JavaSE篇

Q&S基础部分

Q1.JDK 和 JRE 有什么区别?

JDK是功能齐全的Java软件开发包。JRE 是 Java运行时环境。
JDK拥有JRE所拥有的一切，还有编译器（javac）和工具（如javadoc和jdb），它能够创建和编译程序。
JRE 是 Java程序运行所需的内容的集合，它包含了 Java虚拟机（JVM），Java类库，java命令和其他的一些基础构件。但是，它不能用于创建新程序，只运行程序。

Q2.Java的基本类型有哪些？它们各自相对应的封装类又是什么？请说明int和它的封装类之间的区别。

Java的基本类型有8种：
整数型：byte、short、int、long（对应位数：8,16,32,64）
浮点型：float、double（对应位数：32,64）
字符型：char（对应位数：16）
布尔类型：boolean（单独使用的时候是会转换成int类型，如果是数组，则会转换成byte类型，因此对应两种位数，32和8位）

对应的封装类：
整数型包装类：Byte，Short，Integer，Long
浮点型包装类：Float，Double
字符型包装类：Character
布尔类型包装类：Boolean

Integer与int的区别：
int 的默认值为0，而 Integer 的默认值为 null，即 Integer 可以区分出未赋值和值为0的区别，int 则无法表达出未赋值的情况。例如，要想表达出没有参加考试和考试成绩为0的区别，则只能使用 Integer。

Q3.请说出作用域 public，private，protected，以及default的区别

流传的面试题中default经常被写成friendly，这两者没有区别，但是Java中没有friendly关键字。
public：共有的，表明该数据对所有人开放，可以直接调
private：私有的，可以理解为自己的私有财产，仅自己可以使用。
protected：受保护的，可以理解为有一群人组成一个社团，这个社团里的人可以使用，后代也可以使用。这个社团就相当于一个包，在同一个包中的类便可以访问，子类也可以访问。
default：默认的，在同一个包中的类可以访问，同一个包中的子类也可以访问，但是当子类在其他包中，就不能访

问。

作用域	当前类	同一包（package）	子孙类	其他包（package）
public	√	√	√	√
protected	√	√	√	×
friendly	√	√	×	×
private	√	×	×	×

Q4.一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制?

可以包含多个类，但是只有一个类可以使用public来修饰，并且文件名称必须与public修饰的类名称相同。

Q5.switch 语句能否作用在 byte 上，能否作用在 long 上，能否作用在 String 上?

switch表达式中，只能是int类型或者Integer或者枚举类型。byte、short、char可以隐式转换成int类型，因此可以使用这三种类型的表达式，那么long、String类型就不能应用。

Q6.short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?

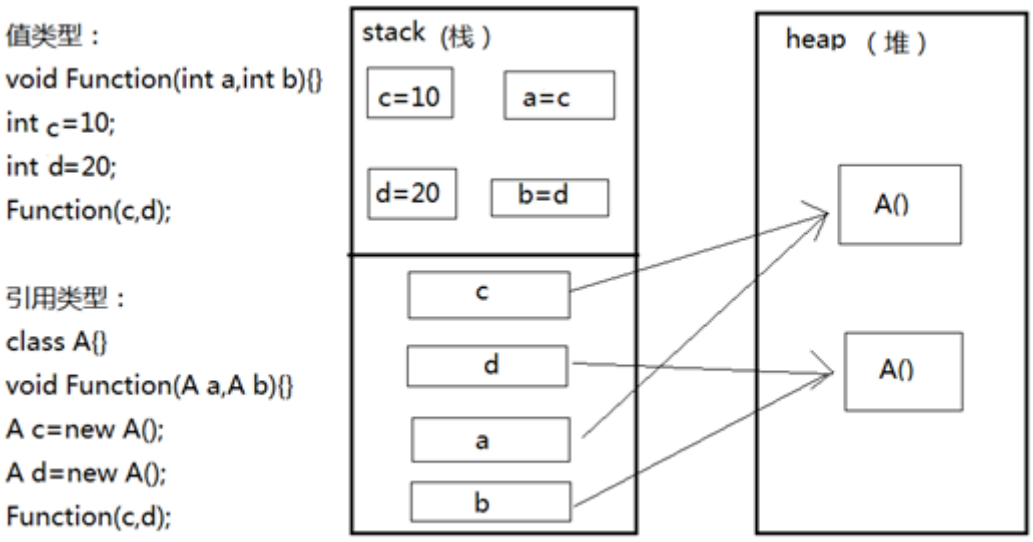
前者中s1+1会自动进行类型转换，结果是int型的，s1是short类型，将整型赋值给short型会出错。而后者中+=语句Java编译时会自动识别类型，并进行特殊处理，因此后者没有错误。

Q7.用最有效率的方法算出 2 乘以 8 等于几?用最有效的方法算出奇数和偶数？

2*8=16，我们可以得到2的二进制位10，而16的二进制数为10000，发现2的二进制数中的1向左移动三位就可以得到16的二进制数。因此我们可以使用位移运算来快速计算2<<3。
奇数的二进制数最后一位总是1，而偶数的二进制数总是0，因此我们可以使用与运算来进行奇偶数的识别。例如这个数为n，if((n&1)= =1)时，此数就是奇数；if((n&1)= =0)时，此数为偶数。

Q8.什么是引用类型?

在Java中类型可分为两大类：值类型与引用类型。值类型就是基本数据类型（如int ,double 等），而引用类型，是指除了基本的变量类型之外的所有类型（如通过 class 定义的类型）。常用引用类型为数组，接口，类（尤其String类，最常见，最长考）。所有的类型在内存中都会分配一定的存储空间(形参在使用的时候也会分配存储空间,方法调用完成之后,这块存储空间自动消失)，基本的变量类型只有一块存储空间(分配在stack中), 而引用类型有两块存储空间(一块在stack中,一块在heap中)。



1） 引用是一种数据类型（保存在stack中），保存对象在内存（heap，堆空间）中的地址，这种类型即不是我们平时所说的简单数据类型也不是类实例(对象)；
2） 不同的引用可能指向同一个对象，换句话说，一个对象可以有多个引用，即该类类型的变量。
引用其实就像是一个对象的名字或者别名 (alias)，一个对象在内存中会请求一块空间来保存数据，根据对象的大小，它可能需要占用的空间大小也不等。访问对象的时候，我们不会直接是访问对象在内存中的数据，而是通过引用去访问。引用也是一种数据类型，我们可以把它想象为类似 C++ 语言中指针的东西，它指示了对象在内存中的地址——只不过我们不能够观察到这个地址究竟是什么。
如果我们定义了不止一个引用指向同一个对象，那么这些引用是不相同的，因为引用也是一种数据类型，需要一定的内存空间（stack，栈空间）来保存。但是它们的值是相同的，都指示同一个对象在内存（heap，堆空间）的中位置。

Q9.== 和 equals 的区别是什么?

对于基本类型和引用类型 == 的作用效果是不同的，基本类型：比较的是值是否相同；引用类型：比较的是引用是否相同；
equals
equals 本质上就是 ==，只不过 String 和 Integer 等重写了 equals 方法，把它变成了值比较。String 重写了 Object 的 equals 方法，把引用比较改成了值比较。

Q10.Java 中操作字符串都有哪些类？它们之间有什么区别?

主要是String、StringBuffer、StringBuild类。
String 类是 final 类型的，因此不可以继承这个类、不能修改这个类，底层源码中有针对String数据的修改方法，都是重新创建了一个String对象，而原来的String对象未曾改变。对于字符串常量，如果内容相同，Java 认为它们代表同 一个 String 对象。而用关键字new调用构造器，总是会创建一个新的对象，无论内容是否相同。字符串如果是变量相加，先开空间，在拼接。字符串如果是常量相加，是先加，然后在常量池找，如果有就直接返回，否则，就创建。

但是为了提高效率节省空间并且可以更改对String类型数据直接更改，我们可使用用StringBuffer 类。StringBuffer线程安全，同步，效率低，开销大，因此可以改用StringBuilder。StringBuilder线程不安全，异步，效率高。

Q11.什么是同步和异步？什么是线程安全?

同步：可以理解为在执行完一个函数或方法之后，一直等待系统返回值或消息，这时程序是出于阻塞的，只有接收到返回的值或消息后才往下执行其他的命令。如打电话，通信双方不能断（我们是同时进行，同步），你一句我一句，这样的好处是，对方想表达的信息我马上能收到，但是，我在打着电话，我无法做别的事情。

异步：执行完函数或方法后，不必阻塞性地等待返回值或消息，只需要向系统委托一个异步过程，那么当系统接收到返回值或消息时，系统会自动触发委托的异步过程，从而完成一个完整的流程。如收发收短信，对方不用保证此刻我一定在手机旁，同时，我也不用时刻留意手机有没有来短信。这样的话，我看着视频，然后来了短信，我就处理短信（也可以不处理），接着再看视频。

线程安全：多个线程访问同一个对象时，如果不用考虑这些线程在运行时环境下的调度和交替执行，也不需要进行额外的同步，或者在调用方进行任何其他操作，调用这个对象的行为都可以获得正确的结果，那么这个对象就是线程安全的。一个类或者程序所提供的接口对于线程来说是原子操作或者多个线程之间的切换不会导致该接口的执行结果存在二义性,也就是说我们不用考虑同步的问题。

线程安全问题大多是由全局变量及静态变量引起的，局部变量逃逸也可能导致线程安全问题。

若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全。

Q12.String str1="i"与 String str2=new String(“i”)一样吗?

