考察重点内容：

A. **以理论知识为主，涉及的重要概念，不考标签和配置的具体内容，有些标签可以用文字描述**
B. **HTTP 协议、XML、Servlet、JSP、JavaBean、Spring、Hibernate**
C. **框架不需要看具体配置，了解框架的用途和方法**

1. JavaEE 为应用程序组件定义了四种容器：
   Web、Enterprise JavaBean(EJB)、应用程序客户机、Applet
2. 典型 JavaEE 结构的应用程序包括四层：
   客户层、表示逻辑层（Web 层）、业务逻辑层、企业信息系统层
3. JavaBean 是一种 Java 语言写成的可重用组件：
   1） 类是公共的
   2） 有一个无参的公共的构造器
   3） 属性必须声明为 private，方法必须声明为 public，且有对应的 get、set 方法
4. HTTP 是 hypertext transfer protocol (**超文本传输协议**)的简写，它是 TCP/IP 协议集中的一个应用层协议，用于定义 **WEB 浏览器**与 **WEB 服务器**之间交换数据的过程以及数据本身的格式。HTPP 协议为“请求和响应”协议。
5. 客户端请求包括：请求行、头、信息体
   客户端请求的方法：
   1） GET、POST、HEAD
   2） DELETE、TRACE、PUT
6. 服务器接收到请求后，返回 HTTP 响应。
   每个响应包括：状态行、头、信息体
7. 浏览器与服务器之间的通信过程经历的四个步骤：
   1） 建立连接
   2） 发出请求信息
   3） 回送响应信息
   4） 关闭连接
8. POST 请求有请求体，GET 请求没有请求体。
9. GET 与 POST 请求的生成方式
   get 方式有四种：
   1）直接在 URL 地址栏中输入 URL。
   2）网页中的超链接。
   3）form 中 method 为 get。
   4）form 中 method 为空时，默认是 get 提交。

   Post 方式：form 中 method 属性为 post。

   数据传送方式
   1）get 方式：表单数据存放在 URL 地址后面。所有 get 方式提交时，**HTTP 中没有消息体**。
   2）post 方式：表单数据存放在 **HTTP 协议的消息体**中以**实体**的方式传送到服务器
10. 响应码：
    1）200：请求成功，浏览器会把响应体内容（通常是 html）显示在浏览器中

2）404：请求的资源没有找到，说明客户端错误的请求了不存在的资源

3）500：请求资源找到了，但服务器内部出现了错误

4）302：重定向，当响应码为 302 时，表示服务器要求浏览器重新再发一个请求服务器会重新发送一个响应头 Location，它指定了新请求的 URL 地址

11. JSP 脚本元素有三种

1）脚本片段：在 JSP 页面中执行的 java 代码，<% System.out.println("Hello World");%>

2）表达式：在 JSP 页面中执行的表达式，<%=str %>

3）声明：在 JSP 页面中定义变量或者方法，<%!Java 代码>

12. JSP 九大隐含对象：

1）**out**（JspWriter）：相当于 response.getWriter()获取的对象，用于在页面中显示信息。常用方法：out.print()。

2）**request**（HttpServletRequest）：HttpServletRequest 对象，封装了用户提交的信息，使用该对象可以获取用户提交的信息（如 form 表单传参、url 传参）。

常用方法：request.getParameter()
request.setAttritute()
request.getAttribute()
request.setCharacterEncoding()

3）**response**（HttpServletResponse）：HttpServletResponse 对象，封装了响应信息，客户端发出每个请求时，服务器都会创建一个 response 对象用来对客户端进行响应。

常用方法：response.sendRedirect()

4）**session**（HttpSession）：HttpSession 对象，称为“会话控制”。session 对象存储特定用户会话所需的属性及配置信息。当用户在应用程序的 Web 页之间跳转时，存储在 session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。

常用方法：session.setAttribute()
session.getAttribute()

补充：

a) session 对象默认 30 分钟没有使用就会被服务器会自动销毁。

b) 配置 session 的失效时间有 3 种方法：
——在 tomcat 的 config 下的 web.xml 文件配置
——在项目里的 web.xml 配置
——Java 代码配置
优先级：1<2<3

c) web.xml 中的配置（以分钟为单位）：

```
<session-config>
    <session-timeout>1</session-timeout>
</session-config>
```

