Java拓展常用类

1、Object类

toString()

getClass()

equals()

clone()

finalize()
其中toString()，getClass()，equals是其中最重要的方法

Object类中getClass()，notify()，notifyAll()，wait()等方法被定义为final类型，因此不能重写。

1、clone() 方法

那么在java语言中，有几种方式可以创建对象呢?

1、使用new操作符创建一个对象

2、使用clone方法复制一个对象

new 操作符执行时，先看new后面的类型，才知道要分配多大的空间，分配完内存之后，再调用构造函数，填充对象的各个域，这一步叫做初始化，构造方法返回后，一个对象创建完 毕，可以把他的引用（地址）发布到外部，在外部就可以使用这个引用操纵这个对象。

而clone在第一步 是和new相似的， 都是分配内存，调用clone方法时，分配的内存和源对象（即调用clone方法的对象） 相同，然后再使用原对象中对应的各个域，填充新对象的域， 填充完成之后，clone方法返回，一个新 的相同的对象被创建，同样可以把这个新对象的引用发布到外部。

clone与copy的区别

copy中的操作可能会影响到原来的属性和方法，他们使用的是同一个对象

clone是新创建的一个对象，对其原来的对象没有影响

我们有时候不希望在 方法里将参数改变，这是就需要在类中复写clone方法（实现深复制）。

clone方法的保护机制

在Object中Clone()是被声明为protected的，这样做是有一定的道理的，通过声明为protected，就可以保证只有类里面才能“克隆”对象.

clone方法的使用

什么时候使用shallow Clone，什么时候使用deep Clone，这个主要看具体对象的域是什么性质的，基本型别还是reference variable

调用Clone()方法的对象所属的类(Class)必须implements Clonable接口，否则在调用Clone方法的时候会抛出CloneNotSupportedException

2、toString()方法

Object 类的 toString 方法返回一个字符串，该字符串由类名（对象是该类的一个实例）、at 标记符“@”和此对象哈希码的无符号十六进制表示组成。

3、getClass()方法

返回次Object的运行时类类型。 不可重写，要调用的话，一般和getName()联合使用，如getClass().getName();

4、finalize()方法

该方法用于释放资源。因为无法确定该方法什么时候被调用，很少使用。

5、equals()方法

Object中的equals方法是直接判断this和obj本身的值是否相等，即用来判断调用equals的对象和形参obj所引用的对象是否是同一对象，

所谓同一对象就是指内存中同一块存储单元，如果this和obj指向的是同一块内存对象，则返回true,如果this和obj指向的不是同一块内存，则返回false。

注意：即便是内容完全相等的两块不同的内存对象，也返回false。

如果是同一块内存，则object中的equals方法返回true,如果是不同的内存，则返回false

6、hashCode()方法

返回该对象的哈希码值。

一般必须满足obj1.equals(obj2)==true。可以推出obj1.hash Code() == obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

7、wait()方法

wait方法就是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。

wait()方法一直等待，直到获得锁或者被中断。wait(long timeout)设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生。

（1）其他线程调用了该对象的notify方法。

（2）其他线程调用了该对象的notifyAll方法。

（3）其他线程调用了interrupt中断该线程。

（4）时间间隔到了。

此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException异常。

8、notify()方法

```
public final native void notify();
```

该方法唤醒在该对象上等待的某个线程。

```
public final native void notifyAll();
```

该方法唤醒在该对象上等待的所有线程。

2、包装类

1、包装类介绍

要转换为 String 类型（经常有这种需要）时只要简单调用 Object 类中定义的toString()即可，而基本数据 类型转换为 String 类型则要麻烦得多。

基本类型和对应的包装类可以相互装换：

由基本类型向对应的包装类转换称为装箱，例如把 int 包装成 Integer 类的对象；

包装类向对应的基本类型转换称为拆箱，例如把 Integer 类的对象重新简化为 int。

2、包装类的应用

1、 实现 int 和 Integer 的相互转换

可以通过 Integer 类的构造方法将 int 装箱，通过 Integer 类的 intValue 方法将 Integer 拆箱。

2、 将字符串转换为整数

Integer 类有一个静态的 paseInt() 方法，可以将字符串转换为整数，

3、将整数转换为字符串

Integer 类有一个静态的 toString() 方法，可以将整数转换为字符串。或者直接在整数后面加空字符串即 可！

3、自动拆箱和装箱

```
int m = 500;
Integer obj = m; // 自动装箱
int n = obj; // 自动拆箱
```

3、Math类

Java 的 Math 包含了用于执行基本数学运算的属性和方法，如初等指数、对数、平方根和三角函数。Math 的方法都被定义为 static 形式，通过 Math 类可以在主函数中直接调用。