String str2 = new String(“i”)会创建2（1）个对象，String str1 = “i”创建1（0）个对象。
注:当字符串常量池中有对象hello时括号内成立!
str1 ==str2 的判断为false;
str1 .equals(str2)为true

Q13.String 类的常用方法都有那些?

1、求字符串长度
public int length()//返回该字符串的长度
2、求字符串某一位置字符
public char charAt(int index)//返回字符串中指定位置的字符；注意字符串中第一个字符索引是0，最后一个是length()-1。
3、提取子串
用String类的substring方法可以提取字符串中的子串，该方法有两种常用参数:
1)**public String substring(int beginIndex)**//该方法从beginIndex位置起，从当前字符串中取出剩余的字符作为一个新的字符串返回。
2)**public String substring(int beginIndex, int endIndex)**//该方法从beginIndex位置起，从当前字符串中取出到endIndex-1位置的字符作为一个新的字符串返回。
4、字符串比较
1)**public int compareTo(String anotherString)**//该方法是对字符串内容按字典顺序进行大小比较，通过返回的整数值指明当前字符串与参数字符串的大小关系。若当前对象比参数大则返回正整数，反之返回负整数，相等返回0。
2)**public int compareToIgnore(String anotherString)**//与compareTo方法相似，但忽略大小写。
3)**public boolean equals(Object anotherObject)**//比较当前字符串和参数字符串，在两个字符串相等的时候返回true，否则返回false。
4)**public boolean equalsIgnoreCase(String anotherString)**//与equals方法相似，但忽略大小写。
5、字符串连接
public String concat(String str)//将参数中的字符串str连接到当前字符串的后面，效果等价于"+"。
6、字符串中单个字符查找
1)**public int indexOf(int ch/String str)**//用于查找当前字符串中字符或子串，返回字符或子串在当前字符串中从左边起首次出现的位置，若没有出现则返回-1。
2)**public int indexOf(int ch/String str, int fromIndex)**//改方法与第一种类似，区别在于该方法从fromIndex位置向后查找。
3)**public int lastIndexOf(int ch/String str)**//该方法与第一种类似，区别在于该方法从字符串的末尾位置向前查找。
4)**public int lastIndexOf(int ch/String str, int fromIndex)**//该方法与第二种方法类似，区别于该方法从fromIndex位置向前查找。
7、字符串中字符的大小写转换
1)**public String toLowerCase()**//返回将当前字符串中所有字符转换成小写后的新串
2)**public String toUpperCase()**//返回将当前字符串中所有字符转换成大写后的新串
8、字符串中字符的替换
1)**public String replace(char oldChar, char newChar)**//用字符newChar替换当前字符串中所有的oldChar字符，并返回一个

新的字符串。
2)**public String replaceFirst(String regex, String replacement)**//该方法用字符replacement的内容替换当前字符串中遇到的第一个和字符串regex相匹配的子串，应将新的字符串返回。
3)**public String replaceAll(String regex, String replacement)**//该方法用字符replacement的内容替换当前字符串中遇到的所有和字符串regex相匹配的子串，应将新的字符串返回。
9、其他类方法
1)**String trim()**//截去字符串两端的空格，但对于中间的空格不处理。
2)**boolean statWith(String prefix)或boolean endWith(String suffix)**//用来比较当前字符串的起始字符或子字符串prefix和终止字符或子字符串suffix是否和当前字符串相同，重载方法中同时还可以指定比较的开始位置offset。
3)**regionMatches(boolean b, int firstStart, String other, int otherStart, int length)**//从当前字符串的firstStart位置开始比较，取长度为length的一个子字符串，other字符串从otherStart位置开始，指定另外一个长度为length的字符串，两字符串比较，当b为true时字符串不区分大小写。
4)**contains(String str)**//判断参数s是否被包含在字符串中，并返回一个布尔类型的值。
10、字符串转换为基本类型
java.lang包中有Byte、Short、Integer、Float、Double类的调用方法：
1)**public static byte parseByte(String s)**
2)**public static short parseShort(String s)**
3)**public static short parseInt(String s)**
4)**public static long parseLong(String s)**
5)**public static float parseFloat(String s)**
6)**public static double parseDouble(String s)**
11、基本类型转换为字符串类型
String类中提供了String valueOf()放法，用作基本类型转换为字符串类型。
1)**static String valueOf(char data[])**
2)**static String valueOf(char data[], int offset, int count)**
3)**static String valueOf(boolean b)**
4)**static String valueOf(char c)**
5)**static String valueOf(int i)**
6)**static String valueOf(long l)**
7)**static String valueOf(float f)**
8)**static String valueOf(double d)**
12、进制转换
使用Long类中的方法得到整数之间的各种进制转换的方法：
Long.toBinaryString(long l)
Long.toOctalString(long l)
Long.toHexString(long l)
Long.toString(long l, int p)//p作为任意进制

Q14.如何将字符串反转?

```
import java.util.Scanner;
/**
 * 使用Java中的StringBuffer完成字符串的翻转
 * @author xuanxuan
 *
 */
public class ReverseString {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
```

```
                //如果使用next()方法读取字符串时，读到空格就会停止，而使用nextLine()方法会读取空格
                String s = in.nextLine();
                System.out.println(ReverseString(s));
                in.close();
        }

        public static String ReverseString(String str) {
                StringBuffer bu = new StringBuffer();
                bu.append(str);
                String str1 = bu.reverse().toString();
                return str1;
        }
}
```

Q15.final 在 java 中有什么作用?

final 修饰的类叫最终类，该类不能被继承。
final 修饰的方法不能被重写。
final 修饰的变量叫常量，常量必须初始化，初始化之后值就不能被修改。
使用 final 关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象中的内容 还是可以改变的。

Q16.java 中的 Math.round(-1.5) 等于多少?

Ceil向上取整，floor向下取整。Round先对一个树+0.5，然后向下取整。因此这个表达式的值为-1

Q17.是否可以从一个 static 方法内部发出对非 static 方法的调用?

不可以。因为非 static 方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对 象上进行方法调用，而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一 个 static 方法被调用时，可能还没有创建任何实例对象，如果从一个 static 方法中发出对非 static 方法的调用，那个非 static 方法是关联到哪个对象上的呢？这个逻辑无法成立，所以， 一个 static 方法内部发出对非 static 方法的调用。/

Q18.Overload 和 Override 的区别。Overloaded 的方法是否可以改变返回值的类型?

Overload是重载的意思，Override是覆盖的意思，也就是重写。

重载 Overload表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写 Override 表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。

子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。

子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是 private 类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

如果几个 Overloaded 的方法的参数列表不一样，它们的返回者类型当然也可以不一样。如果两个方法的参数列表完全一样，是否可以让它们的返回值不同来实现重载 Overload?

这是不行的，我们可以用反证法来说明这个问题， 因为我们有时候调用一个方法时也可以不定义返回结果变量，即不要关心其返回结果，例如，我们调用 map.remove(key)方法时，虽然 remove 方法有返回值，但是我们通常都不会定义接收返回结果的变量，这时候假设该类中有两个名称和参数列表完全相同的方法，仅仅是返回类型不同,java 就无法

确定编程者倒底是想调用哪个方法了，因为它无法通过返回结果类型来判断。 override 可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。对我们来说最熟悉的覆盖就是对接口方法的实现，在接口中一般只是对方法 进行了声明，而我们在实现时，就需要实现接口声明的所有方法。除了这个典型的用法以外， 我们在继承中也可能会在子类覆盖父类中的方法。

在覆盖要注意以下的几点：

1、覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；

2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；

3、覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；

4、被覆盖的方法不能为 private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

overload 对我们来说可能比较熟悉，可以翻译为重载，它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM 就会根据不同的参数样式，来选择合适的方法执行。在使用重载要注意以下的几点：

1、在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是 fun(int,float)， 但是不能为 fun(int,int)）； 2、不能通过访问权限、返回类型、抛出的异常进行重载；

3、方法的异常类型和数目不会对重载造成影响；

4、对于继承来说，如果某一方法在父类中是访问权限是 priavte，那么就不能在子类对其进 行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

Q19.构造器 Constructor 是否可被 override?

构造器 Constructor 不能被继承，因此不能重写 Override，但可以被重载 Overload。

Q20.abstract class 和 interface 有什么区别?

