# JavaSE

## Java语法基础

#### JVM、JRE和JDK的区别？

Java Virtual Machine(JVM)就是一个用于执行字节码文件的”虚拟计算机”。他也定义了指令集、寄存器集、结构栈、垃圾收集堆、内存区域。JVM负责将Java字节码解释运行，边解释边运行，这样，速度就会受到一定的影响。不同的操作系统有不同的虚拟机。Java 虚拟机机制屏蔽了底层运行平台的差别，实现了“一次编译，随处运行”。 Java虚拟机是实现跨平台的核心机制。

Java Runtime Environment (JRE) 包含：Java虚拟机、库函数、运行应用程序所必须的文件。

Java Development Kit (**JDK**)包含：包含JRE，以及增加编译器和调试器等用于程序开发的文件。

#### 环境变量path和classpath的作用是什么？

path是配置Windows可执行文件的搜索路径，即扩展名为.exe的程序文件所在的目录， 用于指定DOS窗口命令的路径。

classpath是配置class文件所在的目录，用于指定类搜索路径，JVM就是通过它来寻找该类的class类文件的。

#### GC 是什么？为什么要有GC？

GC是垃圾收集的意思，垃圾回收可以有效的防止内存泄露，有效的使用内存。**垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。**

内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。Java程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或Runtime.getRuntime().gc() ，但JVM可以屏蔽掉显示的垃圾回收调用。

#### 一个“.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？

答：可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

#### 变量有什么用？为什么要定义变量？什么时候用？

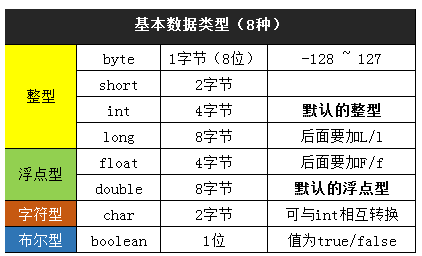
变量的作用：用来存储数据。

为什么要定义变量：用来不断的存放同一类型的常量，并可以重复使用

#### Java的数据类型有哪些？

基本数据类型(Primitive Type)：byte，short，int，long，float，double，char，boolean

引用数据类型(Reference Type)：字符串,数组,枚举,类，接口，注解 ,



**数据类型的相互转换**

正向的过程: 从低字节向高字节可以自动转换 （Java中的自动类型提升 ） byte-> short-> int-> long-> float-> double

三种情况：

int -> float

long -> float

long -> double

特殊情况：char <-> int （相互转换）

#### && 和 & 的区别 ， || 和 | 的区别

(exp1) & (exp2) : exp1,exp2 都要执行

(exp1) | (exp2) : exp1,exp2 都要执行

(exp1) && (exp2) 若exp1 表达式返回false， exp2不执行，直接返回结果false

(exp1) || (exp2) 若exp1 表达式返回true， exp2不执行，直接返回结果true

#### char型变量中能不能存贮一个中文汉字?为什么?

char型变量是用来存储Unicode编码的字符的，unicode编码字符集中包含了汉字，所以，char型变量中当然可以存储汉字啦。不过，如果某个特殊的汉字没有被包含在unicode编码字符集中，那么，这个char型变量中就不能存储这个特殊汉字。补充说明：unicode编码占用两个字节，所以，char类型的变量也是占用两个字节。

#### ++i与i++的区别

**就近原则：**

i++ 先运算在赋值,例如 a=i++,先运算a=i,后运算i=i+1,所以结果是a==1

++i 先赋值在运算,例如 a=++i,先运算i=i+1,后运算a=i,所以结果是a==2

#### short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗?

对于short s1 = 1; s1 = s1 + 1;由于1是int类型，因此s1+1运算结果也是int 型，需要强制转换类型才能赋值给short型。而short s1 = 1; s1 += 1;可以正确编译，因为s1+= 1;相当于s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

#### Java 中的final关键字有哪些用法？

(1)修饰类：表示该类不能被继承，即不能有子类

(2)修饰方法：表示方法不能被重写；

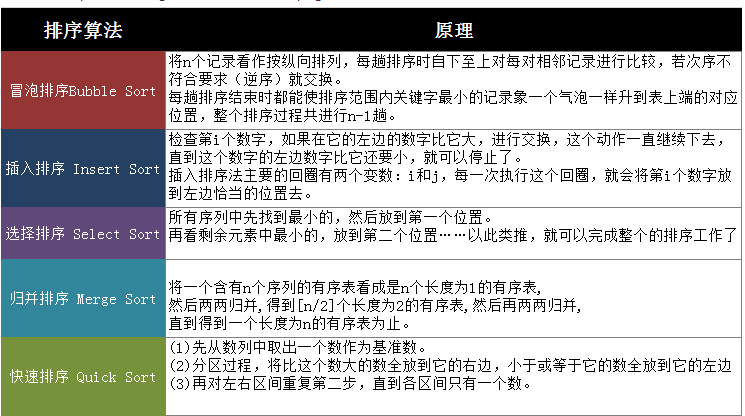
(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

#### swtich是否能作用在byte上，是否能作用在long上，是否能作用在String上？

在Java 5以前，switch(expr)中，expr只能是byte、short、char、int，枚举。

从Java 7开始，expr还可以是字符串（String），但是长整型（long）在目前所有的版本中都是不可以的。

#### 排序都有哪几种方法？请列举。口述用JAVA实现快速排序。



#### String为什么设计为final?

1.为了实现字符串池；

2.为了线程安全；

3.为了实现String可以创建HashCode不可变性；

## 面向对象

#### 你怎么理解面向对象？

根据个人理解，能讲多少是多少

**面对对象就是：**

把数据及对数据的操作方法放在一起，作为一个相互依存的整体——对象。对同类对象抽象出其共性，形成类。类中的大多数数据，只能用本类的方法进行处理。类通过一个简单的外部接口与外界发生关系，对象与对象之间通过消息进行通信。程序流程由用户在使用中决定。

对象即为人对各种具体物体抽象后的一个概念，人们每天都要接触各种各样的对象，如手机就是一个对象。

在面向对象的编程方式中，对象拥有多种特性，如手机有高度、宽度、厚度、颜色、重量等特性，这些特性被称为对象的属性。对象还有很多功能，如手机可以听音乐、打电话、发信息、看电影等工作功能，这些功能被称为对象的方法，实际上这些方法是一种函数。而对象又不是孤立的，是有父子关系的，如手机属于电子产品，电子产品属于物体等，这种父子关系称为对象的继承性。在编程中，把所有的问题看做对象，以人的思维方式解决。这种方式非常人性化，对象实际上就是一组数据的集合，并且数据都已经命名。这些数据根据就是对象的属性，可以被程序访问。对象还包括很多函数，这些函数被称为对象的方法，也可以被程序访问。不过在外部访问对象内的属性或方法，必须先引用对象，然后用点号访问对象的属性和方法

#### 面向对象有什么特征？

（1）封装

概念:是指隐藏对象的属性和实现细节，仅对外提供公共访问方式。

好处：将变化隔离；便于使用；提高重用性；安全性。

封装原则：将不需要对外提供的内容都隐藏起来，把属性都隐藏，提供公共方法对其访问。

（2）继承

好处：a.提高了代码的复用性。

b.让类与类之间产生了关系，提供了多态的前提。

（3）多态

体现：父类引用或者接口的引用指向了自己的子类对象。//Animal a = new Cat();

多态的好处：提高了程序的扩展性。

多态的弊端：当父类引用指向子类对象时，虽然提高了扩展性，但是只能访问父类中具备的方法，不可以访问子类中特有的方法。(前期不能使用后期产生的功能，即访问的局限性)

多态的前提：

a. 必须要有关系，比如继承、或者实现。

b. 通常会有重写操作。

#### Java的访问修饰符有哪些？

public：使用public修饰，对所有类可以访问。

protected：使用protected修饰，对同一包内和所有子类可以访问。

缺省：不使用任何修饰符，在同一包内可以访问。

private：使用private修饰，在同一类内可以访问。

#### 重载（overload）和重写（override）的区别？

方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。

重载发生在一个类中，同名的方法如果有不同的参数列；（参数类型不同、参数个数不同、参数顺序不同）则视为重载；

重写发生在继承关系中，同名的方法有相同的参数列表；

修饰符：子类中的修饰符只能和父类中方法的修饰符相同或者更宽；

返回值：子类中的方法的返回值只能是父类中方法返回值类型的子类或者相同； 对于基本类型必须完全一致；

异常：子类中方法抛出异常只能是父类中方法抛出异常的子类或者相同。 子类中的方法可以不抛出任何异常。

#### 接口和抽象类的区别是什么?（能讲多少讲多少）

**应用方面的区别：**

接口更多的是在系统架构设计方法发挥作用，主要组件之间的通讯协议,为了提高组件之间的通讯安全性、可维护性与降低耦合；

抽象类在代码实现方面发挥作用，可以实现代码的重用。在业务中一系列模块中有若干该类不能自己实现,需要依赖其他组件实现,该功能设计为抽象方法,该类为抽象类.

**语法方面的区别：**



#### Java 支持多继承么?

不支持，Java 不支持多继承。每个类都只能继承一个类，但是可以实现多个接口。

#### Java 中实现多态的机制是什么?

靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，

而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，

也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

#### String，StringBuilder的和StringBuffer的区别？

String: 字符串不可变，适用于字符串内容不经常发生改变的时候

StringBuilder： 字符串可变，适用于字符串内容经常发生改变的时候，适用于单线程（线程不安全），在单线程中，执行效率较高

StringBuffer：字符串可变，适用于字符串内容经常发生改变的时候，适用于多线程（线程安全）

**（讲到此处就可以，以下是理解）**

详细分析：

String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String ，因为每次生成对象都会对系统性能产生影响，特别当内存中无引用对象多了以后， JVM 的 GC 就会开始工作，那速度是一定会相当慢的。

而如果是使用 StringBuffer 类则结果就不一样了，每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用。所以在一般情况下我们推荐使用 StringBuffer ，特别是字符串对象经常改变的情况下。而在某些特别情况下， String 对象的字符串拼接其实是被 JVM 解释成了 StringBuffer 对象的拼接，所以这些时候 String 对象的速度并不会比 StringBuffer 对象慢，而特别是以下的字符串对象生成中， String 效率是远要比 StringBuffer 快的：

String S1 = “This is only a” + “ simple” + “ test”;

StringBuffer Sb = new StringBuilder(“This is only a”).append(“ simple”).append(“ test”);

你会很惊讶的发现，生成 String S1 对象的速度简直太快了，而这个时候 StringBuffer 居然速度上根本一点都不占优势。其实这是 JVM 的一个把戏，在 JVM 眼里，这个

String S1 = “This is only a” + “ simple” + “test”; 其实就是：

String S1 = “This is only a simple test”; 所以当然不需要太多的时间了。但大家这里要注意的是，如果你的字符串是来自另外的 String 对象的话，速度就没那么快了，譬如：

String S2 = “This is only a”;

String S3 = “ simple”;

String S4 = “ test”;

String S1 = S2 +S3 + S4;

这时候 JVM 会规规矩矩的按照原来的方式去做

在大部分情况下 StringBuffer > String

StringBuffer

Java.lang.StringBuffer线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但不能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。

可将字符串缓冲区安全地用于多个线程。可以在必要时对这些方法进行同步，因此任意特定实例上的所有操作就好像是以串行顺序发生的，该顺序与所涉及的每个线程进行的方法调用顺序一致。

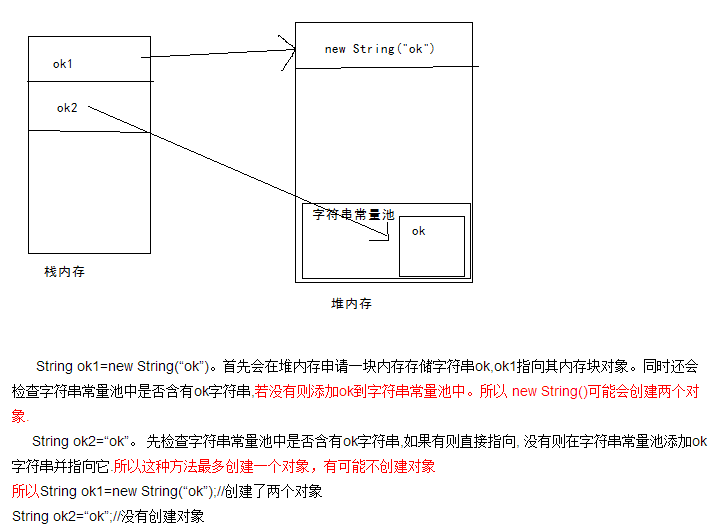
StringBuffer 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串缓冲区中。append 方法始终将这些字符添加到缓冲区的末端；而 insert 方法则在指定的点添加字符。

例如，如果 z 引用一个当前内容是“start”的字符串缓冲区对象，则此方法调用 z.append("le") 会使字符串缓冲区包含“startle”，而 z.insert(4, "le") 将更改字符串缓冲区，使之包含“starlet”。

在大部分情况下 StringBuilder > StringBuffer

java.lang.StringBuilde

java.lang.StringBuilder一个可变的字符序列是5.0新增的。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。两者的方法基本相同.