Java 代码配置（以秒为单位）：

```
session.setMaxInactiveInterval(15);
```

d）手工调用 session.invalidate()方法，摧毁 session

5）**config**（ServletConfig）：对应 Servlet 中的 ServletConfig 对象。

6）**page**（Object）：对应当前 Servlet 对象，实际上就是 this。

7）**pageContext**（PageContext）：当前页面的上下文，也是一个域对象。

8）**exception**（Throwable）：错误页面中异常对象

9）**application**（ServletContext）：ServletContext 对象

13. JavaWeb 的四个域对象（**scope**）：页面、请求、会话和整个应用

1）pageContext：

a）范围：当前 JSP 页面

b）注意：该对象只能在 **JSP 中获取，Servlet 中没有**

2）Request：**一次请求响应完成后，就会销毁**。

3）Session：范围为当前会话，**在一个用户会话期间有效**。就是打开浏览器到关闭浏览器，这一个完整的上网过程叫做一个会话。只要没有关闭浏览器或设置 session 失效，就可以在域中获取到 Session 中的数据。

4）Application：范围为当前应用，应用是最高级的域对象，它代表**整个 WEB 应用**，在这个域对象中设置的数据在所有的域中都能获取。

14. JSP 参数标记的基本语法为：<jsp: param name=”name” value=”value”/>
15. 读取 request 单值参数的方法是 **getParameter()**，读取多值参数的方法是 **getParameterValues ()**，获取所有参数名称的方法是 **getParameterNames()**。
16. request 作用范围变量可以通过 setAttribute() 和 getAttribute ()方法设置和读取变量的数据。
17. 域对象都有三个操作数据的主要方法：

1）在当前域中放入数据：public void setAttribute(String name, Object o);

2）根据名字获取当前域中的数据：public Object getAttribute(String name);

3）根据名字删除当前域中的数据：public void removeAttribute(String name);

18. JSP 指令标签

1） page：通常位于 jsp 页面的顶端，同个页面可以有多个 page 指令。

page指令语法：

```
<%@ page 属性1="属性值" 属性2="属性值1,属性值2"...
         属性n="属性值n"%>
```

属性	描述	默认值
language	指定JSP页面使用的脚本语言	java
import	通过该属性来引用脚本语言中使用到的类文件	无
contentType	用来指定JSP页面所采用的编码方式	text/html, ISO-8859-1

2） include：将一个外部文件嵌入到当前 JSP 文件中，同时解析这个页面中的 JSP 语句。通知 JSP 编译器把另外一个文件完全包含入当前文件中。效果就好像被包含文件的内容直接被粘贴到当前文件中一样。

<%@ include file="somefile.jsp" %>

3） taglib 指令（了解）：使用标签库定义新的自定义标签，在 JSP 页面中启用定制行为。标签库指令描述了要使用的 JSP 标签库。该指令需要指定一个标签前缀 prefix 和标签库的描述 uri（统一资源标识符）

<%@ taglib prefix="myprefix" uri="taglib/mytag.tld" %>

19. JSP 动作标签：与 HTML 标签不同，HTML 标签由浏览器来解析，而 JSP 动作标签需要

服务器（Tomcat）来运行。

1）转发标签<jsp:forward>（page 属性）：在页面中用于转发操作。

2）包含标签<jsp:include>（page 属性）：动态包含，将其他页面包含到当前页面中。

3）操作 JavaBean 的动作标签：<jsp:useBean>（id,class,scope 属性）、<jsp:setProperty>、<jsp:getProperty>（javaBean：公有、无参构造方法、属性私有、getter 和 setter 方法）

示例：

<jsp:useBean id="person" scope="page" class="com.entity.person.Person"/>

20. <jsp:useBean>标记的 scope 属性可取 page、request、session 和 application。
21. JSP 中动态包含与静态包含的区别：
    1) 静态包含使用 include 指令，动态包含使用<jsp:inclued>标签。
    2) 静态包含会直接将目标页面复制到生成的 Servlet 中，动态包含是在生成的 servlet 中使用 include()方法来引入目标页面。
    3) 当目标页面发生改变时，静态包含不能体现，动态包含可以体现。
22. Servlet（Server Applet）：服务器端小程序
    1) 狭义：javax.servlet.Servlet 接口及其子接口
    2) 广义：指实现了 Servlet 接口的实现类
23. 使用 Servlet 接口的方式（创建一个 servlet）：
    1）创建一个类实现 Servlet 接口
    2）在 web.xml 中“注册”这个实现类
    3）**Tomcat（Servlet 容器）**会创建实现类对象
24. 在 web.xml 中配置和映射 servlet：

```
//声明 servlet 对象
<servlet>
    <servlet-name>PersonServlet</servlet-name>
    <servlet-class>com.servlet.person.PersonServlet</servlet-class>
</servlet>
//映射 Servlet
<servlet-mapping>
    <servlet-name>PersonServlet</servlet-name>
    <url-pattern>/person</url-pattern>
</servlet-mapping>
```