Abstract：

- 含有 abstract 修饰符的 class 即为抽象类，abstract 类不能创建的实例对象。
- 含有 abstract 方法的类必须定义为abstract class，abstract class类中的方法不必是抽象的。abstract class类中定义抽象方法必须在具体(Concrete)子类中实现，所以，不能有抽象构造方法或抽象静态方法。
- 如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为 abstract 类型。

接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。
接口中的方法定义默认为 public abstract 类型，接口中的成员变量类型默认为public static final。

两者区别：
1.抽象类可以有构造方法，接口中不能有构造方法。
2.抽象类中可以有普通成员变量，接口中没有普通成员变量
3.抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象 的普通方法。
4.抽象类中的抽象方法的访问类型可以是 public，protected 和（默认类型,虽然 eclipse 下不报错，但应该也不行），但接口中的抽象方法只能是 public 类型的，并且默认即 为 public abstract 类型。
5.抽象类中可以包含静态方法，接口中不能包含静态方法
6.抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任 意，但接口中定义的变量只能是 public static final 类型，并且默认即为 public static final 类 型。
7.一个类可以实现多个接口，但只能继承一个抽象类。

Q21.接口是否可继承接口?抽象类是否可实现(implements)接口?抽象类是否可 继承具体类(concrete class)?抽象类中是否可以有静态的 main 方法?

接口可以继承接口。抽象类可以实现(implements)接口，抽象类可以继承具体类。抽象类中 可以有静态的 main 方法。抽象类与普通类的唯一区别：就是不能创建实例对象和允许有 abstract 方法。

Q22.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。

Q22.说出一些常用的类，包，接口，请各举 5 个?

常用的类：BufferedReader BufferedWriter FileReader FileWirter String Integer java.util.Date，System，Class，List,HashMap
常用的包：java.lang java.io java.util java.sql,javax.servlet,org.hibernate
常用的接口：Remote List Map Document
NodeList,Servlet,HttpServletRequest,HttpServletResponse,Transaction(Hibernate)、 Session(Hibernate),HttpSession

Q23.Java类的初始化顺序

普通类中:

- 静态变量
- 静态代码块
- 普通变量
- 普通代码块
- 构造函数

含有子类的:

- 父类静态变量
- 父类静态代码块
- 子类静态变量
- 子类静态代码块
- 父类普通变量
- 父类普通代码块
- 父类构造函数
- 子类普通变量
- 子类普通代码块
- 子类构造函数

含有接口、抽象类、实现类的

- 接口静态变量
- 抽象类静态变量
- 抽象类静态代码块
- 实现类静态变量
- 实习类静态代码块
- 抽象类普通变量
- 抽象类普通代码块
- 抽象类构造函数
- 实现类普通变量

- 实现类普通代码块
- 实现类构造函数

Q23.private可以通过反射访问，那么private的意义是什么

private其实只是一种Java编写规范，并不具备绝对安全的特征，当外部类对对象进行调用时，可以看到清晰的类结构。

Q24.局部变量为何要显示赋值，否则编译不通过

局部变量的赋值和取值是由绝对性的先后顺序，由编译器来决定，这是一种约束。编译不通过是因为要防止局部变量忘记赋值，而引起的失误。

Q25.静态变量与实例变量的区别

静态变量使用static声明时，属于类，也称为类变量或者全局变量，程序只要加载，静态变量就可以使用。
实例变量属于某个对象属性，必须创建实例对象。

Q26.解释一下什么是构造器

在创建对象时执行初始化，通过new关键字来调用构造器，构造器返回该类对象，但这个对象并不全由构造器负责。构造器不能被继承，因此不能重写，但是可以重载。一个子类在显示调用父类构造器时，必须用super

Q27.1/0和1.0/0.0有什么区别

1/0会爆出异常，而1.0/0.0是无穷大

Q28.将GBK字节流转换为UTF-8

```
byte[] str , dst;
dst = new String(src , "GBK").getBytes("UTF-8");
```

Q28.二维数组的定义

```
float f[][] = new float[6][6];
float []f[] = new float[6][6];
float [][]f = new float[6][6];
float [][]f = new float[6][];
```

Q29.String str = new String(“abc”)在内存中如何分配?

“abc”保存在常量池中，str作为对象被保存在堆中，然而Java7后，将常量池放在了堆里。

Q30.下列程序中有什么错误?

```
父类{方法1}

子类extends父类{重写方法1，方法2}

父类 对象 = new 子类();
```

此代码不能通过编译，编译器认为对象是父类类型，父类中没有子类中的方法2，对象只能调用父类中的方法1。

Q31.JDBC中获取结果集的操作有哪些?

```
1.Statement sta = con.createStatment();
  ResultSet rs = sta.executeQuery("SQL语句");

2.PreparedStatement pst = con.prepareStatement("SQL语句");
  ResultSet rs = pst.executeQuery();
```

Q32.Java中的IO流分几种?

按流向分（站在程序角度考虑）

```
输入流(input)

输出流(output)
```

按类型分:

```
字节流(InputStream/OutputStream)

    任何文件都可以通过字节流进行传输。

字符流(Reader/Writer)

    非纯文本文件，不能用字符流，会导致文件格式破坏，不能正常执行。
```

按功能分:

```
节点流(低级流:直接跟输入输出源对接)

    FileInputStream/FileOutputStream/FileReader/FileWriter/PrintStream/PrintWriter.

处理流(高级流:建立在低级流的基础上)

    转换流：InputStreamReader/OutputStreamWriter，字节流转字符流/字符流转字节流

    缓冲流：BufferedInputStream/BufferedOutputStream   BufferedReader/BufferedReader可对节点
```

Q33.BIO\NIO\AIO中的区别?

- Java BIO： 同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善。
- Java NIO： 同步非阻塞，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有I/O请求时才启动一个线程进行处理。
- Java AIO(NIO.2)： 异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理。

BIO、NIO、AIO适用场景分析:

- BIO方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4以前的唯一选择，但程序直观简单易理解。
- NIO方式适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4开始支持。
- AIO方式使用于连接数目多且连接比较长（重操作）的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂，JDK7开始支持。

Q34.Files的常用方法?

（一）访问文件名或路径

1）String getName() 返回File对象所表示的文件名或文件路径

2）String getPath() 返回File对象所对应的相对路径名。

3）File getAbsoluteFile() 返回File对象的绝对路径文件

4）String getAbsolutePath() 返回File对象所对应的绝对路径名

5）String getParent（） 返回File对象所对应目录的父目录

6. boolean renameTo(File dest) 重命名File对象的文件或目录

（二）文件检测

1boolean exists() 判断File对象的文件或目录是否存在

2)bool canWrite() 判断File对象是否可写

3)boolean canRead()判断File对象是否可读

4)boolean isDirectory() 判断File对象是否是目录

5)boolean isFile() 判断File对象是否是文件

6)boolean isAbsolute() 判断File对象是否采用绝对路径

(三）文件信息

1)long length() ; File对象对应文件的长度

2)long lastNodified() File对象最后修改的时间

(四）文件操作

1）boolean createNewFile() ; 检查文件是否存在，当文件不存在时创建一个新的文件

2. boolean delete() 删除File对象所对应的文件或目录

(五）目录操作

1)boolean mkdir() 创建一个File对象所对应的路径

2)String[] list() 列出File对象所有的子文件名和路径名

3)File[] listFile() 列出File对象的所有子文件或路径

4)static File[] listRoots() 列出系统所有的根路径

Q35.什么是内部类?

内部类就是在一个类的内部定义的类，内部类中不能定义静态成员。内部类可以直接访问外部类中的成员变量，如果不是静态内部类，没有什么限制！内部类可以定义在外部类的方法外面， 也可以定义在外部类的方法体中。静态内部类不能访问外部类的成员。(匿名内部类)是否可以 extends(继承)其它类， 是否可以 implements(实现)interface(接口)? 可以继承其他类或实现其他接口。不仅是可以，而是必须!

Q&S集合部分

Q1.Java中的容器有哪些?

数组、String（底层是char类型数组）、List（线性表）、Set（无序存储，元素不能重复）、Map（无序存储，元素可以重复）

Q2.Collection 和 Collections 有什么区别?