#### 写出String类常见的5个方法,并详细解释?

|  |  |
| --- | --- |
| **SN(序号)** | **方法描述** |
| 1 | [char charAt(int index)](http://www.runoob.com/java/java-string-charat.html)返回指定索引处的 char 值。 |
| 2 | [int compareTo(Object o)](http://www.runoob.com/java/java-string-compareto.html)把这个字符串和另一个对象比较。 |
| 3 | [int compareTo(String anotherString)](http://www.runoob.com/java/java-string-compareto.html) 按字典顺序比较两个字符串。 |
| 4 | [int compareToIgnoreCase(String str)](http://www.runoob.com/java/java-string-comparetoignorecase.html)按字典顺序比较两个字符串，不考虑大小写。 |
| 5 | [String concat(String str)](http://www.runoob.com/java/java-string-concat.html)将指定字符串连接到此字符串的结尾。 |
| 9 | [boolean endsWith(String suffix)](http://www.runoob.com/java/java-string-endswith.html)测试此字符串是否以指定的后缀结束。 |
| 10 | [boolean equals(Object anObject)](http://www.runoob.com/java/java-string-equals.html)将此字符串与指定的对象比较。 |
| 11 | [boolean equalsIgnoreCase(String anotherString)](http://www.runoob.com/java/java-string-equalsignorecase.html)将此 String 与另一个 String 比较，不考虑大小写。 |
| 12 | [byte[] getBytes()](http://www.runoob.com/java/java-string-getbytes.html)使用平台的默认字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。 |
| 13 | [byte[] getBytes(String charsetName)](http://www.runoob.com/java/java-string-getbytes.html)使用指定的字符集将此 String 编码为 byte 序列，并将结果存储到一个新的 byte 数组中。 |
| 14 | [void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)](http://www.runoob.com/java/java-string-getchars.html)将字符从此字符串复制到目标字符数组。 |
| 15 | [int hashCode()](http://www.runoob.com/java/java-string-hashcode.html)返回此字符串的哈希码。 |
| 16 | [int indexOf(int ch)](http://www.runoob.com/java/java-string-indexof.html)返回指定字符在此字符串中第一次出现处的索引。 |
| 17 | [int indexOf(int ch, int fromIndex)](http://www.runoob.com/java/java-string-indexof.html)返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索。 |
| 18 | [int indexOf(String str)](http://www.runoob.com/java/java-string-indexof.html)返回指定子字符串在此字符串中第一次出现处的索引。 |
| 19 | [int indexOf(String str, int fromIndex)](http://www.runoob.com/java/java-string-indexof.html)返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始。 |
| 25 | [int length()](http://www.runoob.com/java/java-string-length.html)返回此字符串的长度。 |
| 29 | [String replace(char oldChar, char newChar)](http://www.runoob.com/java/java-string-replace.html)返回一个新的字符串，它是通过用 newChar 替换此字符串中出现的所有 oldChar 得到的。 |
| 30 | [String replaceAll(String regex, String replacement)](http://www.runoob.com/java/java-string-replaceall.html)使用给定的 replacement 替换此字符串所有匹配给定的正则表达式的子字符串。 |
| 31 | [String replaceFirst(String regex, String replacement)](http://www.runoob.com/java/java-string-replacefirst.html)使用给定的 replacement 替换此字符串匹配给定的正则表达式的第一个子字符串。 |
| 32 | [String[] split(String regex)](http://www.runoob.com/java/java-string-split.html)根据给定正则表达式的匹配拆分此字符串。 |
| 33 | [String[] split(String regex, int limit)](http://www.runoob.com/java/java-string-split.html)根据匹配给定的正则表达式来拆分此字符串。 |
| 34 | [boolean startsWith(String prefix)](http://www.runoob.com/java/java-string-startswith.html)测试此字符串是否以指定的前缀开始。 |
| 35 | [boolean startsWith(String prefix, int toffset)](http://www.runoob.com/java/java-string-startswith.html)测试此字符串从指定索引开始的子字符串是否以指定前缀开始。 |
| 36 | [CharSequence subSequence(int beginIndex, int endIndex)](http://www.runoob.com/java/java-string-subsequence.html)返回一个新的字符序列，它是此序列的一个子序列。 |
| 37 | [String substring(int beginIndex)](http://www.runoob.com/java/java-string-substring.html)返回一个新的字符串，它是此字符串的一个子字符串。 |
| 38 | [String substring(int beginIndex, int endIndex)](http://www.runoob.com/java/java-string-substring.html)返回一个新字符串，它是此字符串的一个子字符串。 |
| 39 | [char[] toCharArray()](http://www.runoob.com/java/java-string-tochararray.html)将此字符串转换为一个新的字符数组。 |
| 40 | [String toLowerCase()](http://www.runoob.com/java/java-string-tolowercase.html)使用默认语言环境的规则将此 String 中的所有字符都转换为小写。 |
| 41 | [String toLowerCase(Locale locale)](http://www.runoob.com/java/java-string-tolowercase.html)使用给定 Locale 的规则将此 String 中的所有字符都转换为小写。 |
| 43 | [String toUpperCase()](http://www.runoob.com/java/java-string-touppercase.html)使用默认语言环境的规则将此 String 中的所有字符都转换为大写。 |
| 44 | [String toUpperCase(Locale locale)](http://www.runoob.com/java/java-string-touppercase.html)使用给定 Locale 的规则将此 String 中的所有字符都转换为大写。 |
| 45 | [String trim()](http://www.runoob.com/java/java-string-trim.html)返回字符串的副本，忽略前导空白和尾部空白。 |
| 46 | [static String valueOf(primitive data type x)](http://www.runoob.com/java/java-string-valueof.html)返回给定data type类型x参数的字符串表示形式。 |

#### Java中的泛型是什么 ? 使用泛型的好处是什么?

泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。

泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，提高代码的重用性。

#### Error和Exception有什么区别?

Error和Exception都是继承于Throwable。

**Error**一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢出等。对于这类错误，Java编译器不去检查他们。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和预防，遇到这样的错误，建议让程序终止。

**Exception**类表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。

#### try{}里有一个return语句，那么紧跟在这个try后的finally{}里的code会不会被执行，什么时候被执行，在return前还是后?

* 1. finally: 语法上无论是否有异常都会执，用于释放资源、节省内存，
  2. return : 方法结束
  3. 先执行return ,再执行finally

#### 在finally中，修改return 的返回值变量，最后返回值究竟有没有发生改变？

若返回值类型是：基本数据类型，finally 中修改返回值变量，返回值不会改变【值传递】

若返回值类型是：引用数据类型，finally 中修改返回值变量，返回值会改变【引用传递】

#### 列出一些你常见的运行时异常？（至少五个）

NullPointerException （空指针异常）

ClassCastException （类转换异常）

ArithmeticException (算术异常)

IndexOutOfBoundsException  （数组下标越界异常）

SecurityException (安全异常)

IllegalArgumentException  （非法参数异常）

#### 15）final, finally, finalize 的区别?

**final：修饰符有三种用法：**

(1)修饰类：表示该类不能被继承，即不能有子类

(2)修饰方法：表示方法不能被重写；

(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

**finally：用在异常结构中**

通常放在try…catch的后面构造总是执行代码块，可以将释放外部资源的代码写在finally块中，即finally的作用是释放资源，节省内存。

**finalize：Object类中GC相关的方法**

Object类中的方法，Java中允许使用finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写finalize() 方法可以整理系统资源或者执行其他清理工作。

#### ==与equals的区别

（1）基本数据类型比较，用双等号（==）,比较的是他们的值。

（2）引用数据类型比较，

  a. 使用==比较，比较的是他们在内存中的存放地址，所以，除非是同一个new出来的对象，比较后的结果为true，否则结果为false。  
  b. 使用equals比较，若没有重写Object类的equals方法，比较还是内存地址（==理解）；

若重写Object类的equals方法，equals比较的是堆中内容是否相等，即两个对象的内容是否相同。

#### 自动装箱与拆箱

装箱：将基本类型---转换成---对应的包装类；

拆箱：将包装类型---转换成---基本数据类型；  
Java使用自动装箱和拆箱机制，节省了常用数值的内存开销和创建对象的开销，提高了效率，由编译器来完成，编译器会在编译期根据语法决定是否进行装箱和拆箱动作。

#### Java中按值传递和引用传递区别

**按值传递**：值传递是指在调用函数时将实际参数复制一份传递到函数中，这样在函数中如果对参数进行修改，将不会影响到实际参数。

**按引用传递**：引用传递就是直接把内存地址传过去，操作的其实都是源数据。具体的数据类型就比较多了，比如Object，二维数组，List，Map等除了基本类型的参数都是引用传递。

#### 什么是反射API？它是如何实现的？

反射是指在运行时能查看一个类的状态及特征，并能进行动态管理的功能。这些功能是通过一些内建类的反射API提供的，比如Class,Method,Field, Constructors等。使用的例子：使用Java反射API的getName方法可以获取到类名。

#### 类加载器工作机制

装载：将Java二进制代码导入jvm中，生成Class文件。  
连接：

a）校验：检查载入Class文件数据的正确性

b）准备：给类的静态变量分配存储空间

c）解析：将符号引用转成直接引用

初始化：对类的静态变量，静态方法和静态代码块执行初始化工作。  
双亲委派模型：类加载器收到类加载请求，首先将请求委派给父类加载器完成 用户自定义加载器->应用程序加载器->扩展类加载器->启动类加载器。