25. Servlet 的生命周期：

    1）初始化（**初始化阶段**）：正常只被调用一次，对应 init()方法。正常情况下第一次使用 Servlet 的时候才进行初始化的操作，也可以通过配置在容器启动的时候自动进行初始化<load-on-startup>。

    2）服务（**处理请求阶段**）：被服务多次，所有的服务器都由 services()方法（service()方法用于处理客户端的请求）分配，主要有 doGet()和 doPost()，分别处理 get 和 post 请求。

    3）销毁（**销毁阶段**）：正常只被一次，调用 destory()方法。如果一个 Servlet 长时间不用，也会自动销毁，而当再次使用的时候就必须重新进行初始化的操作。
26. ServletConfig：代表 Servlet 配置信息
    1）对象：由 Servlet 容器创建，并传入 init(ServletConfig config)方法
    2）作用：
        a) 获取 Servlet 的配置名称

b) 获取 Servlet 初始化参数

```
<!--配置 servlet 的初始化参数-->
<init-param>
    <param-name>user</param-name>
    <param-value>root</param-value>
</init-param>
```

c) 获取 ServletContext 对象

27. ServletContext：Servlet 上下文，代表当前 Web 应用
    1）对象：由 Servlet 容器创建，通过 ServletConfig 对象获取
    2）作用：
        a）获取 WEB 应用程序的初始化参数
        b）获取虚拟路径所映射的本地路径
        c）application 域范围的属性
28. HttpServletRequest：代表 HTTP 请求，对象由 Servlet 容器创建
    功能：
        1) 获取请求参数
        2) 获取请求路径即 URL 地址相关信息
        3) 在请求域中保存数据
        4) 转发请求
29. HttpServletResponse：代表 HTTP 响应，对象由 Servlet 容器创建
    功能：
        1) 向浏览器输出数据
        2) 重定向请求
30. 请求转发：一个 web 资源收到客户端请求后，通知**服务器**去调用另外一个 web 资源进行处理，称之为请求转发。
31. 请求重定向：一个 web 资源收到客户端请求后，通知**客户端**去访问另外一个 web 资源，称之为请求重定向。
32. 请求转发与请求重定向的区别：
    1）request.getRequestDispatcher().**forward**(request,response)，是一次请求，转发后前一次请求对象会保存，地址栏 url 不会变。
    2）response.send**Redirect**()，客户端行为，等同于两次请求，重定向后前一次请求对象不会保存，地址栏 url 会改变。
    forward 是把另一个页面加载到本页面，不改变浏览器的路径；redirect 是跳转到另一个页面，会改变浏览器的路径。
33. filter（过滤器）：Servlet 过滤器是客户端与目标资源间的中间层组件，用于拦截客户端的请求与响应信息。
34. filter 的配置：

```
//声明过滤器对象
<filter>
    <filter-name>helloFilter</filter-name>
    <filter-class>com.person.HellpFilter</filter-class>
</filter>
//映射过滤器
<filter-mapping>
```

<filter-name>helloFilter</filter-name>
<url-pattern>/test.jsp</url-pattern>
</filter-mapping>