1、java.util.Collection 是一个**集合接口（集合类的一个顶级接口）**。它提供了对集合对象进行基本操作的通用接口方法。Collection接口在Java 类库中有很多具体的实现。Collection接口的意义是为各种具体的集合提供了最大化的统一操作方式，其直接继承接口有List与Set。

Collection
├List
| ├LinkedList
| ├ArrayList
| └Vector
| └Stack
└Set

2、Collections则是集合类的一个工具类/帮助类，其中提供了一系列静态方法，用于对集合中元素进行排序、搜索以及线程安全等各种操作。

1. 排序(Sort)
   使用sort方法可以根据元素的自然顺序 对指定列表按升序进行排序。列表中的所有元素都必须实现 Comparable 接口。此列表内的所有元素都必须是使用指定比较器可相互比较的
2. 混排（Shuffling）
   混排算法所做的正好与 sort 相反: 它打乱在一个 List 中可能有的任何排列的踪迹。也就是说，基于随机源的输入重排该 List, 这样的排列具有相同的可能性（假设随机源是公正的）。这个算法在实现一个碰运气的游戏中是非常有用的。Collections.Shuffling(list)
3. 反转(Reverse)
   使用Reverse方法可以根据元素的自然顺序 对指定列表按降序进行排序。Collections.reverse(list)
4. 替换所以的元素(Fill)
   使用指定元素替换指定列表中的所有元素。Collections.fill(li,"aaa");

5. 拷贝(Copy)
   用两个参数，一个目标 List 和一个源 List, 将源的元素拷贝到目标，并覆盖它的内容。目标 List 至少与源一样长。如果它更长，则在目标 List 中的剩余元素不受影响。Collections.copy(list,li): 前面一个参数是目标列表 ,后一个是源列表。
6. 返回Collections中最小元素(min)
   根据指定比较器产生的顺序，返回给定 collection 的最小元素。collection 中的所有元素都必须是通过指定比较器可相互比较的。
   Collections.min(list)
7. 返回Collections中最小元素(max)
   根据指定比较器产生的顺序，返回给定 collection 的最大元素。collection 中的所有元素都必须是通过指定比较器可相互比较的。
   Collections.max(list)
8. lastIndexOfSubList
   返回指定源列表中最后一次出现指定目标列表的起始位置
   int count = Collections.lastIndexOfSubList(list,li);
9. IndexOfSubList
   返回指定源列表中第一次出现指定目标列表的起始位置
   int count = Collections.indexOfSubList(list,li);
10. Rotate
    根据指定的距离循环移动指定列表中的元素
    Collections.rotate(list,-1);如果是负数，则正向移动，正数则方向移动。

Q3.List、Set、Map 有什么区别?

List：

- 可以允许重复的对象。
- 可以插入多个null元素。
- 是一个有序容器，保持了每个元素的插入顺序，输出的顺序就是插入的顺序。
- 常用的实现类有 ArrayList、LinkedList 和 Vector。ArrayList 最为流行，它提供了使用索引的随意访问，而 LinkedList 则对于经常需要从 List 中添加或删除元素的场合更为合适。

Set：

- 不允许重复对象
- 无序容器，你无法保证每个元素的存储顺序，TreeSet通过 Comparator 或者 Comparable 维护了一个排序顺序。
- 只允许一个 null 元素
- Set 接口最流行的几个实现类是 HashSet、LinkedHashSet 以及 TreeSet。最流行的是基于 HashMap 实现的 HashSet；TreeSet 还实现了 SortedSet 接口，因此 TreeSet 是一个根据其 compare() 和 compareTo() 的定义进行排序的有序容器。

Map:

- Map不是collection的子接口或者实现类。Map是一个接口。
- Map 的 每个 Entry 都持有两个对象，也就是一个键一个值，Map 可能会持有相同的值对象但键对象必须是唯一的。
- TreeMap 也通过 Comparator 或者 Comparable 维护了一个排序顺序。
- Map 里你可以拥有随意个 null 值但最多只能有一个 null 键。

- Map 接口最流行的几个实现类是 HashMap、LinkedHashMap、Hashtable 和 TreeMap。（HashMap、TreeMap最常用）

Q4.什么场景下使用list、set、map?

1. 如果你经常会使用索引来对容器中的元素进行访问，那么 List 是你的正确的选择。如果你已经知道索引了的话，那么 List 的实现类比如 ArrayList 可以提供更快速的访问,如果经常添加删除元素的，那么肯定要选择 LinkedList。
2. 如果你想容器中的元素能够按照它们插入的次序进行有序存储，那么还是 List，因为 List 是一个有序容器，它按照插入顺序进行存储。
3. 如果你想保证插入元素的唯一性，也就是你不想有重复值的出现，那么可以选择一个 Set 的实现类，比如 HashSet、LinkedHashSet 或者 TreeSet。所有 Set 的实现类都遵循了统一约束比如唯一性，而且还提供了额外的特性比如 TreeSet 还是一个 SortedSet，所有存储于 TreeSet 中的元素可以使用 Java 里的 Comparator 或者 Comparable 进行排序。LinkedHashSet 也按照元素的插入顺序对它们进行存储。
4. 如果你以键和值的形式进行数据存储那么 Map 是你正确的选择。你可以根据你的后续需要从 Hashtable、HashMap、TreeMap 中进行选择。

Q5.HashMap 和 Hashtable 有什么区别?

共同点：都完成了Map接口的实现
区别：

- HashMap允许空键值，异步处理，非线程安全，只有一个线程的情况下效率高于HashTable。HashTable同步，线程安全。
- HashMap将HashTable中的contains方法更改为containsvalue和containskey。
- HashMap是Map接口的一个实现，HashTable基于陈旧的Dictionary抽象类。

Q6.HashMap 和 Hashtable的实现原理，主要讲讲HashMap的原理？（HashMap的1.7和1.8的区别）

共同点：实现原理基本相同，都是使用哈希表的“拉链法”，基于数组（Entry类型）和链表实现。两者构造方法的意思是相同的。

```
HashMap(){}
默认容量为16，装填因子为0.75。HashTable默认容量为11。
HashMap(int initialCapacity){}
指定初始容量，装填因子默认为0.75
HashMap(int initialCapacity, float loadFactor){}
指定容量和装填因子
HashMap(Map<?extends K,? extends V> m)
构造新的HashMap
```

考察HashMap的原理，就是考察put和get原理。
put原理根据key值获取相应的hash值：

```
int hash = hash(key.has.hascode())
```

其实就是使用哈希表原理中的除留余数法来确定key值应该在数组中的位置，数组的作用是充当索引，key值相同映射到数组中的同一位置，使用头插法放在表头。

get原理利用hash值先进行数组中定位，再遍历链表使用equals()方法匹配。

当HashMap达到默认因子0.75，会自动双倍扩容，扩容后重新计算每个key值的hash值。数组长度必须为16或者2的幂次，这样做的目的是使位运算的结果能够均匀分布。

HashMap没有加锁，因此造成了非线程安全，若HashMap接近临界点，且有两个或多个线程并发put操作，会进行扩容与hash值重新结算，而rehash在并发情况下会形成链表环。

链表环的判断：两个指针A和B，同时指向头结点，然后开始循环遍历，让A每次下移一个节点，让B每次下移两个节点，然后比较两个指针指的节点是否相同，相同则有环。

JDK8中HashMap引入了红黑树（自平衡二叉树），提升了查询，插入和删除

Q7.ConcurrentHashMap的原理？（的1.7和1.8的区别）

concurrenthashmap既可以保证安全，又能保证性能。其原理就是一个二级哈希表，一个总的哈希表下面有若干子哈希表，由segement数组组成，使用分段锁，并行插入时效率高。

concurrenthashmap的put原理：

- 先为key值做hash运算，通过hash值定位到segement对象
- 获取可重入锁，对分段进行加锁
- 再次计算hash值，定位到segemen里的具体位置
- 插入或者覆盖HashEntry对象
- 释放锁

get原理：

- 为输入的key做hash运算，得到hash值
- 通过hash值，定位到对应的segement对象
- 再次通过hash值，定位到segement当中数组的具体位置

JDK8中，concurrenthashmap放弃了segement分段机制，利用Node数组+CAS+Synchronized来保证并发更新安全，底层为数组+链表+红黑树。

Q8.说一下 HashSet 的实现原理?

实现了Set接口，但底层由HashMap支持，不包含重复元素，且维持自己的内部排序，可使用null。

```
iterator()返回set元素
public Iterator<E> iterator(){return map.keyset().iterator();}

size()返回大小
public int size(){return map.size();}

isEmpty()判空
return map.isEmpty();

contains(Object o)判断是否存在某元素
return map.containskey(o);
```

```
add()添加元素
return map.put(e,PRESENT)==null;

remove()删除元素
return map.remove(o)==PRESENT;

clear()清除所有元素
map.clear();

clone获取HashMap的浅表副本，并没有复制这些元素本身
try{
  HashSet<E> newSet = (HashSet<E>) super.clone();
  newSet.map = (HashSet<E,Object>) map.clone();
  return newSet;
}caatch(CloneNotSupportedException e){
  throw new InternalError();
}
```

Q9.ArrayList 和 LinkedList 的区别是什么?

其本质就是顺序表与链表的区别：只不过ArrayList实现了动态扩容。传统顺序表的存储空间是一次性分配，且空间地址连续。

1.ArrayList的实现是基于数组，LinkedList的实现是基于双向链表。

2. 对于随机访问，ArrayList优于LinkedList，以O(1)时间复杂度对元素进行随机访问。LinkedList的每一个元素都依靠地址指针和它后一个元素连接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。
3. 对于插入和删除操作，LinkedList优于ArrayList。当元素被添加到LinkedList任意位置的时候，不需要像ArrayList那样重新计算大小或者是更新索引。
4. LinkedList比ArrayList更占内存，因为LinkedList的节点除了存储数据，还存储了两个引用，一个指向前一个元素，一个指向后一个元素。

Q10.如何实现数组和 List 之间的转换?