## 集合

**集合框架：**用于存储数据的容器

1：对象封装数据，对象多了也需要存储。【集合用于存储对象】

2：对象的个数确定可以使用数组，但是不确定怎么办？可以用集合。【因为集合是可变长度的】

#### 数组和集合的区别？

长度： 数组是固定长度的；集合可变长度的。

存储类型： 数组可以存储基本数据类型，也可以存储引用数据类型；集合只能存储引用数据类型。

存储类型种类：数组存储的元素必须是同一个数据类型；集合存储的对象可以是不同数据类型。

#### 请你列出你常用的集合框架中的类与接口（Java集合框架的基础接口有哪些？）

Collection祖先接口。一个集合代表一组对象，这些对象即为它的元素。Java平台不提供这个接口任何直接的实现。

List是一个可以包含重复元素的集合。你可以通过它的索引来访问任何元素。List更像长度动态变换的数组。常用实现子类:ArrayList,LinkedList

Set是一个不能包含重复元素的集合。这个接口对数学集合抽象进行建模，被用来代表集合，就如一副牌。常用实现子类：HashSet,TreeSet

Map是一个将key映射到value的对象.一个Map不能包含重复的key：每个key最多只能映射一个value。Map的key可以插入一次为null的值

一些其它的接口有Queue、Dequeue、SortedSet、SortedMap和ListIterator。

#### ArrayList、LinkedList 、Vector的区别？

(1) ArrayList本质是数组 ,线程不同步，ArrayList替代了Vector，查询元素的速度非常快,默认大小10，1.5倍长度扩容

- 频繁的添加操作：可能会造成扩容，内存浪费与大量元素复制移位，随着数据量增多，时空复杂度会增大；

- 频繁的删除操作：可能会出现大量元素的移位

- 频繁的取元素操作：能直接根据数组下标定位元素，因为数据在数组中的物理地址是连续的；

- 【所以：ArrayList 适用于元素的随机访问】

(2) LinkedList 本质是链表,线程不同步，增删元素的速度非常快

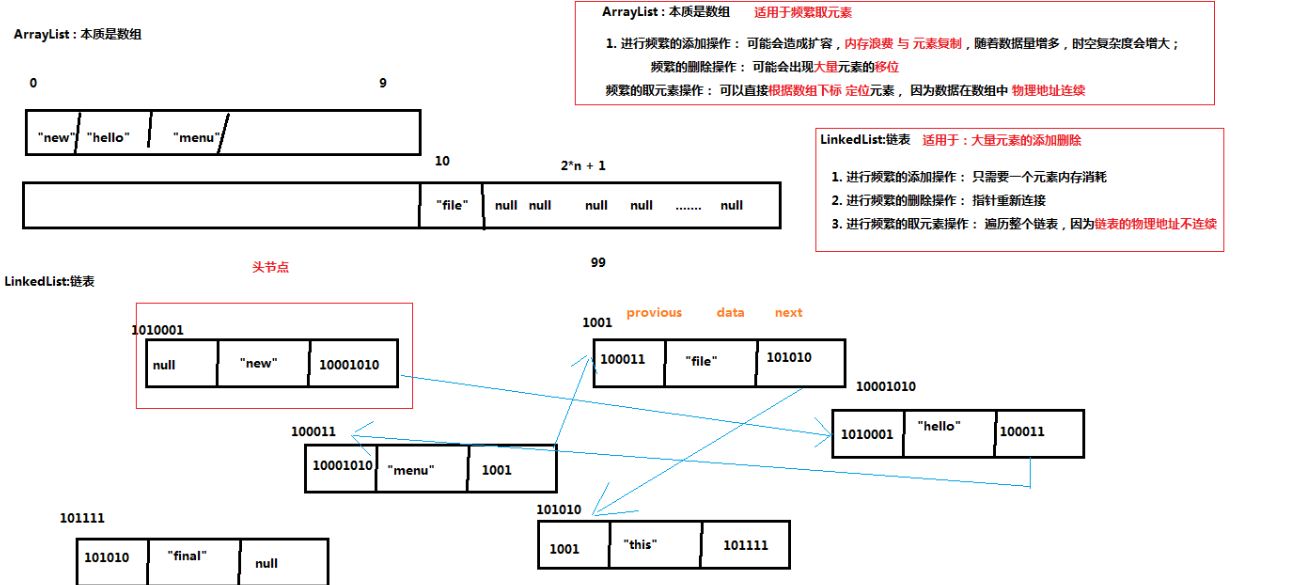
- 频繁的添加操作：只需要一个元素内存消耗

- 频繁的删除操作：只需要元素指针重新连接

- 频繁的取元素操作：需要遍历整个链表，因为链表的物理地址是不连续的

- 【所以：LinkedList适用于元素的大量添加与删除】

(3) Vector：底层的数据结构就是数组，线程同步，Vector无论查询和增删都巨慢。默认大小10，2倍长度扩容。

****

#### HashSet、TreeSet的区别？

HashSet 是由哈希表实现的,HashSet是散列存放，可以放入null，但只能放入一个null，两者中的值都不能重复，唯一性依赖于重写**equals**、**hashcode**方法，底层原理通过**HashMap的key**实现；

TreeSet 是由二叉树实现的，Treeset是有序存放，依赖于自然顺序或自定义排序实现，不允许放入null值。 唯一性：依赖于重写自然排序或自定义排序**compareTo<E>泛型**方法,根据比较的返回值是否为0；

TreeSet判断两个对象不相等的方式是通过排序实现

（1）自然排序

自然排序要实现Comparable接口，该接口里定义了一个compareTo(Object obj)方法，该方法返回一个整数值，实现了该接口的对象就可以比较大小。

obj1.compareTo(obj2)，若返回0，则说明被比较的两个对象相等，若返回一个正数，则表明obj1大于obj2，若返回负数，则表明obj1小于obj2。

（2）自定义排序

自然排序是根据集合元素的大小，以升序排列，如果要定制排序，应该使用Comparator接口，实现 int compare(T o1,T o2)方法。

#### HashTable、HashMap的区别？

**继承的父类不同**：Hashtable继承自Dictionary类，而HashMap继承自AbstractMap类，但二者都实现了Map接口

**线程安全性不同**：Hashtable是线程安全的，它的每个方法中都加入了Synchronize同步关键字。HashMap不是线程安全的，在多线程并发的环境下，可能会产生死锁等问题。

**key和value是否允许null**：  Hashtable中，key和value都不允许出现null值，HashMap中，key、value允许一次为空；

**内部遍历不同**： Hashtable、HashMap都使用了 Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式

**hash值不同：**HashTable直接使用对象的hashCode;而HashMap重新计算hash值。

**初始容量大小和每次扩充容量大小的不同 ：** Hashtable默认的初始大小为11，之后每次扩充，容量变为原来的2n+1。HashMap默认的初始化大小为16。之后每次扩充，容量变为原来的2倍。

#### HashMap工作原理？

HashMap底层由数组+链表组成的，数组是HashMap的主体，链表则是主要为了解决哈希冲突而存在的。如果定位到的数组位置不含链表（当前entry的next指向null）,那么对于查找，添加等操作很快，仅需一次寻址即可；如果定位到的数组包含链表，对于添加操作，其时间复杂度为O(n)，首先遍历链表，存在即覆盖，否则新增；对于查找操作来讲，仍需遍历链表，然后通过key对象的equals方法逐一比对查找。所以，性能考虑，HashMap中的链表出现越少，性能才会越好。

#### Iterator是什么？

Iterator提供遍历任何Collection集合的接口。可以从Collection中使用迭代器方法来获取迭代器实例。迭代器取代了Java集合框架中的Enumeration。迭代器允许调用者在迭代过程中移除元素

#### 为什么要重写equals和hashcode方法？

先通过hashcode方法返回的值，直接定位元素应该存储的物理位置，位置上没有元素则存放，若有元素，再比较对象的equals方法，从而大大减少equals方法调用，提高比较效率。

#### Collection接口的remove()方法和Iterator接口的remove()方法区别？

①性能方面

    Collection的remove方法必须首先找出要被删除的项，找到该项的位置采用的是单链表结构查询，单链表查询效率比较低，需要从集合中一个一个遍历才能找到该对象；

    Iterator的remove方法结合next()方法使用，比如集合中每隔一项删除一项，Iterator的remove()效率更高

②容错方面

    在使用Iterator遍历时，如果使用Collection的remove则会报异常，会出现ConcurrentModificationException，因为集合中对象的个数会改变而Iterator 内部对象的个数不会，不一致则会出现该异常；

在使用Iterator遍历时，不会报错，因为iterator内部的对象个数和原来集合中对象的个数会保持一致

#### Array与ArrayList的区别？

①Array是Java中的数组：在定义一个数组的时候，必须指定这个数组的数据类型及数组的大小，也就是说数组中存放的元素个数固定并且类型一样。

②ArrayList是动态数组：也就是数组的复杂版本，它可以动态的添加和删除元素，被称为”集合“，在不使用泛型的情况下，这个list是可以添加进不同类型的元素的，而且arraylist是可以不用指定长度的。在使用泛型时，我们就只能添加一种类型的数据了。

#### 数组>>List,List>>数组

①数组转list，可以使用Arrays.asList(数组);

②List转数组，使用list.toArray();

## 多线程

#### 我们如何创建一个线程？

一是实现Runnable接口，传递给Thread的构造函数，创建一个Thread对象；

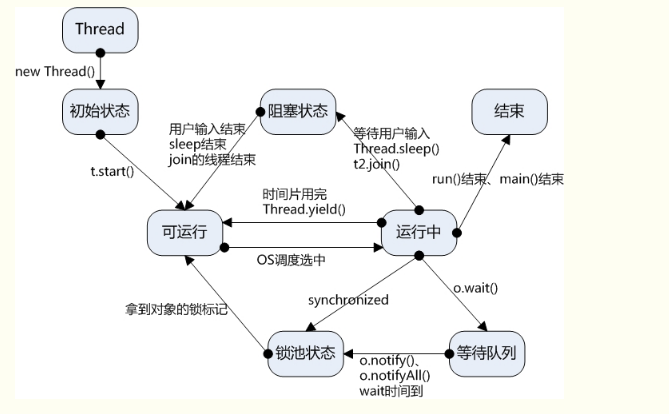
二是直接继承Thread类。

三是通过 Callable 和 Future 创建线程。

#### 启动一个线程是用run()还是start()？

启动线程肯定要用start()方法。当用start()开始一个线程后，线程就进入就绪状态,这意味着它可以由JVM调度并执行。这并不意味着线程就会立即运行。当cpu分配给它时间时，才开始执行run()方法(如果有的话)。而run()方法是你必须重写的. run()方法中包含的是线程的主体。

#### 线程的生命周期？



#### 线程的几种状态？

**初始状态:** 使用 new 关键字和 Thread 类或其子类建立一个线程对象后，该线程对象就处于新建状态。它保持这个状态直到程序 start() 这个线程。

**就绪状态:** 当线程对象调用了start()方法之后，该线程就进入就绪状态。就绪状态的线程处于就绪队列中，要等待JVM里线程调度器的调度。

**运行状态:** 如果就绪状态的线程获取 CPU 资源，就可以执行 run()，此时线程便处于运行状态。处于运行状态的线程最为复杂，它可以变为阻塞状态、就绪状态和死亡状态。

**阻塞状态:** 如果一个线程执行了sleep（睡眠）、suspend（挂起）等方法，失去所占用资源之后，该线程就从运行状态进入阻塞状态。在睡眠时间已到或获得设备资源后可以重新进入就绪状态。可以分为三种：

等待阻塞：运行状态中的线程执行 wait() 方法，使线程进入到等待阻塞状态。

同步阻塞：线程在获取 synchronized 同步锁失败(因为同步锁被其他线程占用)。

其他阻塞：通过调用线程的 sleep() 或 join() 发出了 I/O 请求时，线程就会进入到阻塞状态。当sleep() 状态超时，join() 等待线程终止或超时，或者 I/O 处理完毕，线程重新转入就绪状态。

**死亡状态:** 一个运行状态的线程完成任务或者其他终止条件发生时，该线程就切换到终止状态。

#### start()和run()的区别？

**t2.start():** 表示重新开启一个线程,并且运行run方法,不必等待其他线程运行完，只要得到cpu资源就可以运行该线程。

**t2.run():** 只是调用了一个普通方法，并没有启动另一个线程，程序还是会按照顺序执行相应的代码。

#### sleep()和wait()的区别？

sleep()方法，我们首先要知道该方法是属于Thread类中的。

wait()方法，则是属于Object类中的。

sleep()方法导致了程序暂停执行指定的时间，让出cpu给其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。在调用sleep()方法的过程中，线程不会释放对象锁。而当调用wait()方法的时候，线程会放弃对象锁，进入等待队列，只有此对象调用notify()、notityAll()方法或等待时间结束后本线程才进入锁池状态，然后获取对象锁进入就绪状态。

#### sleep()和yield()的区别？

① sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；

② 线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；

③ sleep()方法声明抛出中断异常（InterruptedException），而yield()方法没有声明任何异常；

④ sleep()方法比yield()方法（跟操作系统CPU调度相关）具有更好的可移植性。

#### [Thread类的常用方法](https://www.cnblogs.com/qin-derella/p/6588421.html)？

void start()  启动一个线程

void run() 线程的主体

static void sleep(long millis):  在指定的毫秒数内让当前正在执行的线程休眠（暂停执行）  
void join(): 等待线程终止。

static void yield(): 暂停当前正在执行的线程对象，执行优先级相同或更高的其他线程。

String getName() 返回该线程的名称。

void setName(String name)　　改变线程名称，使之与参数 name 相同。

int getPriority() 　　返回线程的优先级。

void setPriority(int newPriority) 　　更改线程的优先级。

boolean isDaemon() 　　测试该线程是否为守护线程。

void setDaemon(boolean on)　　将该线程标记为守护线程或用户线程。

void interrupt()　　中断线程。

从Object类继承来的方法　　void notify()     void wait()