35. JDBC（Java Database Connectivity）是一个**独立于特定数据库管理系统、通用的 SQL 数据库存取和操作的公共接口**（一组 API），定义了用来访问数据库的标准 Java 类库，（java.sql,javax.sql）使用这个类库可以以一种标准的方法、方便地访问数据库资源。
36. JDBC 接口（API）包括两个层次：
    1) 面向应用的 API：Java API，抽象接口，供应用程序开发人员使用（连接数据库，执行 SQL 语句，获得结果）
    2) 面向数据库的 API：Java Driver API，供开发商开发数据库驱动程序用
37. 具体描述 Spring：
    1) **轻量级**：Spring 是非侵入性的 - 基于 Spring 开发的应用中的对象可以不依赖于 Spring 的 API。
    2) **依赖注入**（DI --- dependency injection、IOC）
    3) **面向切面编程**（AOP --- aspect oriented programming）
    4) **容器**：Spring 是一个容器，因为它包含并且管理应用对象的生命周期。
    5) **框架**:：Spring 实现了使用简单的组件配置组合成一个复杂的应用。在 Spring 中可以使用 XML 和 Java 注解组合这些对象
    6) **一站式**：在 IOC 和 AOP 的基础上可以整合各种企业应用的开源框架和优秀的第三方类库（实际上 Spring 自身也提供了展现层的 SpringMVC 和 持久层的 Spring JDBC）。
38. Spring 的四个特点：
    1) 控制反转
    2) 持久性的封装和事物管理
    3) 面向切面编程
    4) 提供了对 web 的多种支持
39. 配置 Bean：
    1) 配置形式：基于 XML 文件的方式；基于注解的方式
    2) 配置方式：通过全类名（反射）、FactoryBean
40. IOC 和 DI：
    1) IOC(Inversion of Control)：其思想是反转资源获取的方向，应用 IOC 之后，容器主动地将资源推送给它所管理的组件，组件所要做的仅是选择一种合适的方式来接受资源。这种行为也被称为查找的被动形式。
    2) DI(Dependency Injection) — IOC 的另一种表述方式：即组件以一些预先定义好的方式(例如: setter 方法)接受来自如容器的资源注入。
41. 依赖注入的方式：
    1) 属性注入（Set 方法注入）
        a) 属性注入使用 <property> 元素，使用 name 属性指定 Bean 的**属性名称**，value 属性或 <value> 子节点指定**属性值**。
        b) 示例：

```
<!-- 通过全类名的方式来配置 bean -->
<bean id="helloWorld"
    class="com.atguigu.spring.helloworld.HelloWorld">
    <property name="userName" value="atguigu"></property>
</bean>
```

2) **构造器注入**

a) 通过构造方法注入 Bean 的属性值或依赖的对象，它保证了 Bean 实例在实例化后就可以使用。

b) 构造器注入在 <constructor-arg> 元素里声明属性，<constructor-arg> 中没有 name 属性。

c) 按索引匹配入参：

```
<bean id="car" class="com.atguigu.spring.helloworld.Car">
    <constructor-arg value="奥迪" index="0"></constructor-arg>
    <constructor-arg value="长春一汽" index="1"></constructor-arg>
    <constructor-arg value="500000" index="2"></constructor-arg>
</bean>
```

按类型匹配入参：

```
<bean id="car" class="com.atguigu.spring.helloworld.Car">
    <constructor-arg value="奥迪" type="java.lang.String"/>
    <constructor-arg value="长春一汽" type="java.lang.String"/>
    <constructor-arg value="500000" type="double"/>
</bean>
```

3) 工厂方法注入（很少使用，不推荐）

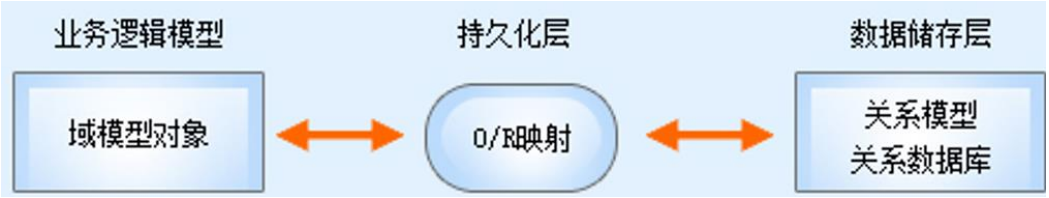
42. IOC 容器：
    1) BeanFactory：IOC 容器的基本实现
    2) ApplicationContext：提供了更多的高级特性，是 BeanFactory 的子接口
    3) BeanFactory 是 Spring 框架的基础设施，**面向 Spring 本身**；ApplicationContext **面向使用 Spring 框架的开发者**，几乎所有的应用场合都直接使用 ApplicationContext 而非底层的 BeanFactory。
43. Spring 容器相关的核心接口是 BeanFactory 和 ApplicationContext，默认的配置文件是 applicationContext. xml
44. Bean 的作用域：在 Spring 中，可以在 <bean> 元素的 scope 属性里设置 Bean 的作用域。