List转数组：toArray(arraylist.size()方法

```
ArrayList<String> list = new ArrayList<String>();
                list.add("dff1");
                list.add("dff2");
                list.add("dff3");
                list.add("dff4");
                list.add("dff5");
                String[] array = new String[list.size()];
                String[] s=list.toArray(array);
                System.out.println(Arrays.toString(s));
```

数组转List：Arrays的asList(a)方法

```
String[] string=new String[]{"s1","s2","s3","s4","s5"};

                List<String> li= Arrays.asList(string);

                System.out.println(li);
```

Q11.Array、ArrayList 和 Vector 的区别是什么?

a. Array是静态连续分配的一片内存区域，与ArrayList相比、不能动态改变大小，通过Arrays进行sort、binarySearch等操作；

b. ArrayList是继承自List的可动态改变大小的数组，和Array一样要求连续分配，内部封闭了一个Object数组，许多方法直接调用Arrays实现；

c. Vector和ArrayList功能基本一致，但Vector是线程安全的。

效率由高到低依次为：Array、ArrayList、Vector。

Q12.在 Queue 中 add()和offer()、poll()和 remove()、element()和peek()有什么区别?

1.queue的增加元素方法add和offer的区别在于，add方法在队列满的情况下将选择抛异常的方法来表示队列已经满了，而offer方法通过返回false表示队列已经满了；在有限队列的情况，使用offer方法优于add方法；

2.remove方法和poll方法都是删除队列的头元素，remove方法在队列为空的情况下将抛异常，而poll方法将返回null；

3.element和peek方法都是返回队列的头元素，但是不删除头元素，区别在与element方法在队列为空的情况下，将抛异常，而peek方法将返回null.

Q13.哪些集合类是线程安全的?

- vector：就比arraylist多了个同步化机制（线程安全），效率较低。
- statck：堆栈类，先进后出
- hashtable：就比hashmap多了个线程安全
- enumeration：枚举，相当于迭代器

Q14.迭代器 Iterator 是什么？Iterator 怎么使用？有什么特点?

在Java中，有很多的数据容器，对于这些的操作有很多的共性。Java采用了迭代器来为各种容器提供了公共的操作接口。这样使得对容器的遍历操作与其具体的底层实现相隔离，达到解耦的效果。

Iterator遍历集合元素有以下几个特点:

- Iterator遍历集合元素的过程中不允许线程对集合元素进行修改，否则会抛出ConcurrentModificationEception的异常。
- Iterator遍历集合元素的过程中可以通过remove方法来移除集合中的元素。
- Iterator必须依附某个Collection对象而存在，Iterator本身不具有装载数据对象的功能。
- Iterator.remove方法删除的是上一次Iterator.next()方法返回的对象。
- 强调以下next（）方法，该方法通过游标指向的形式返回Iterator下一个元素。

itertor接口中常用的方法:

- boolean型的hasNext()，如果仍有元素可以迭代返回true
- E型的next()，返回迭代的下一个元素

- void型的remove()，从迭代器指向的collection中移除迭代器返回的最后一个元素。
- void forEachRemaining(Consumer action) ;使用Lambdda表达式的形式输出Iterator中所以的元素。注意该方法其实是间接调用next()方法进行遍历，所以再次是next（）方法的时候Iterator中的对象已经被遍历完了。

Q15.Iterator 和 ListIterator 有什么区别?

1. ListIterator有add()方法，可以向List中添加对象，而Iterator不能
2. ListIterator和Iterator都有hasNext()和next()方法，可以实现顺序向后遍历，但是ListIterator有hasPrevious()和previous()方法，可以实现逆向（顺序向前）遍历。Iterator就不可以。
3. ListIterator可以定位当前的索引位置，nextIndex()和previousIndex()可以实现。Iterator没有此功能。
4. 都可实现删除对象，但是ListIterator可以实现对象的修改，set()方法可以实现。Iierator仅能遍历，不能修改。

Q16.怎么确保一个集合不能被修改?

```
Collections.unmodifiableList(List)
Collections.unmodifiableMap(Map)
Collections.unmodifiableSet(Set)
```

以上返回的集合对象都是不可修改的，调用修改方法会抛出异常UnsupportedOperationException，使用这种方法返回集合的一个只读视图。

Q&S线程部分

Q1.并行和并发有什么区别?

假设一个有三个学生需要辅导作业，帮每个学生辅导完作业是一个任务

- 顺序执行：老师甲先帮学生A辅导，辅导完之后再取给B辅导，最后再去给C辅导，效率低下 ，很久才完成三个任务
- 并发：老师甲先给学生A去讲思路，A听懂了自己书写过程并且检查，而甲老师在这期间直接去给B讲思路，讲完思路再去给C讲思路，让B自己整理步骤。这样老师就没有空着，一直在做事情，很快就完成了三个任务。与顺序执行不同的是，顺序执行，老师讲完思路之后学生在写步骤，这在这期间，老师是完全空着的，没做事的，所以效率低下。
- 并行：直接让三个老师甲、乙、丙三个老师“同时”给三个学生辅导作业，也完成的很快。
  理解：
  1）并行是指两个或者多个事件在**同一时刻**发生；而并发是指两个或多个事件在同一时间间隔发生。也就是说并发：交替做不同事情的能力。并行：同时做不同事情的能力。
  2）并行是在不同实体上的多个事件，并发是在同一实体上的多个事件。

Q2.线程和进程的区别?

- 进程是资源分配的最小单位，线程是程序执行的最小单位。
- 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据的，使用相同的地址空间，因此CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。

- 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。
- 但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。

Q3.守护线程是什么?

守护线程，是个服务线程，准确地来说就是服务其他的线程。Java里线程分2种，
1、守护线程，比如垃圾回收线程，就是最典型的守护线程。
2、用户线程，就是应用程序里的自定义线程。
如果其他的线程（即用户自定义线程）都执行完毕，连main线程也执行完毕，那么jvm就会退出（即停止运行）——此时，连jvm都停止运行了，守护线程当然也就停止执行了。

Q4.创建线程有哪几种方式?

创建线程的第一种方式：

- 创建一个类继承Thread
- 重写Thread中的run方法 （创建线程是为了执行任务 任务代码必须有存储位置，run方法就是任务代码的存储位置。）
- 创建子类对象，其实就是在创建线程
- 启动线程start（）

这种方式的特点（缺陷）：线程任务和线程是绑定在一起的。

创建线程的第二种方式：

- 创建实现了Runnable接口的子类
- 重写Runnable接口中的run方法
- 创建实现了Runnable接口的子类的对象
- 创建Thread类的对象，也就是在创建线程
- 把实现了Runnable接口的子类对象作为参数传递给Thread类的构造方法

这种方式的特点是：把线程任务进行了描述，也就是面向对象，从而实现了线程任务和线程对象的分离。线程执行什么任务不再重要，只要是实现了Runnable接口的子类对象都可以作为参数传递给Thread的构造方法，此方式较为灵活。第二种方式还有一个好处是实现接口了，还不影响继承其他父类。

使用Callable和Future创建线程：

- 创建Callable接口的实现类，并实现call()方法，然后创建该实现类的实例（从java8开始可以直接使用Lambda表达式创建Callable对象）。
- 使用FutureTask类来包装Callable对象，该FutureTask对象封装了Callable对象的call()方法的返回值
- 使用FutureTask对象作为Thread对象的target创建并启动线程（因为FutureTask实现了Runnable接口）
- 调用FutureTask对象的get()方法来获得子线程执行结束后的返回值

Q5.说一下 runnable 和 callable 有什么区别?

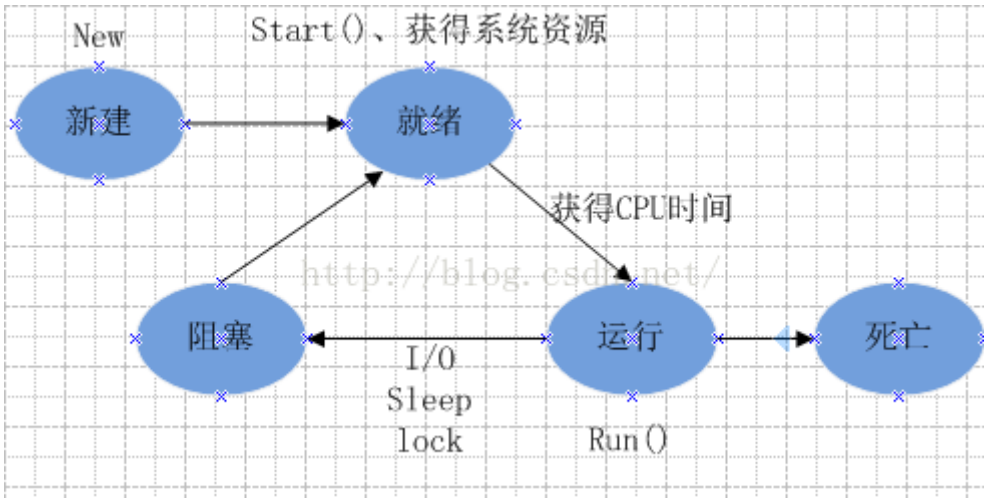
相同点：

1. 两者都是接口；
2. 两者都可用来编写多线程程序；
3. 两者都需要调用Thread.start()启动线程；

不同点：

1. 两者最大的不同点是：实现Callable接口的任务线程能返回执行结果；而实现Runnable接口的任务线程不能返回结果；
2. Callable接口的call()方法允许抛出异常；而Runnable接口的run()方法的异常只能在内部消化，不能继续上抛；

Q6.线程有哪些状态?