#### 线程同步的方法？

wait():使一个线程处于等待状态，并且释放所持有的对象的锁。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException中断异常。

notify():唤醒一个处于等待状态的线程，进入锁池状态，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。

notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

#### 多线程的好处？

多个线程间并发执行可以提高程序的效率，CPU不会因为某个线程需要等待资源而进入空闲状态。多个线程共享堆内存(heap memory)，因此创建多个线程去执行一些任务会比创建多个进程更好。

#### 同步和异步有何异同?

如果数据在线程间共享，例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。

当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

#### 什么是同步?

在非同步保护的多线程程序里面,一个线程正在修改一个共享变量的时候,可能有另一个线程也在使用或者更新它的值，造成了数据污染。

同步用来控制共享资源在多个线程间的访问,以保证同一时间内只有一个线程能访问到这个资源,所以同步有效的避免了脏数据的产生。

#### 线程不安全是什么？线程安全是什么？

**线程不安全**:不提供数据访问保护，有可能出现多个线程先后更改数据造成所得到的数据是脏数据。（StringBuilder,ArrayList，LinkedList，HashMap,等）

**线程安全**:多线程访问时，采用了加锁机制，当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问直到该线程读取完，其他线程才可使用，避免出现脏数据或者数据污染。(StringBuffer ,Vector,Hashtable,)

#### 锁的机制？

**悲观锁**(Pessimistic Lock), 顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

**乐观锁**(Optimistic Lock), 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

**活锁：**是指线程1可以使用资源，但它很礼貌，让其他线程先使用资源，线程2也可以使用资源，但它很绅士，也让其他线程先使用资源。这样你让我，我让你，最后两个线程都无法使用资源。

**死锁：**是指两个或两个以上的进程（或线程）在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。死锁发生的四个条件

互斥条件：线程对资源的访问是排他性的，如果一个线程对占用了某资源，那么其他线程必须处于等待状态，直到资源被释放。

请求和保持条件：线程T1至少已经保持了一个资源R1占用,但又提出对另一个资源R2请求，而此时，资源R2被其他线程T2占用，于是该线程T1也必须等待，但又对自己保持的资源R1不释放。

不剥夺条件：线程已获得的资源，在未使用完之前，不能被其他线程剥夺，只能在使用完以后由自己释放。

环路等待条件：在死锁发生时，必然存在一个“进程-资源环形链”，即：{p0,p1,p2,...pn},进程p0（或线程）等待p1占用的资源，p1等待p2占用的资源，pn等待p0占用的资源。（最直观的理解是，p0等待p1占用的资源，而p1而在等待p0占用的资源，于是两个进程就相互等待）

#### 什么是线程池？

创建线程需要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短 的任务的程序的可扩展线程池）。

根本区别：进程是操作系统资源分配的基本单位，而线程是任务调度和执行的基本单位

#### 线程和进程的区别？

开销方面：每个进程都有独立的代码和数据空间（程序上下文），程序之间的切换会有较大的开销；线程可以看做轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器（PC），线程之间切换的开销小。

所处环境：在操作系统中能同时运行多个进程（程序）；而在同一个进程（程序）中有多个线程同时执行（通过CPU调度，在每个时间片中只有一个线程执行）

内存分配方面：系统在运行的时候会为每个进程分配不同的内存空间；而对线程而言，除了CPU外，系统不会为线程分配内存（线程所使用的资源来自其所属进程的资源），线程组之间只能共享资源。

包含关系：没有线程的进程可以看做是单线程的，如果一个进程内有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的；线程是进程的一部分，所以线程也被称为轻权进程或者轻量级进程。

## IO流

#### 比特(Bit),字节(Byte),字符(Char)的区别？

bit(位)最小的二进制单位 ，是计算机的操作部分 取值0或者1

byte(字节)是计算机操作数据的最小单位由8位bit组成 取值（-128-127）

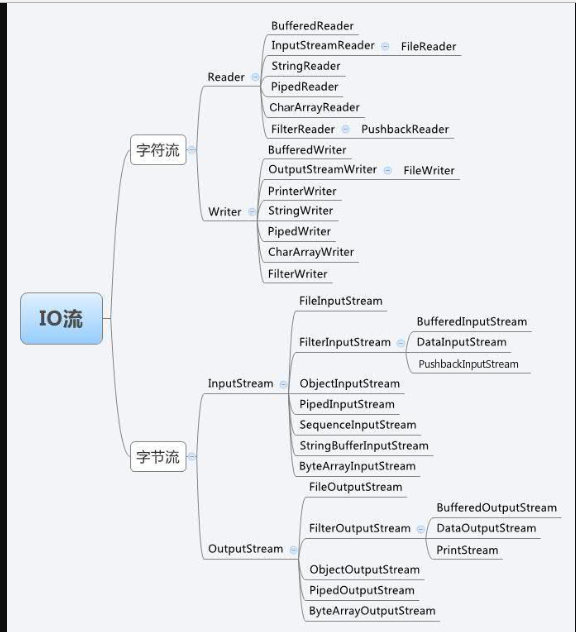
char(字符)是用户的可读写的最小单位，在Java里面由16位bit组成取值（0-65535）

#### 讲讲IO里面的常见类，字节流、字符流、接口、实现类、方法阻塞。

输入流就是从外部文件输入到内存，输出流主要是从内存输出到文件。

IO里面常见的类，第一印象就只知道IO流中有很多类，IO流主要分为字符流和字节流。字符流中有抽象类InputStream和OutputStream，它们的子类FileInputStream，FileOutputStream,BufferedOutputStream等。字符流BufferedReader和Writer等。都实现了Closeable, Flushable, Appendable这些接口。程序中的输入输出都是以流的形式保存的，流中保存的实际上全都是字节文件。

java中的阻塞式方法是指在程序调用改方法时，必须等待输入数据可用或者检测到输入结束或者抛出异常，否则程序会一直停留在该语句上，不会执行下面的语句。比如read()和readLine()方法。



#### 字符流和字节流有什么区别？

要把一片二进制数据逐一输出到某个设备中，或者从某个设备中逐一读取一片二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为IO流，对应的抽象类为OutputStream和InputStream ，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

在应用中，经常要完全是字符的一段文本输出去或读进来，用字节流可以吗？计算机中的一切最终都是二进制的字节形式存在。对于“中国”这些字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，人家专门提供了字符流的包装类。

底层设备永远只接受字节数据，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向IO设备写入或读取字符串提供了一些方便。

#### Java中有几种类型的流？JDK为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？

#### 讲讲NIO。

Java NIO(New IO)是一个可以替代标准Java IO API的IO API（从Java 1.4开始)，NIO提供了与标准IO不同的IO工作方式。

**Java NIO: Channels and Buffers（通道和缓冲区）**标准的IO基于字节流和字符流进行操作的，而NIO是基于通道（Channel）和缓冲区（Buffer）进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。

**Java NIO: Non-blocking IO（非阻塞IO）**Java NIO可以让你非阻塞的使用IO，例如：当线程从通道读取数据到缓冲区时，线程还是可以进行其他事情。当数据被写入到缓冲区时，线程可以继续处理它。从缓冲区写入通道也类似。

**Java NIO: Selectors（选择器）**选择器用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个的线程可以监听多个数据通道。

#### String 编码UTF-8 和GBK的区别?

GBK的文字编码是双字节来表示的，即不论中、英文字符均使用双字节来表示，只不过为区分中文，将其最高位都定成1。至于UTF－8编码则是用以解决国际上字符的多字节编码，它对英文使用8位（即一个字节），中文使用24位（三个字节）来编码。对于英文字符较多的论坛则用UTF－8节省空间。

GBK包含全部中文字符；

UTF-8则包含全世界所有国家需要用到的字符。

GBK是在国家标准GB2312基础上扩容后兼容GB2312的标准；（好像还不是国家标准）

UTF-8编码的文字可以在各国各种支持UTF8字符集的浏览器上显示。

比如，如果是UTF8编码，则在外国人的英文IE上也能显示中文，而无需他们下载IE的中文语言支持包。

所以，对于英文比较多的论坛 ，使用GBK则每个字符占用2个字节，而使用UTF－8英文却只占一个字节。

#### 什么时候使用字节流、什么时候使用字符流?

字节流在操作时本身不会用到缓冲区（内存），是文件本身直接操作的，而字符流在操作时使用了缓冲区，通过缓冲区再操作文件

# 数据库

#### DML ，DDL，DCL是什么？

DML（Data Manipulation Language）数据操纵语言：如：SELECT、UPDATE、INSERT、DELETE。 主要用来对数据库的数据进行一些操作。

DDL（data definition language）数据库定义语言：如：CREATE、ALTER、DROP等。DDL主要是用在定义或改变表的结构，数据类型，表之间的链接和约束等初始化工作上。

DCL（Data Control Language）数据库控制语言：是用来设置或更改数据库用户或角色权限的语句，包括（grant,deny,revoke等）语句。

#### SQL是什么？

SQL(Structured Query Language)结构化查询语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统；同时也是数据库脚本文件的扩展名。

#### 简述 left join、right join 以及inner join 的区别

left join(左联接) 返回包括左表中的所有记录和右表中联结字段相等的记录

right join(右联接) 返回包括右表中的所有记录和左表中联结字段相等的记录

inner join(等值连接) 只返回两个表中联结字段相等的行

#### 简单说一说drop、delete与truncate的区别?

SQL中的drop、delete、truncate都表示删除，但是三者有一些差别

delete和truncate只删除表的数据不删除表的结构

速度,一般来说: drop> truncate >delete

delete语句是dml,这个操作会放到rollback segement中,事务提交之后才生效;  
如果有相应的trigger,执行的时候将被触发. truncate,drop是ddl, 操作立即生效,原数据不放到rollback segment中,不能回滚. 操作不触发trigger.

#### drop、delete与truncate分别在什么场景之下使用？

不再需要一张表的时候，用drop

想删除部分数据行时候，用delete，并且带上where子句

保留表而删除所有数据的时候用truncate

#### 请说明数据库主键、外键的作用?

主键保证了数据的唯一性，外键保证了数据的完整性。

主键是能确定一条记录的唯一标识，比如，一条记录包括身份正号，姓名，年龄。身份证号是唯一能确定你这个人的，其他都可能有重复，所以，身份证号是主键。

外键用于与另一张表的关联。是能确定另一张表记录的字段，用于保持数据的一致性。比如，A表中的一个字段，是B表的主键，那他就可以是A表的外键。

#### 建立索引的好处与坏处？

索引就是加快检索表中数据的方法，数据库的索引类似于书籍的索引，在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息；在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库；索引的实现通常使用B树及其变种B+树。

**MySQL**数据库几个基本的索引类型：普通索引、唯一索引、主键索引、全文索引

* 索引加快数据库的检索速度
* 索引降低了插入、删除、修改等维护任务的速度
* 唯一索引可以确保每一行数据的唯一性
* 通过使用索引，可以在查询的过程中使用优化隐藏器，提高系统的性能
* 索引需要占物理和数据空间

#### 使用索引查询一定能提高查询的性能吗？

通常,通过索引查询数据比全表扫描要快.但是我们也必须注意到它的代价.索引需要空间来存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时,索引本身也会被修改. 这意味着每条记录的INSERT,DELETE,UPDATE将为此多付出4,5 次的磁盘I/O. 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.使用索引查询不一定能提高查询性能,索引范围查询(INDEX RANGE SCAN)适用于两种情况:

* 基于一个范围的检索,一般查询返回结果集小于表中记录数的30%
* 基于非唯一性索引的检索

#### 请简述说明是事务？事务有哪些属性？

事务，是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性。

.事务四大特性   
原子性，要么执行，要么不执行   
隔离性，所有操作全部执行完以前，其它会话不能看到过程   
一致性，事务前后，数据总额一致   
持久性，一旦事务提交，对数据的改变就是永久的

#### 视图的作用，视图可以更改么？

视图是虚拟的表，与包含数据的表不一样，视图只包含使用时动态检索数据的查询；不包含任何列或数据。使用视图可以简化复杂的sql操作，隐藏具体的细节，保护数据；视图创建后，可以使用与表相同的方式利用它们。  
视图不能被索引，也不能有关联的触发器或默认值，如果视图本身内有order by 则对视图再次order by将被覆盖。  
创建视图：create view XXX as XXXXXXXXXXXXXX;  
对于某些视图比如未使用联结子查询分组聚集函数Distinct Union等，是可以对其更新的，对视图的更新将对基表进行更新；但是视图主要用于简化检索，保护数据，并不用于更新，而且大部分视图都不可以更新。