类别	说明
singleton	在 SpringIOC 容器中仅存在一个 Bean 实例，Bean 以单实例的方式存在
prototype	每次调用 getBean() 时都会返回一个新的实例
request	每次 HTTP 请求都会创建一个新的 Bean，该作用域仅适用于 WebApplicationContext 环境
session	同一个 HTTP Session 共享一个 Bean，不同的 HTTP Session 使用不同的 Bean。该作用域仅适用于 WebApplicationContext 环境

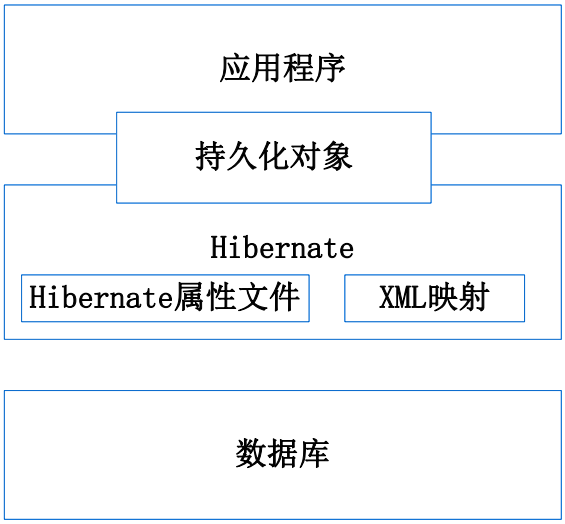
45. Spring 表达式语言（简称 SpEL）：
    1) 是一个支持运行时查询和操作对象图的强大的表达式语言。
    2) 语法类似于 EL：SpEL 使用 #{…} 作为定界符，所有在大框号中的字符都将被认为是 SpEL
    3) SpEL 为 bean 的属性进行动态赋值提供了便利
46. 通过 SpEL 可以实现：
    1) 通过 bean 的 id 对 bean 进行引用
    2) 调用方法以及引用对象中的属性

3) 计算表达式的值
4) 正则表达式的匹配

47. IOC 容器中 Bean 的生命周期：
    1) Spring IOC 容器可以管理 Bean 的生命周期，Spring 允许在 Bean 生命周期的特定点执行定制的任务。
    2) IOC 容器对 Bean 的生命周期的管理过程：
        a) 通过构造器或工厂方法创建 Bean 实例
        b) 为 Bean 的属性设置值和对其他 Bean 的引用
        c) 调用 Bean 的初始化方法
        d) Bean 可以使用了
        e) 当容器关闭时，调用 Bean 的销毁方法
    3) 在 Bean 的声明里设置 init-method 和 destroy-method 属性，为 Bean 指定初始化和销毁方法。
48. 通过注解的方式配置 Bean：在 classpath 中扫描组件
    1) 组件扫描(component scanning)：Spring 能够从 classpath 下自动扫描，侦测和实例化具有特定注解的组件。
    2) 特定组件包括：
        a) @Component：基本注解，标识了一个受 Spring 管理的组件
        b) @Respository: 标识持久层组件
        c) @Service: 标识服务层(业务层)组件
        d) @Controller: 标识表现层组件
49. AOP(Aspect-Oriented Programming，面向切面编程)：是一种新的方法论，是对传统 OOP(Object-Oriented Programming，面向对象编程) 的补充。是一种**横向抽取机制**，取代了传统的纵向继承机制。
50. AspectJ 支持 5 种类型的通知注解：
    1) @Before：前置通知，在方法执行之前执行
    2) @After: 后置通知，在方法执行之后执行
    3) @AfterRunning: 返回通知，在方法返回结果之后执行
    4) @AfterThrowing: 异常通知，在方法抛出异常之后
    5) @Around: 环绕通知，围绕着方法执行
51. Spring 中的事务管理：
    1) 事务管理是企业级应用程序开发中必不可少的技术， 用来**确保数据的完整性和一致性**。
    2) 事务就是一系列的动作，它们被当做一个单独的工作单元. 这些动作要么全部完成，要么全部不起作用。
    3) **声明式事务管理**：大多数情况下比编程式事务管理更好用。它将事务管理代码从业务方法中分离出来，以声明的方式来实现事务管理。事务管理作为一种**横切关注点**，可以通过 AOP 方法模块化。Spring 通过 **Spring AOP 框架**支持声明式事务管理。
52. 事物的四个基本特征是：原子性、一致性、隔离性、持久性
53. ORM 原理：ORM（Object Relational Mapping）是对象到关系的映射，它的作用是在**关系数据库**和**对象**之间做一个自动映射，将数据库中**数据表映射成为对象**，也就是**持久化类**。对关系型数据以对象的形式进行操作，减少应用开发过程中数据持久化的编程任务。