线程运行状态

Q7.sleep() 和 wait() 有什么区别?

- 原理不同。sleep()方法是Thread类的静态方法，是线程用来控制自身流程的，他会使此线程暂停执行一段时间，而把执行机会让给其他线程，等到计时时间一到，此线程会自动苏醒。例如，当线程执行报时功能时，每一秒钟打印出一个时间，那么此时就需要在打印方法前面加一个sleep()方法，以便让自己每隔一秒执行一次，该过程如同闹钟一样。而wait()方法是object类的方法，用于线程间通信，这个方法会使当前拥有该对象锁的进程等待，直到其他线程调用notify（）方法或者notifyAll（）时才醒来，不过开发人员也可以给他指定一个时间，自动醒来。
- 对锁的 处理机制不同。由于sleep()方法的主要作用是让线程暂停执行一段时间，时间一到则自动恢复，不涉及线程间的通信，因此，调用sleep()方法并不会释放锁。而wait()方法则不同，当调用wait()方法后，线程会释放掉他所占用的锁，从而使线程所在对象中的其他synchronized数据可以被其他线程使用。
- 使用区域不同。wait()方法必须放在同步控制方法和同步代码块中使用，sleep()方法则可以放在任何地方使用。sleep()方法必须捕获异常，而wait()、notify（）、notifyAll（）不需要捕获异常。在sleep的过程中，有可能被其他对象调用他的interrupt（），产生InterruptedException。由于sleep不会释放锁标志，容易导致死锁问题的发生，因此一般情况下，推荐使用wait（）方法。

Q8.notify()和 notifyAll()有什么区别?

先说两个概念：锁池和等待池

- 锁池:假设线程A已经拥有了某个对象(注意:不是类)的锁，而其它的线程想要调用这个对象的某个synchronized方法(或者synchronized块)，由于这些线程在进入对象的synchronized方法之前必须先获得该对象的锁的拥有权，但是该对象的锁目前正被线程A拥有，所以这些线程就进入了该对象的锁池中。
- 等待池:假设一个线程A调用了某个对象的wait()方法，线程A就会释放该对象的锁后，进入到了该对象的等待池中

然后再来说notify和notifyAll的区别

- 如果线程调用了对象的 wait()方法，那么线程便会处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。
- 当有线程调用了对象的 notifyAll()方法（唤醒所有 wait 线程）或 notify()方法（只随机唤醒一个 wait 线程），被唤醒的的线程便会进入该对象的锁池中，锁池中的线程会去竞争该对象锁。也就是说，调用了notify后只要一个线程会由等待池进入锁池，而notifyAll会将该对象等待池内的所有线程移动到锁池中，等待锁竞争。
- 优先级高的线程竞争到对象锁的概率大，假若某线程没有竞争到该对象锁，它还会留在锁池中，唯有线程再次调用 wait()方法，它才会重新回到等待池中。而竞争到对象锁的线程则继续往下执行，直到执行完了 synchronized 代码块，它会释放掉该对象锁，这时锁池中的线程会继续竞争该对象锁。

Q9.线程的 run()和 start()有什么区别?

每个线程都有要执行的任务。线程的任务处理逻辑可以在Tread类的run实例方法中直接实现或通过该方法进行调用，因此run()相当于线程的任务处理逻辑的入口方法，它由Java虚拟机在运行相应线程时直接调用，而不是由应用代码进行调用。

而start()的作用是启动相应的线程。启动一个线程实际是请求Java虚拟机运行相应的线程，而这个线程何时能够运行是由线程调度器决定的。start()调用结束并不表示相应线程已经开始运行，这个线程可能稍后运行，也可能永远也不会运行。

Q10.创建线程池有哪几种方式?

newSingleThreadExecutor

创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

newFixedThreadPool

创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

newCachedThreadPool

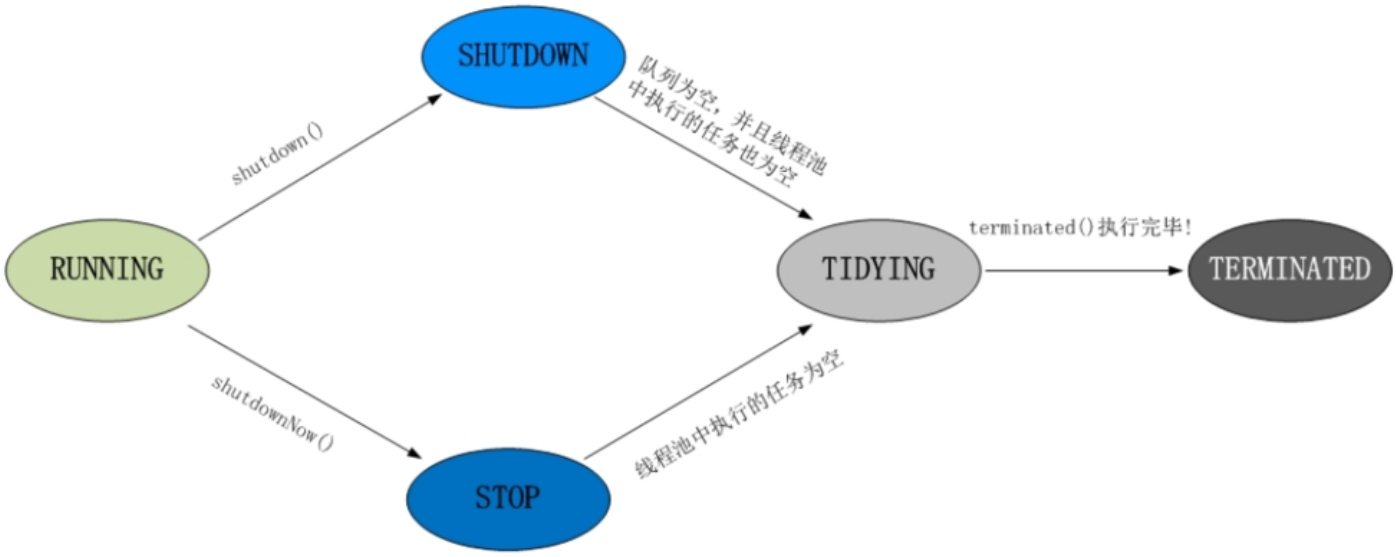
创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，

那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。

newScheduledThreadPool

创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。

Q11.线程池都有哪些状态?



1、 RUNNING
(1) 状态说明：线程池处在RUNNING状态时，能够接收新任务，以及对已添加的任务(堵塞队列中的任务)进行处理。
(02) 状态切换：线程池的初始化状态是RUNNING。换句话说，线程池被一旦被创建，就处于RUNNING状态，并且线程池中的任务数为0!
2、 SHUTDOWN
(1) 状态说明：线程池处在SHUTDOWN状态时，不接收新任务，但能处理已添加的任务。
(2) 状态切换：调用线程池的shutdown()接口时，线程池由RUNNING -> SHUTDOWN。
3、 STOP
(1) 状态说明：线程池处在STOP状态时，不接收新任务，不处理已添加的任务，并且会中断正在处理的任务。
(2) 状态切换：调用线程池的shutdownNow()接口时，线程池由(RUNNING or SHUTDOWN) -> STOP。
4、 TIDYING
(1) 状态说明：当所有的任务已终止，ctl(有效线程数)记录的"任务数量"为0，线程池会变为TIDYING状态。当线程池变为TIDYING状态时，会执行钩子函数terminated()。terminated()在ThreadPoolExecutor类中是空的，若用户想在线程池变为TIDYING时，进行相应的处理；可以通过重载terminated()函数来实现。
(2) 状态切换：当线程池在SHUTDOWN状态下，阻塞队列为空并且线程池中执行的任务也为空时，就会由 SHUTDOWN -> TIDYING。
当线程池在STOP状态下，线程池中执行的任务为空时，就会由STOP -> TIDYING。
5、 TERMINATED
(1) 状态说明：线程池彻底终止，就变成TERMINATED状态。
(2) 状态切换：线程池处在TIDYING状态时，执行完terminated()之后，就会由 TIDYING -> TERMINATED。

Q12.线程池中 submit()和 execute()方法有什么区别?

线程池中的execute方法开启线程执行池中的任务。还有一个方法submit也可以做到，它的功能是提交指定的任务去执行并且返回Future对象，即执行的结果。
两者的三个区别：
1、接收的参数不一样
2、submit有返回值，而execute没有用到返回值的。比如说我有很多个做validation的task，我希望所有的task执行完，然后每个task告诉我它的执行结果，是成功还是失败，如果是失败，原因是什么。然后我就可以把所有失败的原因综合起来发给调用者。
3、submit方便Exception处理
意思就是task里会抛出checked或者unchecked exception，
希望外面的调用者能够感知这些exception并做出及时的处理，那么就需要用到submit，通过捕获Future.get抛出的异常。

Q13.在 Java 程序中怎么保证多线程的运行安全?