#### 什么是存储过程？

存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以多次调用。如果某次操作需要执行多次SQL，使用存储过程比单纯SQL语句执行要快。

#### 存储过程和函数的区别?

**存储过程**是用户定义的一系列sql语句的集合，涉及特定表或者其他对象的任务，用户可以调用存储过程，

**函数**通常是数据库已定义的方法，它接受参数并返回某种类型的值并且不涉及特定用户表。

#### 触发器分为事前触发和事后触发，两种触发有什么区别？语句级触发和行级触发有什么区别？

事前触发运行于触发事件发生之前，而事后触发运行于触发事件发生之后。通常事前触发器可以获取事件之前和新的字段值。

语句级触发可以在语句执行前或后执行，而行级触发在触发器所影响的每一行触发一次。

#### MySQL两种存储引擎的区别?

InnoDb：

* 支持ACID的事务，支持事务的四种隔离级别；
* 支持行级锁及外键约束：因此可以支持写并发；
* 不存储总行数；
* 一个InnoDb引擎存储在一个文件空间（共享表空间，表大小不受操作系统控制，一个表可能分布在多个文件里），也有可能为多个（设置为独立表空，表大小受操作系统文件大小限制，一般为2G），受操作系统文件大小的限制；
* 主键索引采用聚集索引（索引的数据域存储数据文件本身），辅索引的数据域存储主键的值；因此从辅索引查找数据，需要先通过辅索引找到主键值，再访问辅索引；最好使用自增主键，防止插入数据时，为维持B+树结构，文件的大调整。

MyISAM：

* 不支持事务，但是每次查询都是原子的；
* 支持表级锁，即每次操作是对整个表加锁；
* 存储表的总行数；
* 一个MYISAM表有三个文件：索引文件、表结构文件、数据文件；
* 采用菲聚集索引，索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致，但是辅索引不用保证唯一性。

#### 关系型数据库和非关系数据库区别?

**关系型数据库（MySQL、Oracle、SQL Server）**

表和表、表和字段、数据和数据存在着关系

**优点:**

1.数据之间有关系，进行数据的增删改查的时候是非常方便的

2.关系型数据库是有事务操作的，保证数据的完整性和一致性。

**缺点：**

1.因为数据和数据是有关系的，底层是运行了大量的算法，大量算法会降低系统的效率，会降低性能

2.面对海量数据的增删改查的时候会显的无能为力

3.海量数据对数据进行维护变得非常的无力

**非关系数据库的（redis和MangDB）**

为了处理海量数据，非关系数据库设计之初就是为了替代关系型数据库的关系

**优点：**

1.海量数据的增删改查是可以的

2.海量数据的维护和处理非常轻松

**缺点：**

1.数据和数据没有关系，他们之间就是单独存在的

2.非关系数据库没有关系，没有强大的事务关系，没有保证数据的完整性和安全性

#### 数据库的乐观锁和悲观锁是什么？

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。

乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

#### 数据库三范式是什么?

第一范式：原子性，表中的每个字段都不可再分割。

第二范式：满足第一范式并且表中的非主键字段都依赖于主键字段。

第三范式：满足第二范式并且表中的非主键字段必须不传递依赖于主键字段（就是主键和非主键没啥关系）。

#### 说出数据连接池的工作机制是什么？（能讲多少将多少）

**数据库连接池的基本原理:**

就是为数据库连接 建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。

设定连接池最大连接数来防止系统无尽的与数据库连接。

通过连接池的管理机制监视数据库的连接的数量的使用情况，为系统开发、测试及性能调整提供依据。

**数据库连接池的工作原理:**

第一、连接池的建立。一般在系统初始化时，连接池会根据系统配置建立，并在池中创建了几个连接对象，以便使用时能从连接池中获取。 连接池中的连接不能随意创建和关闭，这样避免了连接随意建立和关闭造成的系统开销。Java中提供了很多容器类可以方便的构建连接池，例如Vector、Stack等。

第二、连接池的管理。连接池管理策略是连接池机制的核心，连接池内连接的分配和释放对系统的性能有很大的影响。其管理策略是： 当客户请求数据库连接时，首先查看连接池中是否有空闲连接，如果存在空闲连接，则将连接分配给客户使用；如果没有空闲连接， 则查看当前所开的连接数是否已经达到最大连接数，如果没达到就重新创建一个连接给请求的客户；如果达到就按设定的最大等待时间进行等待， 如果超出最大等待时间，则抛出异常给客户。 当客户释放数据库连接时，先判断该连接的引用次数是否超过了规定值， 如果超过就从连接池中删除该连接，否则保留为其他客户服务。该策略保证了数据库连接的有效复用，避免频繁的建立、 释放连接所带来的系统资源开销。

第三、连接池的关闭。当应用程序退出时，关闭连接池中所有的连接，释放连接池相关的资源，该过程正好与创建相反。

**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：**

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；

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。

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。

#### MySQL数据库优化？

1.优化SQL语句

2.优化数据库结构

3.优化数据库服务器，硬件上提高数据库配置

4.加缓存，想办法提高缓存命中率，redis

优化SQL语句

使用多列索引时，查询条件必须使用索引的第一个字符。

like关键字配置的字符串不能以符号“%”开头。

or关键字连接的所有条件都必须使用索引。如果or前后有一个条件的列不是索引，那么查询中将不使用索引。

    尽可能对每一条运行的sql语句进行Explain

    尽力少使用Join（少使用多表查询）

    尽量少排序

    尽量避免select \* 查询

    尽量使用连接代替子查询

    尽量少使用or关键字

    尽量使用Union all代替union

    尽量早过滤

    避免“类型转换”

    优化先优化高并发的sql语句

    从全局出发优化，而不是片面调整

尽量使用表的别名(Alias)

使用>=替代>

尽量避免在 where 子句中使用!=、<> 、or、in、not 判断null，否则将引擎放弃使用索引而进行全表扫描。

尽量避免在 where 子句中使用 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描

尽量避免大事务操作，提高系统并发能力。

使用视图加速查询；

能用DISTINCT的就不用GROUP BY

尽量不要用SELECT INTO语句

使用exists代替in、使用not exists 代替not in。

避免在索引列上使用IS NULL和IS NOT NULL

 如果检索数据量超过30%的表中记录数,使用索引将没有显著的效率提高

如果某列存在空值，即使对该列建索引也不会提高性能

用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤

优化数据库结构

优化字段类型

 数据库最耗时的就是IO处理，所以尽可能减少IO读写量。

 数字类型：尽量不要使用double类型，不仅是长度问题，还有精度问题。对于整数，数据量大时区分TinyInt、Int、BigInt，如果确定不实用负数，添加unsigned定义。

 字符类型：尽量不要使用text类型，性能低于char和varchar，定长使用char，不定长使用varchar

时间类型：尽量不使用Timestamp，只精确到某一天的话，可用Date类型。

Enum和Set：状态字段用enum，如果存放可预先定义的属性数据可以尝试用set。

优化字符编码

适当进行拆分

适当增加冗余

优化数据库表

#### 已知教学数据库包含三个关系？

学生S（SND,SNAME,SA,SD）

课程C（CNO,CN,TNAME）,

选课SC（SNO,CNO,G）

其中SNO代表学号，SNAME代表学生姓名，SA代表学生年龄，SD代表学生所在系，CNO代表课程号，CN代表课程名，TNAME代表任课老师姓名，G代表成绩，

**请用SQL语句实现：**

create table S(

sno int(10) primary key,-- 学号

sname varchar(20) not null,-- 姓名

sa int(3),--年龄

sd varchar(10)-- 所在系

)

create table C(

cno int(10) primary key,,-- 课程号

cn varchar (20) unique , -- 课程名

tname int(3) not null--任课老师

)

create table SC(

sno int(10),-- 学号

cno varchar(20), -- 课程号

g int(3),--成绩

constraint fk\_sc\_s foreign key(sno) references s(sno)

constraint fk\_sc\_c foreign key(cno) references c(cno)

primary key (sno,cno)

)

将下列学生信息插入学生关系中：李丹，18岁，电信系，学号：20070206

INSERT into S(SNO,SNAME,SD) VALUES(20070206,'李丹','电信系');

找出选修了课程为“112002”的学生学号和姓名

SELECT S.SNO,S.SNAME FROM S S,SC C where S.SNO=C.SNO AND C.CNO='112002';

修改学号为“20070206”的学生所在的系为计算机

UPDATE S SET SD='计算机' WHERE SNO='20070206'

查询选修了数据库系统原理这门课的学生的姓名和成绩

SELECT S.SNAME,C.G FROM S S,SC C where S.SNO=C.SNO;

# JDBC

#### 简述JDBC的编程步骤？

1. 加载数据库驱动：Class.forName(driverClass)
2. 获取数据库连接：Connection;
3. 创建Statement/PreparedStatement对象;
4. 发送并执行SQL语句;
5. 处理数据库的返回结果集: ResultSet
6. 释放数据库资源,包括关闭ResultSet, PreparedStatement Connection

#### 简述JDBC中的Statement 和PreparedStatement，CallableStatement的区别？

* 效率性: PreparedStatement是预编译的SQL语句，效率高于Statement。
* 灵活性: PreparedStatement支持?占位符操作符，相对于Statement更加灵活。
* 安全性: PreparedStatement可以防止SQL注入，安全性高于Statement。
* CallableStatement适用于执行存储过程。

#### JDBC中大数据量的分页解决方法?

**mysql语法：**

SELECT \*

FROM 表名

LIMIT [START], length;

**oracle语法：**

SELECT \*FROM (

SELECT 列名,列名,ROWNUM rn

FROM 表名

WHERE ROWNUM<=(currentPage\*lineSize)) temp

WHERE temp.rn>(currentPage-1)\*lineSize;

#### Java中如何进行事务的处理?

事务是业务上的一个逻辑单元，它能够保证其中对数据所有的操作，要么成功，要么失败。

一个逻辑工作单元必须有四个属性，称为**原子性、一致性、隔离性和持久性 (ACID)** 属性。 Connection类中提供了4个事务处理方法:

setAutoCommit(Boolean autoCommit):设置是否自动提交事务,默认为自动提交,即为true,通过设置false禁止自动提交事务;

commit():提交事务;

rollback():回滚事务.

savepoint:保存点

注意：savepoint不会结束当前事务，普通提交和回滚都会结束当前事务的

# Servlet&JSP

#### 说出Servlet的生命周期? \*\*\*

Servlet生命周期可以分成四个阶段：加载、初始化、服务、销毁。

当客户第一次请求时，首先判断是否存在Servlet对象，若不存在，则由Web容器创建对象，而后调用init()方法对其初始化，此初始化方法在整个Servlet生命周期中只调用一次。

完成Servlet对象的创建和实例化之后，Web容器会调用Servlet对象的service()方法来处理请求。当Web容器关闭或者Servlet对象要从容器中被删除时，会自动调用destory()方法。

Servelt加载和实例化。

Servlet 通过调用 init () 方法进行初始化。

Servlet 调用 service() 方法来处理客户端的请求。

Servlet 通过调用 destroy() 方法终止（结束）。

最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

#### JSP和Servlet的区别？

JSP是Servlet技术的扩展，本质上就是Servlet的简易方式，JSP编译后是“servlet类”;

JSP更擅长表现于页面显示,Servlet更负责于逻辑控制;

#### cookie和session的区别？

Cookie是服务器在本地机器上存储的一段小文本并随下一次请求发送至同一服务器，是在客户端保持状态的方案。

Session:存在服务器的一种用来存放用户数据的类HashTable结构。浏览器第一次发送请求时，服务器自动生成了一HashTable和一Session ID来唯一标识这个HashTable，并将其通过响应发送到浏览器。浏览器第二次发送请求会将前一次服务器响应中的Session ID放在请求中一并发送到服务器上，服务器从请求中提取出Session ID，并和保存的所有Session ID进行对比，找到这个用户对应的HashTable。