Math.PI 记录的圆周率

Math.E 记录e的常量 Math中还有一些类似的常量，都是一些工程数学常用量。

Math.abs 求绝对值 Math.sin 正弦函数

Math.asin 反正弦函数

Math.cos 余弦函数

Math.acos 反余弦函数

Math.tan 正切函数

Math.atan 反正切函数

Math.atan2 商的反正切函数

Math.toDegrees 弧度转化为角度

Math.toRadians 角度转化为弧度

Math.ceil 得到不小于某数的最大整数

Math.floor 得到不大于某数的最大整数

Math.IEEEremainder 求余

Math.max 求两数中最大

Math.min 求两数中最小

Math.sqrt 求开方

Math.pow 求某数的任意次方, 抛出ArithmeticException处理溢出异常

Math.exp 求e的任意次方

Math.log10 以10为底的对数

Math.log 自然对数

Math.rint 求距离某数最近的整数（可能比某数大，也可能比它小）

Math.round 同上，返回int型或者long型（上一个函数返回double型）

Math.random 返回0，1之间的一个随机数

4、Random类

Java中存在着两种Random函数：

1、java.lang.Math.Random

调用这个Math.Random()函数能够返回带正号的double值，该值大于等于0.0且小于1.0，即取值范围是[0.0,1.0)的左闭右开区间，返回值是一个伪随机选择的数，在该范围内（近似）均匀分布。

2、java.util.Random

下面是Random()的两种构造方法：

Random()：创建一个新的随机数生成器。

Random(long seed)：使用单个 long 种子创建一个新的随机数生成器。

5、日期时间类

1、Date类

java.util 包提供了 Date 类来封装当前的日期和时间。

Java中获取当前日期和时间很简单，使用 Date 对象的 toString() 方法来打印当前日期和时间

2、SimpleDateFormat

SimpleDateFormat 是一个以语言环境敏感的方式来格式化和分析日期的类。SimpleDateFormat 允许你选择任何用户自定义日期时间格式来运行。

3、Calendar类

6、String类

1、String概述

String底层使用一个字符数组来维护的

2、创建字符串对象方式

直接赋值方式创建对象是在方法区的常量池

通过构造方法创建字符串对象是在堆内存

两种实例化方式的区别

1. 直接赋值（String str = "hello"）：只开辟一块堆内存空间，并且会自动入池，不会产生垃圾。
2. 构造方法（String str= new String("hello");）:会开辟两块堆内存空间，其中一块堆内存会变成垃圾被系统回收，而且不能够自动入池，需要通过public String intern();方法进行手工入池。
3. 在开发的过程中不会采用构造方法进行字符串的实例化。

首先了解： == 和public boolean equals()比较字符串的区别

==在对字符串比较的时候，对比的是**内存地址**，而equals比较的是**字符串内容**，在开发的过程中，equals()通过接受参数，可以避免空指向。

3、String常用的方法

4、String的判断

```
boolean equals(Object obj)：比较字符串的内容是否相同
boolean equalsIgnoreCase(String str)：  比较字符串的内容是否相同,忽略大小写
boolean startsWith(String str)：  判断字符串对象是否以指定的str开头
boolean endsWith(String str)：  判断字符串对象是否以指定的str结尾
```

5、String的截取

```
int length():获取字符串的长度，其实也就是字符个数
char charAt(int index):获取指定索引处的字符
int indexOf(String str):获取str在字符串对象中第一次出现的索引
String substring(int start):从start开始截取字符串
String substring(int start,int end):从start开始，到end结束截取字符串。包括start，不包括end
```

6、String的转换

```
char[] toCharArray()：把字符串转换为字符数组
String toLowerCase()：把字符串转换为小写字符串
String toUpperCase()：把字符串转换为大写字符串
```