54. Hibernate（**数据持久层框架**）简介：
    1) **配置类(Configuration)**：主要负责管理 Hibernate 的配置信息以及启动 Hibernate，在 Hibernate 运行时配置类（Configuration）会读取一些底层实现的基本信息，其中包括数据库 URL、数据库用户名、数据库用户密码、数据库驱动类和数据库适配器（dialect）。
    2) **会话工厂类（SessionFactory)**：是生成 Session 的工厂，它保存了当前数据库中所有的映射关系，可能只有一个可选的二级数据缓存，并且**它是线程安全的**。但是会话工厂类（SessionFactory）是一个**重量级对象**，它的初始化创建过程会耗费大量的系统资源。
    3) **会话类（Session)**：是 Hibernate 中数据库持久化操作的核心，它将负责 Hibernate 所有的持久化操作，通过它开发人员可以实现数据库基本的增、删、改、查的操作。但会话类（Session）并不是线程安全的，应**注意不要多个线程共享一个 Session**。



55. 持久化类：符合最基本的 **JavaBean** 编码规范，也就是 **POJO（Plain Old Java Object）编程模型**，持久化类中的每个属性都有相应的 set()和 get()方法。
56. Hibernate 的核心配置文件
    1) Hibernate.cfg.xml：用于指定各个参数
    2) Hibernate.hbm.xml 文件：建立表和类的映射关系
57. Hibernate 的三个**核心组件**：连接管理、事物管理、对象/关系映射管理。
58. Hibernate 映射：Hibernate 的核心就是**对象关系映射**，对象（POJO）和关系型数据库之间的映射通常是用 **XML 文档**来实现的。

    示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
```

```
<hibernate-mapping package="com.lyq.model.user">
    <class name="Customer" table="tb_customer">

        <id name="id" column="id">
            <generator class="native"/>
        </id>

        <property name="username" column="username" not-null="true"
        length="50"/>

    </class>
</hibernate-mapping>
```

注解：

1) <DOCTYPE>元素：在所有的 Hibernate 映射文件中都需要定义如上所示的<DOCTYPE>元素，用来获取 DTD 文件。
2) <hibernate-mapping>元素：是映射文件中其他元素的根元素，这个元素中包含一些可选的属性，例如“schema”属性是指明了该文件映射表所在数据库的 schema 名称；“package”属性是指定一个包前缀，如果在<class>元素中没有指定全限定的类名，就将使用“package”属性定义的包前缀作为包名。
3) <class>元素：主要用于指定持久化类和映射的数据库表名。“name”属性需要指定持久化类的全限定的类名（例如“com.mr.User”）；“table”属性就是持久化类所映射的数据库表名。

59. Hibernate 主键策略：

持久化类映射文件<property>元素的常用配置属性

属性名称	说明
increment	用于为long、short、或者int类型生成唯一标识。在集群下不要使用该属性。
identity	由底层数据库生成主键，前提是底层数据库支持自增字段类型
sequence	根据底层数据库的序列生成主键，前提是底层数据库支持序列
hilo	根据高/低算法生成，把特定表的字段作为高位值来源，在默认的情况下选用hibernate_unique_key表的next_hi字段
native	根据底层数据库对自动生成标识符的支持能力选择identity、sequence或hilo
assigned	由程序负责主键的生成，此时持久化类的唯一标识不能声明为private类型
select	通过数据库触发器生成主键
foreign	使用另一个相关联的对象的标识符，通常和<one-to-one>一起使用

60. Hibernate 实例状态：
    1) **瞬时状态（Transient）**：指程序数据保存在内存中，程序退出时，数据就不存在。
    2) **持久化状态（Persistent）**：将程序中数据在瞬时状态和持久状态间转换的机制。（程序数据直接保存成文本文件属于持久化）
    3) **脱管状态（Detached）**
61. Hibernate 初始化类：**Session 对象**是 Hibernate 中数据库持久化操作的**核心**，它将负责 Hibernate 所有的持久化操作，通过它开发人员可以实现数据库基本的增、删、改、查的操作。
    1) 创建一个 Configuration 对象，该对象中保存了 Hibernate 底层运行所需要的信息

和 Hibernate 的映射信息：

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

factory = cfg.buildSessionFactory();

2) 从 SessionFactory 中获取 session 对象：

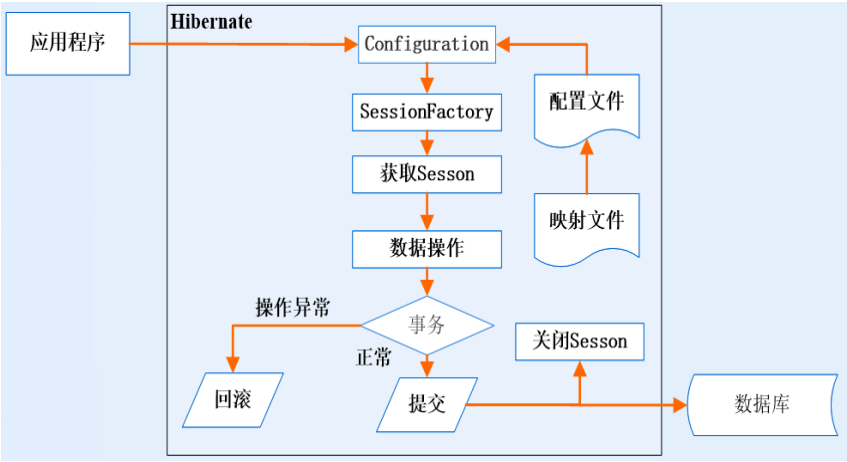
Session session = factory.openSession();

62. Hibernate 框架中的 SessionFactory 类中两个获得 session 的方法

1) openSession()：在同一线程中，需要使用不同的 Session

2) getCurrentSession()：在同一线程中，保证使用同一 Session

63. 保存数据：