**区别**

**存储数据量方面**：session 能够存储任意的 java 对象，cookie 只能存储 String 类型的对象，单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

**存储数据位置不同**：cookie数据存放在客户的浏览器上，session数据放在服务器上。

**安全性：**考虑到安全应当使用session，因Cookie在客户端所以可以编辑伪造，不是十分安全。

**时间：**session会在一定时间内保存在服务器上。tomcat默认时间30分钟,cookie可以自由设置存放时间。

**内存：**当访问增多，session会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用COOKIE。

#### Session什么时候失效？

服务器会把长时间没有活动的Session从服务器内存中清除，此时Session便失效，Tomcat中Session的默认失效时间为30分钟。

调用Session的invalidate()方法。

#### get和post的区别？

**作用不同**：get是用来从服务器上获取数据，而post是用来向服务器传递数据；

**数据形式**：get将表单中数据按照variable=value的形式，添加到action所指向的URL后面，并且两者用"？"连接，变量之间用"&"连接；而post是将表单中的数据放在form的数据体中，按照变量与值对应的方式，传递到action所指定的URL。

**安全性：**get是不安全的，因为在传输过程中，数据是被放在请求的URL中;而post的所有操作对用户来说都是不可见的。

**传输量：**get传输的数据量小，这主要应为受url长度限制;而post可以传输大量的数据，所有上传文件只能用post提交。

**编码方式**：get数据集必须为ASCII字符；而post支持整个IS01字符集。

**默认方法**：get是form表单的默认方法。

#### JSP的三大指令？

**page**指令：定义页面的一些属性，常用属性：contentType="text/html;charset=utf-8";    向浏览器端输出数据的编码；pageEncoding="utf-8";JSP编译成java文件时所用的编码；session="true"是否自动创建session；

**include**指令：引入一个静态的JSP页面；

**taglib**指令：用于定义和指定自定义标签；

#### JSP的四大作用域？

**pageContext**：如果把变量放到pageContext里，就说明它的作用域是page，它的有效范围只在当前jsp页面里。 从把变量放到pageContext开始，到jsp页面结束，你都可以使用这个变量。

**request**：如果把变量放到request里，就说明它的作用域是request，它的有效范围是当前请求周期。所谓请求周期，就是指从http请求发起，到服务器处理结束，返回响应的整个过程。在这个过程中可能使用forward的方式跳转了多个jsp页面，在这些页面里你都可以使用这个变量。

**Session**：如果把变量放到session里，就说明它的作用域是session，它的有效范围是当前会话。所谓当前会话，就是指从用户打开浏览器开始，到用户关闭浏览器这中间的过程。这个过程可能包含多个请求响应。也就是说，只要用户不关浏览器，服务器就有办法知道这些请求是一个人发起的，整个过程被称为一个会话（session）。

**Application**：如果把变量放到application里，就说明它的作用域是application，它的有效范围是整个应用。整个应用是指从应用启动，到应用结束。我们没有说“从服务器启动，到服务器关闭”是因为一个服务器可能部署多个应用，当然你关闭了服务器，就会把上面所有的应用都关闭了。application作用域里的变量，它们的存活时间是最长的，如果不进行手工删除，它们就一直可以使用。与上述三个不同的是，application里的变量可以被所有用户共用。如果用户甲的操作修改了application中的变量，用户乙访问时得到的是修改后的值。这在其他scope中都是不会发生的，page, request, session都是完全隔离的，无论如何修改都不会影响其他

#### JSP的七大动作？

JSP：include       用于动态引入一个jsp页面

JSP：forward       用于页面跳转，将请求的处理转发到另一个页面

JSP：param 用于传递参数

JSP: plugin 用于下载javaBean或applet到客户端执行

JSP：useBean     （获得JavaBean的一个实例）

JSP：setProperty （设置JavaBean的属性）

JSP：getProperty （获得JavaBean的属性）

#### JSP的九大内置对象？\*\*\*

1. 输出输入对象:request对象、response对象、out对象

2. 通信控制对象:pageContext对象、session对象、application对象

3. Servlet对象:page对象、config对象

4. 错误处理对象:exception对象

JSP内置对象：可以不用声明和创建，就可以直接在JSP页面脚本（Java程序片和Java表达式）中使用的成员变量。

那为什么不用生命和创建就可以直接使用呢？这是因为，内置对象必须由支持JSP的容器去创建。例如Tomcat就是一个容器。

**Request**(Javax.servlet.ServletRequest)包含了有关浏览器请求的信息.通过该对象可以获得请求中的头信息、Cookie和请求参数。

**Response**(Javax.servlet.ServletResponse)作为JSP页面处理结果返回给用户的响应存储在该对象中。并提供了设置响应内容、响应头以及重定向的方法(如cookies,头信息等)

**Out**(Javax.servlet.jsp.JspWriter)用于将内容写入JSP页面实例的输出流中,提供了几个方法向浏览器回送输出结果。

**pageContext**(Javax.servlet.jsp.PageContext)描述了当前JSP页面的运行环境。可以返回JSP页面的其他隐式对象及其属性的访问,另外,它还实现将控制权从当前页面传输至其他页面的方法。

**Session**(javax.servlet.http.HttpSession)会话对象存储有关此会话的信息,也可以将属性赋给一个会话,每个属性都有名称和值。会话对象主要用于存储和检索属性值。

**Application**(javax.servle.ServletContext)存储了运行JSP页面的servlet以及在同一应用程序中的任何Web组件的上下文信息。

**Page**([Java](http://lib.csdn.net/base/javaee).lang.Object)表示当前JSP页面的servlet实例

**Config**(javax.servlet.ServletConfig)该对象用于存取servlet实例的初始化参数。

**Exception**(Javax.lang.Throwable)在某个页面抛出异常时,将转发至JSP错误页面,提供此对象是为了在JSP中处理错误。只有在错误页面中才可使用<%@page isErrorPage=“true”%>

#### forward和 redirect的区别？

**转发方式**（forward）: 客户端或浏览器只发出一次请求，Servlet、HTML、JSP或其它信息资源，由第二个信息资源响应该请求，在请求对象request中，保存的对象对于每个信息资源是共享的。

**重定向方式**（redirect）: 实际是两次HTTP请求，服务器端在响应第一次请求的时候，让浏览器再向另外一个URL发出请求，从而达到转发的目的。

**区别？**

重定向是客户端行为，转发是服务器端行为

重定向时服务器产生两次请求，转发产生一次请求，重定向时可以转发到项目以外的任何网址，转发只能在当前项目里转发

重定向会导致request对象信息丢失。转发则不会

转发的url不会变,request.getRequestDispatch().forward()

重定向的url会改变,response.getRedirect().sendRedirect();

#### JSP中动态include和静态include的区别？

静态include：语法：<%@ include file="文件名" %>，相当于复制，编辑时将对应的文件包含进来，当内容变化时，不会再一次对其编译，不易维护。

动态include：语法：<jsp:include page="文件名">,能够自动检查被包含文件，当客户端对JSP文件进行请求时，会重新将对应的文件包含进来，进行实时的更新。

#### Filter过滤器、Interceptor拦截器、Listener监听器

一：拦截器 ：是在面向切面编程的就是在你的service或者一个方法，前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，springmvc的aop中的前置通知和后置通知。

二： 过滤器：是在java web中，你传入的request,response提前过滤掉一些信息，或者提前设置一些参数，  
然后再传入servlet或者struts的 action进行业务逻辑，比如过滤掉非法url（不是login.do的地址请求，如果用户没有登陆都过滤掉）,  
或者在传入servlet或者struts的action前统一设置字符集，或者去除掉一些非法字符。主要为了减轻服务器负载，减少压力

拦截器和过滤器的区别？

1、拦截器是基于java的反射机制的，而过滤器是基于函数回调   
2、过滤器依赖与servlet容器，而拦截器不依赖与servlet容器   
3、拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用   
4、拦截器可以访问action上下文、值栈里的对象，而过滤器不能   
5、在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次

在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次  
执行顺序 ：过滤前 - 拦截前 - Action处理 - 拦截后 - 过滤后。  
个人认为过滤是一个横向的过程，首先把客户端提交的内容进行过滤(例如未登录用户不能访问内部页面的处理)；  
过滤通过后，拦截器将检查用户提交数据的验证，做一些前期的数据处理，接着把处理后的数据发给对应的Action;  
Action处理完成返回后，拦截器还可以做其他过程，再向上返回到过滤器的后续操作。

三：监听器：Servlet的监听器Listener，它是实现了javax.servlet.ServletContextListener  
接口的服务器端程序，它也是随web应用的启动而启动，只初始化一次，随web应用的停止而销毁。主要作用是：做一些初始化的内容添加工作、设置一些基本的内容、比如一些参数或者是一些固定的对象等等。

#### MVC的各个部分都有那些技术来实现?如何实现?

MVC的具体含义是：Model+View+Control，即模型+视图+控制 它们各自处理自己的任务： （1）模型Model：模型持有所有的数据、状态和程序逻辑。

（2）视图 View：用来呈现模型。视图通常直接从模型中取得它需要显示的状态与数据。对于相同的信息可以有多个不同的显示形式或视图。

（3）控制器Control：位于视图和模型中间，负责接受用户的输入，将输入进行解析并反馈给模型，通常一个视图具有一个控制器。

MVC模式将它们分离以提高系统的灵活性和复用性，MVC模式实现了模型和视图的分离，这带来了几个好处。

（1）一个模型提供不同的多个视图表现形式，也能够为一个模型创建新的视图而无须重写模型。一旦模型的数据发生变化，模型将通知有关的视图，每个视图相应地刷新自己。

（2）模型可复用。因为模型是独立于视图的，所以可以把一个模型独立地移植到新的平台工作。

（3）提高开发效率。在开发界面显示部分时，你仅仅需要考虑的是如何布局一个好的用户界面；开发模型时，你仅仅要考虑的是业务逻辑和数据维护，这样能使开发者专注于某一方面的开发，提高开发效率。

#### 什么是JSTL?

JSTL是Jsp Standard Tag Library的缩写。JSP标准标签库，包含各种标签可用于 [JSP](https://www.baidu.com/s?wd=JSP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao) 页面中，其中核心标签库包含out,foreach,if,set,choose,when,otherwise等标签。

为什么要用JSTL：可以编写没有java代码的[jsp](https://www.baidu.com/s?wd=jsp&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)页面。使JSP页面很容易被WEB设计人员理解，表达更清晰。减少程序出错，是程序容易维护。

#### 如何防止表单重复提交？

使用session技术：（如注册示例）

a、在注册表单页面中生成一个随机值sessionId，将其保存到session中，同时将其保存为表单的隐藏域的值。

b、在处理注册的请求时，获取session中的值，获取请求参数的值，比较两者是否相同，如果相同说明不是重复提交;如果请求通过则删除session中保存的值，如果不相同则是重复提交，不能通过。

#### Tomcat根目录下有哪些文件？

a、config：配置文件存放路径

b、webapps：项目部署的目录

c、bin： Tomcat运行需要的脚本与jar的目录

d、lib：运行项目时所需要的jar的目录

e、word：部署项目的缓存目录

f、temp：临时文件的存放目录

g、logs：记录日志的目录

#### 什么是重量级？什么是轻量级？

轻量级是指它的创建和销毁不需要消耗太多的资源，意味着可以在程序中经常创建和销毁eg: session对象。

重量级意味着不能随意的创建和销毁它的实例，会占用太多的资源。

#### 什么是web容器？

给应用程序组件（JSP、Servlet）提供一个环境，是JSP、Servlet直接跟容器中的变量交互，不必关注其他系统问题；主要有web服务器来实现，例如：tomcat、weblogic、JBoss、sphere、等。该容器提供的接口严格遵守J2EE规范中的web application标准，我们把遵守以上标准的web服务器叫做J2EE的web容器。

#### 说明工厂模式

根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

# Spring

#### 谈谈你对Spring的理解？

Spring是一个开源框架，为简化企业级应用开发而生，

Spring可以是使简单的JavaBean实现以前只有EJB才能实现的功能。

Spring是个IoC和AOP容器框架  
Spring容器的主要核心是；  
控制反转(IoC)，传统的java开发模式中，当需要一个对象时，我们会自己使用new或者 getinstance等直接或者间接调用构造方法创建一个对象。而在spring开发模式中spring容器使用了工厂模式为我先创建了所需要的对象不需要我们自己创建了，直接调用spring提供的对象就可以了，这是控制反转的思想;  
依赖注入(D)，spring使用javaBean对象的set方法或者带参数的构造方法为我们在创建所需对象时将其属性自动设置所需要的值的过程，就是依赖注入的思想；  
面向切面编程(AOP)，在面向对象编程(oop)思想中，我们将事物纵向抽成一个个的对象，而在面向切面编程中，我们将一个个的对象某些类似的方面横向抽成一个切面，对这个切面进行一些如权限控制、事物管理，记录日志等公用操作处理的过程就是面向切面编程的思想。AOP底层是动态代理，如果是接口采用JDK动态代理，如果是类采用CGLB方式实现动态代理；