7、其他方法

```
去除字符串两端空格：String trim()
按照指定符号分割字符串：String[] split(String str)
```

8、String的不可变性

String是个常量，从一出生就注定不可变。

在object中，equals()是用来比较内存地址的，但是String重写了equals()方 法，用来比较内容的，即使是不同地址，只要内容一致，也会返回true

String不可变的好处

可以实现多个变量引用堆内存中的同一个字符串实例，避免创建的开销。

我们的程序中大量使用了String字符串，有可能是出于安全性考虑。 大家都知道HashMap中key为String类型，如果可变将变的多么可怕。

当我们在传参的时候，使用不可变类不需要去考虑谁可能会修改其内部的值，如果使用可变类的 话，可能需要每次记得重新拷贝出里面的值，性能会有一定的损失。

9、字符串常量池

享元模式，提高的内存利用效率

使用String s = new String("hello");会创建几个对象

答：会创建2个对象 首先，出现了字面量"hello"，那么去String Pool中查找是否有相同字符串存在，因为程序就这一行 代码所以肯定没有，那么就在Java Heap中用字面量"hello"首先创建1个String对象。 接着，new String("hello")，关键字new又在Java Heap中创建了1个对象，然后调用接收String 参数的构造器进行了初始化。最终s的引用是这个String对象

7、StringBuilder 和 StringBuffer

1、概述

StringBuilder 是一个可变的字符序列。它继承于AbstractStringBuilder，实现了CharSequence接口。StringBuffer 也是继承于AbstractStringBuilder的子类；但是，StringBuilder和StringBuffer不同，前者是非线程安全的，后者是线程安全的。

2、常用方法

```
StringBuilder sbuilder = new StringBuilder();
// 在位置0处插入字符数组
sbuilder.insert(0, new char[]{'a', 'b', 'c', 'd', 'e'});
// 在位置0处插入字符数组。0表示字符数组起始位置，3表示长度
sbuilder.insert(0, new char[]{'A', 'B', 'C', 'D', 'E'}, 0, 3);
// 在位置0处插入float
sbuilder.insert(0, 1.414f);
// 在位置0处插入double
sbuilder.insert(0, 3.14159d);
// 在位置0处插入boolean
sbuilder.insert(0, true);
// 在位置0处插入char
sbuilder.insert(0, '\n');
// 在位置0处插入int
```

```
sbuilder.insert(0, 100);
// 在位置0处插入long
sbuilder.insert(0, 12345L);
// 在位置0处插入StringBuilder对象
sbuilder.insert(0, new StringBuilder("StringBuilder"));
// 在位置0处插入StringBuilder对象。6表示被在位置0处插入对象的起始位置(包括)，13是结束位置(不包括)
sbuilder.insert(0, new StringBuilder("STRINGBUILDER"), 6, 13);
// 在位置0处插入StringBuffer对象。
sbuilder.insert(0, new StringBuffer("StringBuffer"));
// 在位置0处插入StringBuffer对象。6表示被在位置0处插入对象的起始位置(包括)，12是结束位置(不包括)
sbuilder.insert(0, new StringBuffer("STRINGBUFFER"), 6, 12);
// 在位置0处插入String对象。
sbuilder.insert(0, "String");
// 在位置0处插入String对象。1表示被在位置0处插入对象的起始位置(包括)，6是结束位置(不包括)
sbuilder.insert(0, "0123456789", 1, 6);
sbuilder.insert(0, '\n');
// 在位置0处插入Object对象。此处以HashMap为例
HashMap map = new HashMap();
map.put("1", "one");
map.put("2", "two");
map.put("3", "three");
sbuilder.insert(0, map);
System.out.printf("%s\n\n", sbuilder);
```

3、replace

```
/**
* StringBuilder 的replace()示例
*/
private static void testReplaceAPIs() {
System.out.println("------------------- testReplaceAPIs-----------------
--");
StringBuilder sbuilder;
sbuilder = new StringBuilder("0123456789");
sbuilder.replace(0, 3, "ABCDE");
System.out.printf("sbuilder=%s\n", sbuilder);
sbuilder = new StringBuilder("0123456789");
sbuilder.reverse();//倒叙
System.out.printf("sbuilder=%s\n", sbuilder);
sbuilder = new StringBuilder("0123456789");
sbuilder.setCharAt(0, 'M');
System.out.printf("sbuilder=%s\n", sbuilder);
System.out.println();
```