线程安全在三个方面体现

1. 原子性：提供互斥访问，同一时刻只能有一个线程对数据进行操作，（atomic,synchronized）；
2. 可见性：一个线程对主内存的修改可以及时地被其他线程看到，（synchronized,volatile）；
3. 有序性：一个线程观察其他线程中的指令执行顺序，由于指令重排序，该观察结果一般杂乱无序，（happens-before原则）。

原子性---atomic

- JDK里面提供了很多atomic类，AtomicInteger,AtomicLong,AtomicBoolean等等。
- 它们是通过CAS完成原子性。

原子性---synchronized
synchronized是一种同步锁，通过锁实现原子操作。JDK提供锁分两种：一种是synchronized，依赖JVM实现锁，因此在这个关键字作用对象的作用范围内是同一时刻只能有一个线程进行操作；另一种是LOCK，是JDK提供的代码层面的锁，依赖CPU指令，代表性的是ReentrantLock。

synchronized修饰的对象有四种：

（1）修饰代码块，作用于调用的对象；

（2）修饰方法，作用于调用的对象；

（3）修饰静态方法，作用于所有对象；

（4）修饰类，作用于所有对象。

可见性---volatile

对于可见性，JVM提供了synchronized和volatile。这里我们看volatile。
1）volatile的可见性是通过内存屏障和禁止重排序实现的。volatile会在写操作时，会在写操作后加一条store屏障指令，将本地内存中的共享变量值刷新到主内存，volatile在进行读操作时，会在读操作前加一条load指令，从内存中读取共享变量，但是volatile不是原子性的，进行++操作不是安全的。

volatile适用的场景：
既然volatile不适用于计数，那么volatile适用于哪些场景呢：

1. 对变量的写操作不依赖于当前值
2. 该变量没有包含在具有其他变量不变的式子中有序性

有序性
是指，在JMM中，允许编译器和处理器对指令进行重排序，但是重排序过程不会影响到单线程程序的执行，却会影响到多线程并发执行的正确性。可以通过volatile、synchronized、lock保证有序性。另外，JMM具有先天的有序性，即不需要通过任何手段就可以得到保证的有序性。这称为happens-before原则。如果两个操作的执行次序无法从happens-before原则推导出来，那么它们就不能保证它们的有序性。虚拟机可以随意地对它们进行重排序。

happens-before原则：

1.程序次序规则：在一个单独的线程中，按照程序代码书写的顺序执行。

2.锁定规则：一个unlock操作happen—before后面对同一个锁的lock操作。

3.volatile变量规则：对一个volatile变量的写操作happen—before后面对该变量的读操作。

4.线程启动规则：Thread对象的start()方法happen—before此线程的每一个动作。

5.线程终止规则：线程的所有操作都happen—before对此线程的终止检测，可以通过Thread.join()方法结束、Thread.isAlive()的返回值等手段检测到线程已经终止执行。

6.线程中断规则：对线程interrupt()方法的调用happen—before发生于被中断线程的代码检测到中断时事件的发生。

7.对象终结规则：一个对象的初始化完成（构造函数执行结束）happen—before它的finalize()方法的开始。

8.传递性：如果操作A happen—before操作B，操作B happen—before操作C，那么可以得出A happen—before操作C。

Q14.多线程锁的升级原理是什么？

先说说为什么会有锁升级
因为Sycronized是重量级锁（也是悲观锁），每次在要进行锁的请求的时候，如果当前资源被其他线程占有要将当前的线程阻塞加入到阻塞队列，然后清空当前线程的缓存，等到锁释放的时候再通过notify或者notifyAll唤醒当前的线程，并让其处于就绪状态。这样线程的来回切换是非常消耗系统资源的，而且有的时候，线程刚挂起资源就释放了。而Java的线程是映射到操作系统的原生线程之上的每次线程的阻塞或者唤醒都要经过用户态到核心态或者核心态到用户态的转化，这样是十分浪费资源的，这样就会造成性能上的降低，因此JVM对Sychronized进行了优化，将Sycronized分为三种锁的级别：偏向锁，轻量级锁，重量级锁。

很多的时候，对于一个可能发生并发访问的对象而言，其实很少会被竞争，就算有些资源存在竞争也是在很少的一段时间资源就会被释放，而这样的情况下将线程挂起是十分浪费性能的。

偏向锁（乐观锁）：
当锁对象第一次被线程获取的时候，虚拟机会将锁对象的对象头中的锁标志位设置成为01，并将偏向锁标志设置为1，线程通过CAS的方式将自己的ID值放置到对象头中（因为在这个过程中有可能会有其他线程来竞争锁，所以要通过CAS的方式，一旦有竞争就会升级为轻量级锁了），如果成功线程就获得了该轻量级锁。这样每次再进入该锁对象的时候不用进行任何的同步操作，直接比较当前锁对象的对象头是不是该线程的ID，如果是就可以直接进入。

偏向锁升级为轻量级锁
偏向锁是一种无竞争锁，一旦出现了竞争大多数情况下就会升级为轻量级锁。现在我们假设有线程1持有偏向锁，线程2来竞争偏向锁会经历以下几个过程：

1. 首先线程2会先检查偏向锁标记，如果是1，说明当前是偏向锁，那么JVM会找到线程1，看线程1是否还存活着2
2. 如果线程1已经执行完毕，就是说线程1不存在了（线程1自己是不会去释放偏向锁的），那么先将偏向锁置为0，对象头设置成为无锁的状态，用CAS的方式尝试将线程2的ID放入对象头中，不进行锁升级，还是偏向锁
3. 如果线程1还活着，先暂停线程1，将锁标志位变成00（轻量级锁）然后在线程1的栈帧中开辟出一块空间（Display Mark Word）将对象头的Mark Word置换到线程一的栈帧当中，而对象头中此时存储的是指向当前线程栈帧的指针。此时就变成了轻量级锁。继续执行线程1，然后线程2采用CAS的方式尝试获取锁。

轻量级锁与偏向锁最大的不同之处
轻量级锁和偏向锁的不同之处就在于轻量级锁对于获取锁对象采用CAS的同步方式而偏向锁直接是把整个同步过程给取消。

轻量级锁(乐观锁)
轻量级锁如何创建在上面已经讲过了，接下来说说轻量级锁如何获取锁对象，轻量级锁是通过CAS也就是自旋的方式尝试获取锁对象，一旦失败会先检查，对象头中存储的是否是指向当前线程栈帧的指针，如果是，就可以获取对象，如果不是说明存在竞争那么就要膨胀为重量级锁。轻量级锁的解锁也是通过CAS的方式尝试将对象头的Mark Word和线程中的Display Mark Word替换回来，如果成功，就释放锁，如果失败说明还有许多其他等待锁的线程（说明此时已经不是轻量级锁而是重量级锁了），会将这些线程唤醒，然后释放锁。

轻量级锁膨胀为重量级锁
一旦有两条以上的线程竞争锁，轻量级锁膨胀为重量级锁，锁的状态变成10，此时对象头中存储的就是指向重量级锁的栈帧的指针。而且其他等待锁的线程要进入阻塞状态，等待重量级锁释放再来被唤醒然后去竞争。

Q15.什么是死锁？怎么防止死锁？

线程死锁是指由于两个或者多个线程互相持有对方所需要的资源，导致这些线程处于等待状态，无法前往执行。当线程进入对象的synchronized代码块时，便占有了资源，直到它退出该代码块或者调用wait方法，才释放资源，在此期间，其他线程将不能进入该代码块。当线程互相持有对方所需要的资源时，会互相等待对方释放资源，如果线程都不主动释放所占有的资源，将产生死锁。

当然死锁的产生是必须要满足一些特定条件的：

1. 互斥条件：进程对于所分配到的资源具有排它性，即一个资源只能被一个进程占用，直到被该进程释放
2. 请求和保持条件：一个进程因请求被占用资源而发生阻塞时，对已获得的资源保持不放。
3. 不剥夺条件：任何一个资源在没被该进程释放之前，任何其他进程都无法对他剥夺占用
4. 循环等待条件：当发生死锁时，所等待的进程必定会形成一个环路（类似于死循环），造成永久阻塞。

避免死锁

- 加锁顺序：当多个线程需要相同的一些锁，但是按照不同的顺序加锁，死锁就很容易发生。如果能确保所有的线程都是按照相同的顺序获得锁，那么死锁就不会发生。按照顺序加锁是一种有效的死锁预防机制。但是，这种方式需要你事先知道所有可能会用到的锁，并对这些锁做适当的排序，但总有些时候是无法预知的。
- 加锁时限：另外一个可以避免死锁的方法是在尝试获取锁的时候加一个超时时间，这也就意味着在尝试获取锁的过程中若超过了这个时限该线程则放弃对该锁请求。若一个线程没有在给定的时限内成功获得所有需要的锁，则会进行回退并释放所有已经获得的锁，然后等待一段随机的时间再重试。这段随机的等待时间让其它线程有机会尝试获取相同的这些锁，并且让该应用在没有获得锁的时候可以继续运行。加锁超时后可以先继续运行干点其它事情，再回头来重复之前加锁的逻辑。
- 死锁检测：是一个更好的死锁预防机制，它主要是针对那些不可能实现按序加锁并且锁超时也不可行的场景。每当一个线程获得了锁，会在线程和锁相关的数据结构中（map、graph等等）将其记下。除此之外，每当有线程请求锁，也需要记录在这个数据结构中。当一个线程请求锁失败时，这个线程可以遍历锁的关系图看看是否有死锁发生。

Q16.synchronized 和 volatile 的区别是什么?

