1. Statement、PreparedStatement和CallableStatement都是接口(interface)。
2. Statement继承自Wrapper、PreparedStatement继承自Statement、CallableStatement继承自PreparedStatement。
    - Statement接口提供了执行语句和获取结果的基本方法；
    - PreparedStatement接口添加了处理 IN 参数的方法；
    - CallableStatement接口添加了处理 OUT 参数的方法。
3. a.Statement:
   普通的不带参的查询SQL；支持批量更新,批量删除;
   b.PreparedStatement:
   可变参数的SQL,编译一次,执行多次,效率高;
   安全性好，有效防止Sql注入等问题;
   支持批量更新,批量删除;
   c.CallableStatement:
   继承自PreparedStatement,支持带参数的SQL操作;
   支持调用存储过程,提供了对输出和输入/输出参数(INOUT)的支持;

Statement每次执行sql语句，数据库都要执行sql语句的编译 ，
最好用于仅执行一次查询并返回结果的情形，效率高于PreparedStatement。

PreparedStatement是预编译的，使用PreparedStatement有几个好处

1. 在执行可变参数的一条SQL时，PreparedStatement比Statement的效率高，因为DBMS预编译一条SQL当然会比多次编译一条SQL的效率要高。
2. 安全性好，有效防止Sql注入等问题。
3. 对于多次重复执行的语句，使用PreparedStament效率会更高一点，并且在这种情况下也比较适合使用batch；
4. 代码的可读性和可维护性。

Spring:

PROPAGATION_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。 PROPAGATION_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。
PROPAGATION_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。
PROPAGATION_REQUIRES_NEW--新建事务，如果当前存在事务，把当前事务挂起。
PROPAGATION_NOT_SUPPORTED--以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。 PROPAGATION_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

Servlet 与 CGI 的比较

和CGI程序一样，Servlet可以响应用户的指令(提交一个FORM等等)，也可以象CGI程序一样，收集用户表单的信息并给予动态反馈(简单的注册信息录入和检查错误)。 然而，Servlet的机制并不仅仅是这样简单的与用户表单进行交互。传统技术中，动态的网页建立和显示都是通过CGI来实现的，但是，有了Servlet,您可以大胆的放弃所有CGI(perl?php?甚至asp!)，利用Servlet代替CGI,进行程序编写。
对比一：当用户浏览器发出一个Http/CGI的请求，或者说** 调用一个CGI程序的时候，服务器端就要新启用一个进程 **(而且是每次都要调用)，调用CGI程序越多(特别是访问量高的时候)，就要消耗系统越多的处理时间，只剩下越来越少的系统资源，对于用户来说，只能是漫长的等待服务器端的返回页面了，这对于电子商务激烈发展的今天来说，不能不说是一种技术上的遗憾。 **而Servlet充分发挥了**

**服务器端的资源并高效的利用。每次调用Servlet时并不是新启用一个进程**，而是在一个Web服务器的进程敏感词享和分离线程，而线程最大的好处在于可以共享一个数据源，使系统资源被有效利用。对比二：传统的CGI程序，不具备平台无关性特征，系统环境发生变化，CGI程序就要瘫痪，而Servlet具备Java的平台无关性，在系统开发过程中保持了系统的可扩展性、高效性。 对比三：传统技术中，一般大都为二层的系统架构，即Web服务器+数据库服务器，导致网站访问量大的时候，无法克服CGI程序与数据库建立连接时速度慢的瓶颈，从而死机、数据库死锁现象频繁发生。而我们的Servlet有连接池的概念，它可以利用多线程的优点，在系统缓存中事先建立好若干与数据库的连接，到时候若想和数据库打交道可以随时跟系统"要"一个连接即可，反应速度可想而知。

Servlet:

Servlet的生命周期分为5个阶段：加载、创建、初始化、处理客户请求、卸载。

(1)加载：容器通过类加载器使用servlet类对应的文件加载servlet

(2)创建：通过调用servlet构造函数创建一个servlet对象

(3)初始化：调用init方法初始化

(4)处理客户请求：每当有一个客户请求，容器会创建一个线程来处理客户请求

(5)卸载：调用destroy方法让servlet自己释放其占用的资源

Struts1 和 Struts2

从action类上分析: 1.Struts1要求Action类继承一个抽象基类。Struts1的一个普遍问题是使用抽象类编程而不是接口。

2. Struts 2 Action类可以实现一个Action接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2提供一个ActionSupport基类去实现常用的接口。Action接口不是必须的，任何有execute标识的POJO对象都可以用作Struts2的Action对象。
   从Servlet 依赖分析:
3. Struts1 Action 依赖于Servlet API ,因为当一个Action被调用时HttpServletRequest 和 HttpServletResponse 被传递给execute方法。
4. Struts 2 Action不依赖于容器，允许Action脱离容器单独被测试。如果需要，Struts2 Action仍然可以访问初始的request和response。但是，其他的元素减少或者消除了直接访问HttpServetRequest 和 HttpServletResponse的必要性。
   从action线程模式分析:
5. Struts1 Action是单例模式并且必须是线程安全的，因为仅有Action的一个实例来处理所有的请求。单例策略限制了Struts1 Action能作的事，并且要在开发时特别小心。Action资源必须是线程安全的或同步的。
6. Struts2 Action对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）

AWT和Swing:

AWT ： 是通过调用操作系统的native方法实现的，所以在Windows系统上的AWT窗口就是Windows的风格，而在Unix系统上的则是XWindow风格。 AWT 中的图形函数与 操作系统 所提供的图形函数之间有着一一对应的关系，我们把它称为peers。 也就是说，当我们利用 AWT 来构件图形用户界面的时候，我们实际上是在利用 操作系统 所提供的图形库。由于不同 操作系统 的图形库所提供的功能是不一样的，在一个平台上存在的功能在另外一个平台上则可能不存在。为了实现Java语言所宣称的"一次编译，到处运行"的概念，AWT 不得不通过牺牲功能来实现其平台无关性，也就是说，AWT 所提供的图形功能是各种通用型操作系统所提供的图形功能的交集。由于AWT 是依靠本地方法来实现其功能

的，我们通常把AWT控件称为重量级控件。

Swing ： 是所谓的Lightweight组件，不是通过native方法来实现的，所以Swing的窗口风格更多样化。但是,Swing里面也有heaveyweight组件。比如JWindow，Dialog,JFrame

Swing是所谓的Lightweight组件，不是通过native方法来实现的，所以Swing的窗口风格更多样化。但是,Swing里面也有heaveyweight组件。比如JWindow，Dialog,JFrame

Swing由纯Java写成，可移植性好，外观在不同平台上相同。所以Swing部件称为轻量级组件（ Swing是由纯JAVA CODE所写的，因此SWING解决了JAVA因窗口类而无法跨平台的问题，使窗口功能也具有跨平台与延展性的特性，而且SWING不需占有太多系统资源，因此称为轻量级组件！！！）

redirect 和 forward:

redirect：请求重定向：客户端行为，本质上为2次请求，地址栏改变，前一次请求对象消失。举例：你去银行办事（forward.jsp），结果告诉你少带了东西，你得先去公安局办(index.html)临时身份证,这时你就会走出银行，自己前往公安局，地址栏变为index.html.

forward：请求转发:服务器行为，地址栏不变。举例：你把钱包落在出租车上，你去警察局（forward.jsp）报案，警察局说钱包落在某某公司的出租车上（index.html)，这时你不用亲自去找某某公司的出租车,警察局让出租车自己给你送来，你只要在警察局等就行。所以地址栏不变，依然为forward.jsp

加载驱动方法

1.Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

2. DriverManager.registerDriver(new com.mysql.jdbc.Driver());

3.System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");

Java中的多线程是一种抢占式的机制，而不是分时机制。抢占式的机制是有多个线程处于可运行状态，但是只有一个线程在运行。 **共同点：** \1. 他们都是在多线程的环境下，都可以在程序的调用处阻塞指定的毫秒数，并返回。 \2. wait()和sleep()都可以通过interrupt()方法 打断线程的暂停状态 ，从而使线程立刻抛出InterruptedException。 如果线程A希望立即结束线程B，则可以对线程B对应的Thread实例调用interrupt方法。如果此刻线程B正在wait/sleep/join，则线程B会立刻抛出InterruptedException，在catch() {} 中直接return即可安全地结束线程。 需要注意的是，InterruptedException是线程自己从内部抛出的，并不是interrupt()方法抛出的。对某一线程调用interrupt()时，如果该线程正在执行普通的代码，那么该线程根本就不会抛出InterruptedException。但是，一旦该线程进入到 wait()/sleep()/join()后，就会立刻抛出InterruptedException 。 **不同点：** 1.每个对象都有一个锁来控制同步访问。Synchronized关键字可以和对象的锁交互，来实现线程的同步。 sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。 2.wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用 3.sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

4.sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。

5.wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

Integer i01=59 的时候，会调用 Integer 的 valueOf 方法，

这个方法就是返回一个 Integer 对象，只是在返回之前，看作了一个判断，判断当前 i 的值是否在 [-128,127] 区别，且 IntegerCache 中是否存在此对象，如果存在，则直接返回引用，否则，创建一个新的对象。

在这里的话，因为程序初次运行，没有 59 ，所以，直接创建了一个新的对象。

int i02=59 ，这是一个基本类型，存储在栈中。

Integer i03 =Integer.*valueOf*(59); 因为 IntegerCache 中已经存在此对象，所以，直接返回引用。

Integer i04 = **new** Integer(59) ； 直接创建一个新的对象。

System. *out *.println(i01== i02); i01 是 Integer 对象， i02 是 int ，这里比较的不是地址，而是值。 Integer 会自动拆箱成 int ，然后进行值的比较。所以，为真。

System. *out *.println(i01== i03); 因为 i03 返回的是 i01 的引用，所以，为真。

System. *out *.println(i03==i04); 因为 i04 是重新创建的对象，所以 i03,i04 是指向不同的对象，因此比较结果为假。

System. *out *.println(i02== i04); 因为 i02 是基本类型，所以此时 i04 会自动拆箱，进行值比较，所以，结果为真。 https://www.nowcoder.com/test/question/done?tid=10252924&qid=15318#summary

**运行时异常：** 都是RuntimeException类及其子类异常，如NullPointerException(空指针异常)、IndexOutOfBoundsException(下标越界异常)等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。

运行时异常的特点是Java编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用try-catch语句捕获它，也没有用throws子句声明抛出它，也会编译通过 **非运行时异常 （编译异常）：** 是RuntimeException以外的异常，类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过。如IOException、SQLException等以及用户自定义的Exception异常，一般情况下不自定义检查异常。

https://www.nowcoder.com/test/question/done?tid=10252924&qid=15319#summary

-Xmx：最大堆大小

-Xms：初始堆大小

-Xmn:年轻代大小

-XXSurvivorRatio：年轻代中Eden区与Survivor区的大小比值

年轻代5120m， Eden：Survivor=3，Survivor区大小=1024m（Survivor区有两个，即将年轻代分为5份，每个Survivor区占一份），总大小为2048m。

-Xms初始堆大小即最小内存值为10240m