```
Session session = null;                          //声明Session对象
Product product = new Product();                 //实例化持久化类
//为持久化类属性赋值
product.setName("Java EE");  //设置产品名称
product.setPrice(79.00);                  //设置产品价格
product.setFactory("东北师大");                  //设置生产商
product.setRemark("无");                  //设置备注
//Hibernate的持久化操作
try {
   session = HibernateInitialize.getSession();   //获取Session
   session.beginTransaction();               //开启事务
   session.save(product);                  //执行数据库添加操作
   session.getTransaction().commit();          //事务提交
} catch (Exception e) {
   session.getTransaction().rollback();       //事务回滚
   System.out.println("数据添加失败");
   e.printStackTrace();
   }finally{
   HibernateInitialize.closeSession();       //关闭Session对象
}
```

64. 查询数据

1) get()方法：如果开发人员不确定数据库中是否有匹配的记录存在，就可以使用 get()方法进行对象装载，因为它会**立刻访问数据库**。如果数据库中没有匹配记录存在，

会返回 null。

示例：Product product = (Product) session.get(Product.class, new Integer("1"));

2) load()方法：load()方法返回对象的代理，只有在**返回对象被调用**的时候，Hibernate 才会发出 **SQL 语句**去查询对象。

示例：Product product = (Product) session.load(Product.class, new Integer("1"));

65. 删除数据：只有对象在持久化状态时才能进行执行 delete()方法，所以在删除数据之前，首先需要将**对象的状态转换为持久化状态**。

示例：

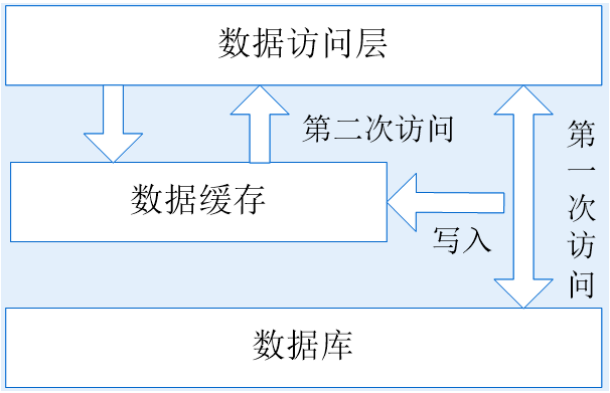
```
Product product = (Product) session.get(Product.class, new Integer("1"));
session.delete(product);
```

66. 修改数据：在 Hibernate 的 Session 的管理中，如果程序对持久化状态的对象做出了修改，当 Session 刷新时，Hibernate 会对实例进行持久化操作。

示例：