4、delete

```
/**
* StringBuilder 中delete相关API演示
*/
StringBuilder sbuilder = new StringBuilder("0123456789");
// 删除位置0的字符，剩余字符是“123456789”。
sbuilder.deleteCharAt(0);
// 删除位置3(包括)到位置6(不包括)之间的字符，剩余字符是“123789”。
sbuilder.delete(3,6);
```

```
// 获取sb中从位置1开始的字符串
String str1 = sbuilder.substring(1);
// 获取sb中从位置3(包括)到位置5(不包括)之间的字符串
String str2 = sbuilder.substring(3, 5);
// 获取sb中从位置3(包括)到位置5(不包括)之间的字符串，获取的对象是CharSequence对象，此处转型
为String
String str3 = (String)sbuilder.subSequence(3, 5);
System.out.printf("sbuilder=%s\nstr1=%s\nstr2=%s\nstr3=%s\n",
sbuilder, str1, str2, str3);
```

5、index

```
/**
* StringBuilder 中index相关API演示
*/
StringBuilder sbuilder = new StringBuilder("abcAbcABCabCaBcAbCaBCabc");
System.out.printf("sbuilder=%s\n", sbuilder);
// 1. 从前往后，找出"bc"第一次出现的位置
System.out.printf("%-30s = %d\n", "sbuilder.indexOf(\"bc\")",
sbuilder.indexOf("bc"));
// 2. 从位置5开始，从前往后，找出"bc"第一次出现的位置
System.out.printf("%-30s = %d\n", "sbuilder.indexOf(\"bc\", 5)",
sbuilder.indexOf("bc", 5));
// 3. 从后往前，找出"bc"第一次出现的位置
System.out.printf("%-30s = %d\n", "sbuilder.lastIndexOf(\"bc\")",
sbuilder.lastIndexOf("bc"));
// 4. 从位置4开始，从后往前，找出"bc"第一次出现的位置
System.out.printf("%-30s = %d\n", "sbuilder.lastIndexOf(\"bc\", 4)",
sbuilder.lastIndexOf("bc", 4));
System.out.println();
```

6、其他API

```
System.out.println("----------- testOtherAPIs -----------");
StringBuilder sbuilder = new StringBuilder("0123456789");
int cap = sbuilder.capacity();
System.out.printf("cap=%d\n", cap);
/*
capacity()返回的是字符串缓冲区的容量
StringBuffer( ); 分配16个字符的缓冲区
StringBuffer( int len ); 分配len个字符的缓冲区
StringBuffer( String s ); 除了按照s的大小分配空间外,再分配16个 字符的缓冲区
你的StringBuffer是用字符构造的，"0123456789"的长度是10另外再分配16个字符，所
以一共是26。
*/
char c = sbuilder.charAt(6);
System.out.printf("c=%c\n", c);
char[] carr = new char[4];
sbuilder.getChars(3, 7, carr, 0);
for (int i=0; i<carr.length; i++){
System.out.printf("carr[%d]=%c ", i, carr[i]);
}
System.out.println()
```

7、StringBuffer

和StringBulider用法差不多，不过多介绍，主要看一下三者的区别

8、小结

【String、StringBuffer、StringBuilder之间的区别】

首先需要说明的是：

String 字符串常量 StringBuffer 字符串变量（线程安全）

StringBuilder 字符串变量（非线程安全）

在大多数情况下三者在执行速度方面的比较：StringBuilder > StringBuffer > String

对于三者使用的总结：

1）如果要操作少量的数据用 = String

2）单线程操作字符串缓冲区 下操作大量数据 = StringBuilder

3）多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

9、面试题的回答

StringBuilder 与StringBuffer的区别，StringBuilder与String的区别。

1）StringBuilder效率高，线程不安全，StringBuffer效率低，线程安全。

2）String是不可变字符串，StringBuilder是可变字符串。为什么有这样的差异，可以深入源码去解析，比如String类内的 priver final char value[] 等方法的原因。

3）如果是简单的声明一个字符串没有后续过多的操作，使用String,StringBuilder均可，若后续对字符穿做频繁的添加，删除操作,或者是在循环当中动态的改变字符穿的长度应该用StringBuilder。使用String会产生多余的字符串，占用内存空间。

8、File类