#### 简述Spring的优点？

**1.降低了组件之间的耦合性 ，实现了软件各层之间的解耦   
2.使用提供的众多服务，如事务管理，消息服务等   
3.容器提供单例模式支持   
4.容器提供了AOP技术，利用它很容易实现如权限拦截，运行期监控等功能   
5.容器提供了众多的辅助类，能加快应用的开发   
6.spring对于主流的框架提供了集成支持，如hibernate，JPA，Struts等   
7.spring属于低侵入式设计，代码的污染极低   
8.独立于各种应用服务器   
9.spring的DI机制降低了业务对象替换的复杂性   
10.spring的高度开放性，并不强制应用完全依赖于spring，开发者可以自由选择spring的部分或全部**

#### 简述Spring的IOC与AOP？

[**IOC**](https://www.baidu.com/s?wd=IOC&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**：**[**控制反转**](https://www.baidu.com/s?wd=%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**，是一种设计模式。一层含义是控制权的转移：由传统的在程序中控制依赖转移到由容器来控制；第二层是**[**依赖注入**](https://www.baidu.com/s?wd=%E4%BE%9D%E8%B5%96%E6%B3%A8%E5%85%A5&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**：将相互依赖的对象分离，在spring配置文件中描述他们的依赖关系。他们的依赖关系只在使用的时候才建立。**[**AOP**](https://www.baidu.com/s?wd=AOP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**：面向切面编程，是一种编程思想，**[**OOP**](https://www.baidu.com/s?wd=OOP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**的延续。将系统中非核心的业务提取出来，进行单独处理然后横切到业务中。比如事务、日志和安全等。  
Spring 的**[**IOC**](https://www.baidu.com/s?wd=IOC&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**和**[**AOP**](https://www.baidu.com/s?wd=AOP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**都是为了解决系统代码耦合度过高的问题，使代码重用度高、易于维护。  
不过**[**IOC**](https://www.baidu.com/s?wd=IOC&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**和**[**AOP**](https://www.baidu.com/s?wd=AOP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)**并不是spring中特有的，只是spring把他们应用的更灵活方便**

#### Spring的几种注入bean的方式？\*\*\*

**使用属性的setter方法注入**

**使用构造器注入**

**使用静态工厂注入（用于注解方式）**

#### Spring常用注解有哪些？

@Controller：用于标注控制层，它用于标记在一个类上；

@RequestMapping：请求地址映射的注解；

@RequestBody:实现接受Http请求的json数据，将json数据转换为java对象；

@ResponseBody：将controller方法返回对象转化为json响应给客户；

@PathVariable：用于将请求URL中的模板变量映射到功能处理方法的参数上，即取出uri模板中的变量作为参数；

@requestParam主要用于在SpringMVC后台控制层获取参数，类似一种是request.getParameter("name")，它有三个常用参数：defaultValue = "0", required = false, value = "isApp"；defaultValue 表示设置默认值，required 铜过boolean设置是否是必须要传入的参数，value 值表示接受的传入的参数类型。

@Qualifier("class-name")：即可指定bean；

@Service：用于标注业务层；

@Autowire：自动装配，其作用是为了消除代码Java代码里面的getter/setter与bean属性中的property。

@Resource和@Autowired都是做bean的注入时使用，

@Repository：用于标注数据访问组件；

@Component：用于泛指组件，当组件不好归类的时候，可以使用Component进行标注

#### IOC是什么？

**控制反转( Inversion of Control )**对软件而言，某一个接口具体实现的选择控制权从调用类中移除，转交给第三方容器决定。 （换而言之，即通过Spring提供的IoC容器，可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。）

**DI(Dependency Injection)** 依赖注入，让调用类对某一接口实现类的依赖关系由第三方容器注入，以移除调用类对某一接口的实现类的依赖。

#### AOP是什么?

AOP的全称是Aspect Oriented Programming，面向切面编程，是一种编程思想。是对OOP（Object Orient Programming）的一种补充，专门用于处理一些具有横切性质的服务。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。

**这样就能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。**

#### 什么场合会用AOP？

* 日志功能
* 安全检查
* 事务管理
* 缓存

#### AOP实现机制是什么？

Spring AOP采用JDK动态代理机制和CGlib字节码生成技术实现。

**JDK动态代理：**基于接口的代理，一定是基于接口，会生成目标对象的接口类型的子对象。其核心的两个类是InvocationHandler和Proxy。

**CGLib代理：**基于类的代理，只是它在运行期间生成的代理对象是针对目标类扩展的子类。CGLIB是高效的代码生成包，底层是依靠ASM（开源的java字节码编辑类库）操作字节码实现的，性能比JDK强.

#### spring如何实现事务管理？

Spring既支持编程式事务管理，也支持声明式的事务管理

**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

#### 事务的传播特性？

事务传播行为就是多个事务方法调用时，如何定义方法间事务的传播。Spring定义了7种传播行为：

（1）propagation\_requierd：如果当前没有事务，就新建一个事务，如果已存在一个事务中，加入到这个事务中，这是Spring默认的选择。

（2）propagation\_supports：支持当前事务，如果没有当前事务，就以非事务方法执行。

（3）propagation\_mandatory：使用当前事务，如果没有当前事务，就抛出异常。

（4）propagation\_required\_new：新建事务，如果当前存在事务，把当前事务挂起。

（5）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。

（6）propagation\_never：以非事务方式执行操作，如果当前事务存在则抛出异常。

（7）propagation\_nested：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与propagation\_required类似的操作。

#### Spring的Scope有以下几种，通过@Scope注解来实现:

　　（1）Singleton：一个Spring容器中只有一个Bean的实例，此为Spring的默认配置，全容器共享一个实例。

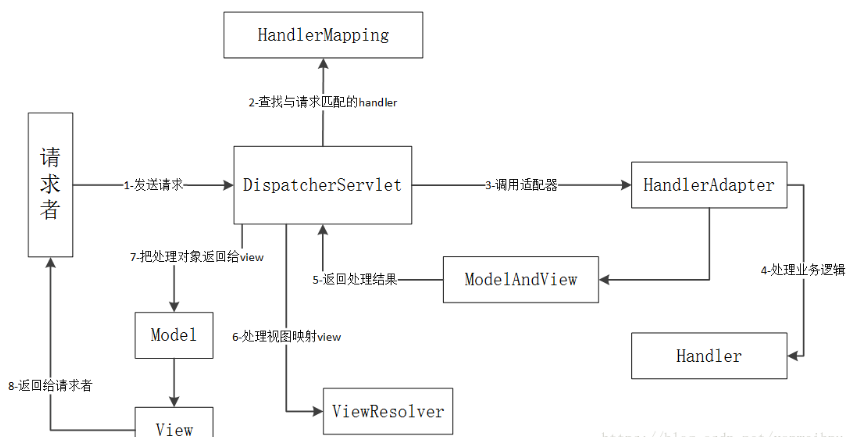
　　（2）Prototype：每次调用新建一个Bean实例。

　　（3）Request：Web项目中，给每一个 http request 新建一个Bean实例。

　　（4）Session：Web项目中，给每一个 http session 新建一个Bean实例。

　　（5）GlobalSession：这个只在portal应用中有用，给每一个 global http session 新建一个Bean实例。

#### SpringMVC 运行原理？



流程说明：

（1）客户端发送请求，直接请求到DispatcherServlet。

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

（3）解析到对应的Handler后，开始由HandlerAdapter适配器处理。

（4）HandlerAdapter会根据Handler来调用真正的处理器开处理请求，并处理相应的业务逻。

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

（6）ViewResolver会根据逻辑View查找实际的View。

（7）DispaterServlet把返回的Model传给View。

（8）通过View返回给客户端。

#### SpringMVC 的主要组件有哪些？

* 前端控制器 DispatcherServlet

作用：接收请求、响应结果 相当于转发器，有了DispatcherServlet 就减少了其它组件之间的耦合度；

* 处理器映射器HandlerMapping

作用：根据请求的URL来查找Handler；

* 处理器适配器HandlerAdapter

作用：根据handler(controlelr类）的开发方式（注解开发/其他开发） 方式的不同区寻找不同的处理器适配器

* 视图解析器 ViewResolve

作用：进行视图的解析 根据视图逻辑名解析成真正的视图；

* 视图View

作用：View是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf等等）；

#### SpringBoot 优势有哪些？

* 约定优于配置，降低环境搭建的复杂度；
* 大部分第三方配置库开箱即用；
* 内置微型tomcat，无需再装tomcat进行部署项目，maven打成jar包就可以直接运行项目；
* 自动装配Spring；
* 准生产的应用监控；
* 无代码生产和xml配置；
* SpringBoot不是借助与代码生成来实现的，而是通过条件注解来实现的，这是Spring4.x提供的新特性；

#### SpringMVC与Struts2的区别?

一、机制

SpringMVC架构的入口是以servlet开始的，而Struts2的入口是Filter（拦截器），这样就导致了两者拦截机制不同。

Struts2有自己的Interceptor实现机制，因此就导致Struts2的配置文件比较庞大，久而久之难以持久维护。虽然Struts2的配置文件是可以继承的，但是相对于SpringMVC来说还是比较麻烦的。SpringMVC采用的是独立的AOP方式，使用起来更加简洁。

SpringMVC的各个方法之间基本上是独立的，自己使用自己的Request请求数据和Response响应数据，处理结果通过Model或者JSON格式返回给框架，方法之间互不干扰，不共享参数或者变量。 而Struts2就比较麻烦了，虽然各个方法之间是相互独立的，但是一个Action中的多个方法却是参数或者变量共享的。虽然在实际使用过程中不会对运行或者数据处理产生更多的影响。但是却给开发人员和后期维护人员带来了麻烦，因为不知道这个成员变量到底是哪些地方在用。

二、性能

SpringMVC速度相对于Struts来说较快。因为SpringMVC是基于方法设计的，而Struts2是基于类设计的。Struts2每次请求都会实例化一个新的Action做处理，每一个Action都会被注入相应的属性。Struts2实际上是通过getter和setter方法与Request做数据交互的，每一个Action对应一个Request上下文。而SpringMVC的控制力度更加细腻，是方法级别的拦截，拦截方法后根据参数名称或者注解内容，把请求数据注入进去，每个Request请求都对应一个方法。

三、参数传递

Struts2在可以使用成员属性来接受参数的，也就是说这些属性在这个Action中的多个方法之间是相互共享的。而SpringMVC的Controller中的参数传递是通过方法的参数或者方法上的注解实现的。

四、设计思想

Struts2更加符合OOP的设计思想，而SpringMVC比较谨慎，不对HttpServletRequest请求做过多的干涉，只在Servlet层面做扩展。

五、补充

SpringMVC在处理Ajax的时候也是很方便的，只需要在返回View或者方法上加一个@ResponseBody注解即可。

# MyBatis

#### 什么是ORM?

**概念：**对象关系映射（Object Relational Mapping，简称ORM）模式是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。ORM是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系数据库中.