```
Product product = (Product) session.get(Product.class, new Integer("1"));
product.setName("Java EE");
product.setRemark("东北师大出品");
session.flush();
```

67. Hibernate 的缓存：



1) 一级缓存的使用：Hibernate 的一级缓存属于 **Session 级缓存**，所以它的生命周期与 Session 是相同的，它随 Session 的创建而创建，随 Session 的销毁而销毁。

2) 二级缓存的使用：Hibernate 的二级缓存将由从属于**一个 SessionFactory 的所有 Session 对象共享**。

```
<hibernate-configuration>
    <session-factory>
        <-- 开启二级缓存 -->
        <property name="hibernate.cache.use_second_level_cache">true</property>
        <-- 配置使用的二级缓存的产品 -->
        <property ame="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
    </session-factory>
</hibernate-configuration>
```

68. 多对一单向关联：

```
<!-- 多对一关联映射 -->
//name：多这一端关联的一那一端的属性的名字
//class：一那一端的属性对应的类名
//column：一那一端在多的一端对应的数据表中的外键的名字
<many-to-one name="factory" class="com.mr.factory.Factory">
```

```
    <!-- 映射的字段 -->
    <column name="factoryid"/>
</many-to-one>
```

69. 多对一双向关联：

```
<set name="products" inverse="true">
    <!-- 定义一对多映射 -->
    <key column="factoryid"/>
    <one-to-many class="com.mr.product.Product"/>
</set>
```

70. 一对一主键关联：

```
<!-- 公民信息字段配置信息 -->
<hibernate-mapping>
    <class name="com.mr.people.People" table="tab_people">
        ……
        <one-to-one name="com.mr.idcard.IDcard"cascade="all"/>
    </class>
</hibernate-mapping>

<!-- 公民身份证字段信息配置信息 -->
<hibernate-mapping>
    <class name="com.mr.idcard.IDcard" table="tab_idcard">
        ……
        <one-to-one name="com.mr.people.People" constrained="true"/>
    </class>
</hibernate-mapping>
```

71. 一对一外键关联：

```
<!-- 公民信息字段配置信息 -->
<hibernate-mapping>
    <class name="com.mr.people.People" table="tab_people">
        ……
        //使用 many-to-one 的方式来映射 1-1 关联关系
        <many-to-one name="idcard" unique="true">
            <column name="card_id"/>
        </many-to-one>
    </class>
</hibernate-mapping>

<!-- 公民身份证字段信息配置信息 -->
<hibernate-mapping>
    <class name="com.mr.idcard.IDcard" table="tab_idcard">
        ……
        <property name="idcard_code" type="string" length="45" not-null="true">
            <column name="IDcard_code"/>
        </property>
```

```
        </class>
    </hibernate-mapping>
```

72. 多对多关联关系：

```
    <!-- User 实体对象 -->
    <hibernate-mapping>
        <class name="com.mr.user.User" table="tab_user">
            ……
            <set name="roles" table="tab_mapping">
                <key column="user_id"></key>
                //使用 many-to-many 指定多对多的关联关系，column 执行 set 集合中的持久化类在中间表的外键列的名称
                <many-to-many class="com.mr.role.Role" column="role_id"/>
            </set>
        </class>
    </hibernate-mapping>

    <!-- Role 实体对象 -->
    <hibernate-mapping>
        <class name="com.mr.role.Role" table="tab_role">
            ...
            <set name="users" table="tab_mapping">
                <key column="role_id"></key>
                <many-to-many class="com.mr.user.User" column="user_id"/>
            </set>
        </class>
    </hibernate-mapping>
```

73. 级联操作：

参 数	说 明
all	所有情况下均采用级联操作
none	默认参数，所有情况下均不采用级联操作
save-update	在执行save-update方法时执行级联操作
delete	在执行delete方法时执行级联操作

```
    <one-to-one name="idcard" class="com.mr.idcard.IDcard" cascade="delete"/>
```

74. HQL 参数绑定机制

1） 利用顺序占位符“？”替代具体参数：

```
String hql = "from Employee emp where emp.sex=?";
```

2） 利用引用占位符“：parameter”替代具体参数：

```
String hql = "from Employee emp where emp.sex=:sex";
```