一旦一个共享变量（类的成员变量、类的静态成员变量）被volatile修饰之后，那么就具备了两层语义：保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的；禁止进行指令重排序。

volatile本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。

1.volatile仅能使用在变量级别；synchronized则可以使用在变量、方法、和类级别的。

2.volatile仅能实现变量的修改可见性，并不能保证原子性；synchronized则可以保证变量的修改可见性和原子性。

3.volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞。

4.volatile标记的变量不会被编译器优化；synchronized标记的变量可以被编译器优化。

Q17.synchronized 和 Lock 有什么区别?

1.首先synchronized是java内置关键字，在jvm层面，Lock是个java类；

2.synchronized无法判断是否获取锁的状态，Lock可以判断是否获取到锁；

3.synchronized会自动释放锁(a 线程执行完同步代码会释放锁 ； b 线程执行过程中发生异常会释放锁)，Lock需在finally中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；

4.用synchronized关键字的两个线程1和线程2，如果当前线程1获得锁，线程2线程等待。如果线程1阻塞，线程2则会一直等待下去，而Lock锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了；

5.synchronized的锁可重入、不可中断、非公平，而Lock锁可重入、可判断、可公平（两者皆可）

6.Lock锁适合大量同步的代码的同步问题，synchronized锁适合代码少量的同步问题。

Q18.同步和异步有何异同，在什么情况下分别使用他们？举例说明。

如果数据将在线程间共享。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。

Q&S反射部分

Q1.什么是反射?

Java反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。
要想解剖一个类,必须先要获取到该类的字节码文件对象。而解剖使用的就是Class类中的方法.所以先要获取到每一个字节码文件对应的Class类型的对象.

Q2.什么是 java 序列化？什么情况下需要序列化？解释 Serializable 接口的作用

我们有时候将一个 java 对象变成字节流的形式传出去或者从一个字节流中恢复成一个 java 对象，例如，要将 java 对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个 java 对象变成某个格式的字节流再传输，但是jre 本身就提供了这种支持，我们可以调用 OutputStream 的 writeObject 方法来做，如果要让 java 帮我们做，要被传输的对象必须实现 serializable 接口，这样，javac 编译时就会进行特殊处理，编译的类才可以被 writeObject 方法操作，这就是所谓的序列化。需要被序列化的类必须实现 Serializable 接口，该接口是一个 mini 接口，其中没有需要实现的方法， implementsSerializable 只是为了标注该对象是可被序列化的。

Q3.动态代理是什么？有哪些应用？怎么实现动态代理?

动态代理：当想要给实现了某个接口的类中的方法，加一些额外的处理。比如说加日志，加事务等。可以给这个类创建一个代理，故名思议就是创建一个新的类，这个类不仅包含原来类方法的功能，而且还在原来的基础上添加了额外处理的新类。这个代理类并不是定义好的，是动态生成的。具有解耦意义，灵活，扩展性强。

动态代理实现：首先必须定义一个接口，还要有一个InvocationHandler(将实现接口的类的对象传递给它)处理类。再有一个工具类Proxy(习惯性将其称为代理类，因为调用他的newInstance()可以产生代理对象,其实他只是一个产生代理对象的工具类）。利用到InvocationHandler，拼接代理类源码，将其编译生成代理类的二进制码，利用加载器加载，并将其实例化产生代理对象，最后返回。

动态代理的应用：Spring的AOP，加事务，加权限，加日志。

Q&S异常部分

Q1.throw 和 throws 的区别?

throws是用来声明一个方法可能抛出的所有异常信息，声明但是不处理，将异常往上传，谁调用就交给谁处理。例如，如果一个方法里面不想有任何的异常处理，则在没有任何代码进行异常处理的时候，必须对这个方法进行声明有可能产生的所有异常（其实就是，不想自己处理，那就交给别人吧，告诉别人我会出现什么异常，报自己的错，让别人处理去吧）。
而throw则是指抛出的一个具体的异常类型。就是自己进行异常处理，处理的时候有两种方式，要么自己捕获异常（也就是try catch进行捕捉），要么声明抛出一个异常（就是throws 异常~~）。throw一旦进入被执行，程序立即会转入异常处理阶段，后面的语句就不再执行，而且所在的方法不再返回有意义的值!

Q2.finally、finalize 有什么区别?

finally作为异常处理的一部分，它只能用在try/catch语句中，并且附带一个语句块，表示这段语句最终一定会被执行（不管有没有抛出异常），经常被用在需要释放资源的情况下。（×）（这句话其实存在一定的问题）
很多人都认为finally语句块一定会执行，但真的是这样么？答案是否定的。只有与finally对应的try语句块得到执行的情况下，finally语句块才会执行。但是，在某些情况下，即使try语句执行了，finally语句也不一定执行。在 try 语句块中执行了 System.exit (0) 语句，终止了 Java 虚拟机的运行。那有人说了，在一般的 Java 应用中基本上是不会调用这个 System.exit(0) 方法的。那么不调用 System.exit(0) 这个方法，那么 finally 语句块就一定会执行吗？答案还是否定的。当一个线程在执行 try 语句块或者 catch 语句块时被打断（interrupted）或者被终止（killed），与其相对应的 finally 语句块可能不会执行。还有更极端的情况，就是在线程运行 try 语句块或者 catch 语句块时，突然死机或者断电，finally 语句块肯定不会执行了。

但是，在某些情况下，即使try语句执行了，finally语句也不一定执行。
Q3.try-catch-finally 中哪个部分可以省略?

```
try-catch
try-finally
try-catch-finally
```

catch和finally不能同时省略。

Q4.常见的异常类有哪些?

异常类	说明
NullPointerException	访问Null对象的方法
IllegalArgumentException	接收非法参数
ClassNotFoundException	不能加载所需要的类
ArithmeticException	算术运算异常，如除数为零
ArrayIndexOutOfBoundsException	数组小于或大于数组的长度
InputMisMatchException	接收的数据类型与实际输入的类型不匹配
NumberFormatException	格式化数据异常
IOException	文件读写异常

Q5.运行时异常与一般异常有何异同?

一般异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。

Java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

Q6.error 和 exception 有什么区别?

Java 对异常进行了分类，不同类型的异常分别用不同的 Java 类表示，所有异常的根类为 java.lang.Throwable，Throwable 下面又派生了两个子类：Error 和 Exception。

Error 表示应用程序本身无法克服和恢复的一种严重问题。
Exception 表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常，系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉，例如，数组脚本越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；
普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。 java 为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须 try..catch 处 理或用 throws 声明继续抛给上层调用方法处理，所以普通异常也称为 checked 异常，而系 统异常可以处理也可以不处理，所以，编译器不强制用 try..catch 处理或用 throws 声明，所以系统异常也称为 unchecked 异常。

Q6.请写出你最常见到的 3个 runtime exception。

NullPointerException、 ArrayIndexOutOfBoundsException、 ClassCastException。

Q&S对象拷贝部分

Q1.为什么要使用克隆?

想对一个对象进行处理，又想保留原有的数据进行接下来的操作，就需要克隆了。当我们new一个对象之后是要对该对象进行初始化的，不然这个对象是空的没有内容。而使用克隆，则会得到一个原对象以及原对象里面包含的内容。例如，你有一个User对象，里面的包含了相关的属性。此时你想要修改里面的某一属性，但又不想破坏原对象里面的数据，此时就可以克隆User这个对象，然后在克隆的这个User对象上进行修改操作。除此，如果你在操作完之后判断一下属性是否更改成功，则使用克隆的对象和原对象做一下对比即可。

Q2.如何实现对象克隆?

浅克隆就是把原对象中的一些属性值克隆过来。使用clone（）方法进行浅克隆。但注意：必须要在被克隆类上实现Cloneable接口，并重写clone方法。若不没有实现该接口，则会抛出CloneNotSupportedException异常！浅克隆只是克隆原对象中的引用类型指向，并非克隆了原对象中的全部数据。
深克隆和浅克隆的区别在于：浅克隆只克隆了原对象的引用类型的指向。深克隆则是克隆了原对象的所有。也就是说像上面案例所示，如果使两个对象之间互不影响，则使用深克隆。深克隆的使用：在引用类型所在的类使其实现Cloneable接口，并使用public修饰符重写Clone（）方法。

Q3.深拷贝和浅拷贝区别是什么?

克隆分浅克隆和深克隆，浅克隆后的对象中非基本对象和原对象指向同一块内存，因此对这些非基本对象的修改会同时更改克隆前后的对象。深克隆可以实现完全的克隆，可以用反射的方式或序列化的方式实现。