，也可以是 CGLIB 代理。前者为实现接口的目标对象的代理，后者为不实现接口的目标对象的代理。

99、Spring 的 AOP 代理有什么实现方式？

Spring AOP 代理有两种实现方式：
1. 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。
2. 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，所以系统的解耦不如使用 JDK 的动态代理好。

100、什么是 Spring MVC 框架？

Spring 框架既可与其他第三方 MVC 框架（如 Struts 2）整合，也可直接使用 Spring 本身提供的 Spring MVC 框架。因此 Spring MVC 就是一个类似于 Struts 2 的 MVC 框架。

101、DispatcherServlet？

DispatcherServlet 是 Spring MVC 框架的核心控制器，Spring MVC 框架用它来处理所有的 HTTP 请求，DispatcherServlet 收到用户请求之后，再将用户请求转发给特定的业务控制器。

102、WebApplicationContext？

WebApplicationContext 是 ApplicationContext 的子接口，主要适合在 Web 应用中使用。WebApplicationContext 在 ApplicationContext 基础上增加了 getServletContext()方法，该方法可获取它所在的 ServletContext 对象（application 对象）。
在 Web 应用中使用 WebApplicationContext 时，通常每个 Web 应用都对应一个 root 容器，而应用中其他 Bean 组件（包括 DispatchServlet）可能都在自己的子容器。

103、什么是 Spring MVC 框架的控制器？

与 Struts 2 一样，Spring MVC 框架的控制器由 2 部分组成：
1. 核心控制器：由于 Spring 的 DispatchServlet 代表。
2. 业务控制器：由于开发者自己编写的业务控制器代表，该业务控制器需要使用@Controller 修饰，处理方法通常会使用@RequestMapping 等注解修饰。

104、@Controller 注解？

Spring MVC 使用@Controller 修饰控制器类。

105、@RequestMapping 注解？

Spring MVC 使用@RequestMapping 注解修饰控制器的处理方法，该注解用于将处理方法映射到特定 URL。与该注解类似的，Spring 还提供了@PostMapping、@PutMapping、@DeleteMapping、@PatchMapping 和@RequestMapping 等注解。

106、Spring MVC 的流程？

1. 浏览器发送请求至核心控制器：DispatcherServlet。

2. DispatcherServlet 收到请求调用映射处理器（HandlerMapping）处理请求和业务控制器。
3. 映射处理器根据请求 URL 找到对应的业务控制器、创建业务控制器实例以及业务控制器的拦截器（如果有则创建），然后一并返回给 DispatcherServlet。
4. 执行业务控制器（也就是使用@Controller 修饰的组件）。
5. Controller 执行完成后将结果封装在 ModelAndView 中，然后将 ModelAndView 返回该 DispatcherServlet。
6. DispatcherServlet 将 ModelAndView 传给视图解析器（ViewReslover）。
7. 视图解析器（ViewReslover）解析后返回具体的视图页面。
8. DispatcherServlet 将数据模型传入视图页面，调用视图页面对浏览器生成响应。