ORM提供了实现持久化层的另一种模式，它采用映射元数据来描述对象关系的映射，使得ORM中间件能在任何一个应用的业务逻辑层和数据库层之间充当桥梁。

Java典型的ORM中间件有:Hibernate,Mybatis等。

#### MyBatis的优势是什么？

* Mybatis把SQL语句从Java源程序中独立出来， 放在单独的XML文件中编写，提高程序的维护性；
* Mybatis封装了底层JDBC API的调用细节，并能自动将结果集转换成JavaBean对象， 大大简化了Java数据库编程的重复工作；
* 因为Mybatis可以灵活控制sql语句， 因此能够实现比hibernate等全自动orm框架更高的查询效率，能够完成复杂查询；

#### MyBatis的缓存机制?

mybatis的查询缓存分为一级缓存和二级缓存，

**一级缓存**: 是SqlSession级别的缓存，在操作数据库时需要构造SqlSession对象，在对象中有一个HashMap用于存储缓存数据，不同的SqlSession之间缓存数据区域（HashMap）是互相不影响的；如果SqlSession执行了DML操作（insert、update、delete），并执行commit（）操作，mybatis则会清空SqlSession中的一级缓存，这样做的目的是为了保证缓存数据中存储的是最新的信息，避免出现脏读现象；Mybatis默认开启一级缓存，不需要进行任何配置。

**二级缓存**: mapper级别的缓存，二级缓存是多个SqlSession共享的。它同样是使用HashMap进行数据存储，相比一级缓存，二级缓存的范围更大，多个SqlSession可以共用二级缓存，二级缓存是跨SqlSession的。mybatis通过缓存机制减轻数据压力，提高数据库性能。Mybatis默认没有开启二级缓存，需要在setting全局参数中配置开启二级缓存。

#### 什么是延迟加载?

**延迟加载的条件**：resultMap可以实现高级映射（使用association、collection实现一对一及一对多映射），association、collection具备延迟加载功能。

**延迟加载的好处：**

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。

**延迟加载的实例：**

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。

#### 如何开启延迟加载功能?

* Mybatis的延迟加载功能默认是关闭的
* 需要在SqlMapConfig.xml文件中通过setting标签配置来开启延迟加载功能
* 开启延迟加载的属性：

lazyLoadingEnabled：全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。默认为false  
aggressiveLazyLoading：当设置为‘true’的时候，懒加载的对象可能被任何懒属性全部加载。否则，每个属性都按需加载。默认为true

#### #{}和${}的区别是什么？

1.#{}是预编译处理，${}是字符串替换；

2.Mybatis在处理#{}时，会将sql中的#{}替换为?占位符号，调用PreparedStatement的set方法来赋值；Mybatis在处理${}时，就是把${}替换成变量的值。

3.使用#{}可以有效的防止SQL注入，提高系统安全性。

#### Mybatis是如何进行分页的？分页插件的原理是什么？

   Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

        分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

#### MyBatis和Hibernate的区别？

**开发速度的对比**: 一个项目中用到的复杂查询基本没有，就是简单的增删改查，这样选择hibernate效率就很快了，因为基本的sql语句已经被封装好了，根本不需要你去写sql语句，这就节省了大量的时间，但是对于一个大型项目，复杂语句较多，这样再去选择hibernate就不是一个太好的选择，选择[mybatis](http://www.kindve.com/forum.php?mod=viewthread&tid=23&extra=page%3D1)就会加快许多，而且语句的管理也比较方便。

**开发工作量的对比**: Hibernate和MyBatis都有相应的代码生成工具。可以生成简单基本的DAO层方法。针对高级查询，Mybatis需要手动编写SQL语句，以及ResultMap。而Hibernate有良好的映射机制，开发者无需关心SQL的生成与结果映射，可以更专注于业务流程。

**sql优化方面**: Hibernate的查询会将表中的所有字段查询出来，这一点会有性能消耗。Hibernate也可以自己写SQL来指定需要查询的字段，但这样就破坏了Hibernate开发的简洁性。而Mybatis的SQL是手动编写的，所以可以按需求指定查询的字段。

**Mybatis优势**

MyBatis可以进行更为细致的SQL优化，可以减少查询字段。

MyBatis容易掌握，而Hibernate门槛较高。

**Hibernate优势**

Hibernate的DAO层开发比MyBatis简单，Mybatis需要维护SQL和结果映射。

Hibernate对象的维护和缓存要比MyBatis好，对增删改查的对象的维护要方便。

Hibernate数据库移植性很好，MyBatis的数据库移植性不好，不同的数据库需要写不同SQL。

Hibernate有更好的二级缓存机制，可以使用第三方缓存。MyBatis本身提供的缓存机制不佳。

# AJAX

#### 要完整实现一个AJAX异步调用和局部刷新,通常需要以下几个步骤:

      (1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.

      (2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.

      (3)设置响应HTTP请求状态变化的函数.

      (4)发送HTTP请求.

      (5)获取异步调用返回的数据.

      (6)使用JavaScript和DOM实现局部刷新.

# JSON

# XML

#### 你在项目中用到了xml技术的哪些方面?如何实现的?

用到了数据存储，信息配置两方面。

在做数据交换平台时，将数据源中的数据组装成XML文件，然后将XML文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同XML文件中还原相关信息进行处理。在做软件配置时，利用XML可以很方便的进行，软件的各种配置参数都存贮在XML文件中。

#### XML文档定义有几种形式？它们之间有何本质区别？

两种形式 DTD，Schema，

区别:

* DTD无法解读扩充的内容，DTD只能把文件类型定义为一个字符串解析

XML Schema却允许把文件类型定义为整数,浮点数,字符串,布尔值或其他各数据类型,而无须重新定义

* Schema利用命名空间将文件中特殊的节点与Schema说明相联系,一个XML文件可以有多个对应的Schema;而一个XML文件只能有一个相对应的DTD文件.
* Schema是标准的XML文件,而DTD则使用自己的特殊语法,因此,只需要知道XML的语法规则就可以编写Schema了,不需要在学习其他语法规则.

#### XML文档有哪几种方式？

有DOM,SAX,STAX等

* DOM:处理大型文件时其性能下降的非常厉害。这个问题是由DOM的树结构所造成的，这种结构占用的内存较多，而且DOM必须在解析文件之前把整个文档装入内存,适合对XML的随机访问
* SAX:不限于DOM,SAX是事件驱动型的XML解析方式。它顺序读取XML文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML文件，适合对XML的顺序访问 STAX:Streaming API for XML (StAX)

#### JSON与XML比较（能讲多少讲多少）

XML: 扩展标记语言 (Extensible Markup Language, XML)

JSON(JavaScript Object Notation)一种轻量级的数据交换格式，具有良好的可读和便于快速编写的特性。

**XML的优缺点**

<1>.XML的优点  
　　A.格式统一，符合标准；  
　　B.容易与其他系统进行远程交互，数据共享比较方便。  
<2>.XML的缺点  
　　A.XML文件庞大，文件格式复杂，传输占带宽；  
　　B.服务器端和客户端都需要花费大量代码来解析XML，导致服务器端和客户端代码变得异常复杂且不易维护；  
　　C.客户端不同浏览器之间解析XML的方式不一致，需要重复编写很多代码；  
　　D.服务器端和客户端解析XML花费较多的资源和时间。

**JSON的优点：**  
　　A.数据格式比较简单，易于读写，格式都是压缩的，占用带宽小；  
　　B.易于解析，客户端JavaScript可以简单的通过eval()进行JSON数据的读取；  
　　C.支持多种语言，支持服务器端语言，便于服务器端的解析；  
　　D.在PHP世界，已经有PHP-JSON和JSON-PHP出现了，偏于PHP序列化后的程序直接调用，PHP服务器端的对象、数组等能直接生成JSON格式，便于客户端的访问提取；  
　　E.因为JSON格式能直接为服务器端代码使用，大大简化了服务器端和客户端的代码开发量，且完成任务不变，并且易于维护。  
<2>.JSON的缺点  
　　JSON格式目前在Web Service中推广还属于初级阶段。

# HTTP

#### TCP与UDP区别总结：

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接。

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付，Tcp通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。

3、UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。

4.每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信

5、TCP对系统资源要求较多，UDP对系统资源要求较少。

#### OSI七层模型

[应用层](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E5%B1%82/4329788)

[表示层](https://baike.baidu.com/item/%E8%A1%A8%E7%A4%BA%E5%B1%82/4329716)

[会话层](https://baike.baidu.com/item/%E4%BC%9A%E8%AF%9D%E5%B1%82/4329656)

[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82/4329536)

网络层

[数据链路层](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E9%93%BE%E8%B7%AF%E5%B1%82/4329290)

[物理层](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%B1%82/4329158)

# ES6

默认参数的使用：

var link = function(height = 50, color = 'red', url = 'http://azat.co') {

...

｝

变量声明：var let const

数组解构: let [a,b,c]=[1,2,3]

Let[a,[b,c],d]=[1,[2,3],4]

对象的解构赋值  
let {foo,bar} = {foo:'JSPang',bar:'技术胖'};

# Maven

#### 一个项目里面同一个jar包一般不能有两个版本的 不然可能会出现jar包冲突的情况

常见报错有这三个：

1、java.lang.ClassNotFoundException(找不到类)

2、java.lang.NoSuchMethodError(找不到具体方法)

3、java.lang.NoSuchFieldError(字段错误)

4、java.lang.LinkageError(类错误 这个我没怎么见过)

解决方法有：

1、删了旧版本的jar包，不过可能会导致一些旧代码不能用 （不推荐）

2、合并新旧两个jar包，这个有比较高的技术要求（不会）

3、拆开jar包，修改里面的package，然后在生成一个新的jar包（方便，简单，如果遇到需要就该jar包内容的情况，这个方法也适用）

# Git

#### git上传项目到github \*\*\*

第一步：建立git仓库

git init // cd到你的本地项目根目录下，执行git命令

第二步：将项目的所有文件添加到仓库中

git add . //如果想添加某个特定的文件，只需把.换成特定的文件名即可

第三步：将添加的文件commit到仓库

git commit -m "注释语句"

第四步：去github拿到创建的仓库（Repository）的https地址

第五步：将本地的仓库关联到github上

git remote add origin https:*//github.com/hanhailong/CustomRatingBar*

第六步：上传github之前，要先pull一下，执行如下命令：

git pull origin master

第七步，也就是最后一步，上传代码到github远程仓库

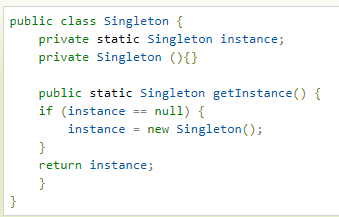
git push -u origin master

中间可能会让你输入Username和Password，你只要输入github的账号和密码就行了；

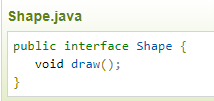
# 设计模式

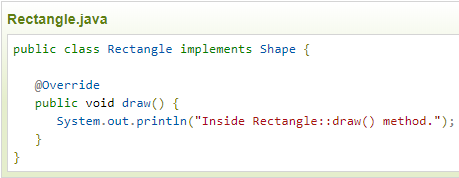
## 单例模式

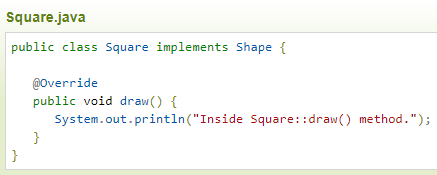
构造方法私有化。

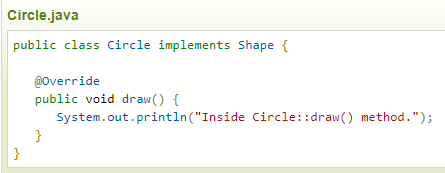


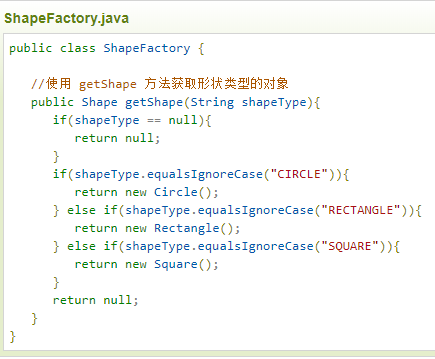
## 工厂模式













# 算法

## 排序算法

**基本思想：（分治）**先从数列中取出一个数作为key值；将比这个数小的数全部放在它的左边，大于或等于它的数全部放在它的右边；对左右两个小数列重复第二步，直至各区间只有1个数。

* public static void quickSort(int a[],int l,int r){
* if(l>=r)
* return;
* int i = l; int j = r; int key = a[l];//选择第一个数为key
* while(i<j){
* while(i<j && a[j]>=key)//从右向左找第一个小于key的值
* j--;
* if(i<j){
* a[i] = a[j];
* i++;
* }
* while(i<j && a[i]<key)//从左向右找第一个大于key的值
* i++;
* if(i<j){
* a[j] = a[i];
* j--;
* }
* }
* //i == j
* a[i] = key;
* quickSort(a, l, i-1);//递归调用
* quickSort(a, i+1, r);//递归调用
* }