前言

ok,各位同学们,那么从今天开始呢，我们就正式开始迈入Java常用类的学习了.之前呢，我们已经学习了面向对象相关的知识，也算正式踏入了Java开发的大门了，那么后面的课程都属于是高级部分学习了！

我们会学习：常用类，集合框架，IO流，多线程，网络编程，注解和反射，以及GUI编程！

其实，Java的学习，除了思想，本质还是在学习一个个类的使用！

我们来看下，这一章会学习哪些内容呢？

首先，我们要学习Math类，这里面有许多关于数学操作的方法，然后我们会学习时间和日期类，使用Java来获得时间相关的对象，然后会给大家讲解String类，这个我们从学java的第一天就接触的类，我们其实并未真正的了解，包括它的一些扩展StringBuilder 和 StringBuffer，接下来就是我们的包装类了，之前学习的8大基本数据类型都有其对应的包装类，还有自动装箱和拆箱的原理，我们需要知道。然后就是所有类的老祖宗类：Object的探究及分析，最后给大家讲讲File类，为之后会学习的IO流打个基础！

这些就是我们这个阶段会学习的一些东西了！是一个很重要的阶段，以后开发中，这些都是天天用的东西，不能不会！

Object类

大家都知道Object是所有类的父类，任何类都默认继承Object。

理论上Object类是所有类的父类，即直接或间接的继承java.lang.Object类。

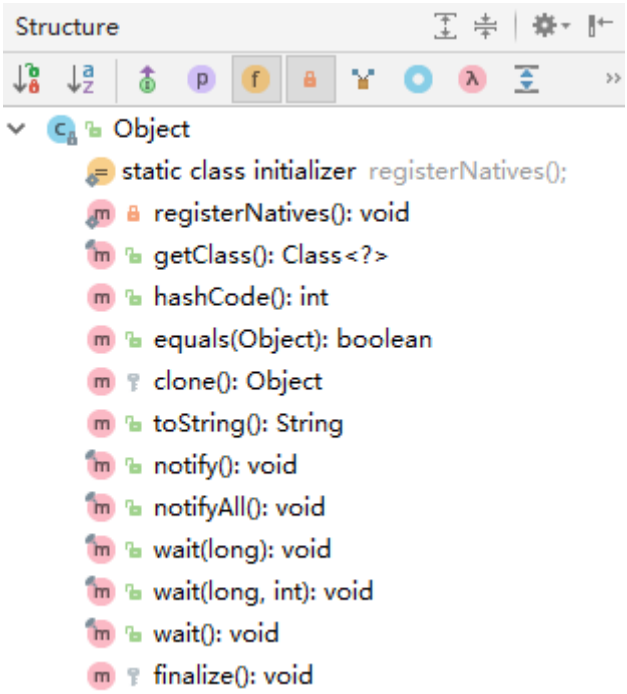
由于所有的类都继承在Object类，因此省略了extends Object关键字。

该类中主要有以下方法:

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

【演示：查看Object类源码】

狂神社群笔记资料，禁止外传，本人QQ：24736743



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

1、clone() 方法

详解文章：https://blog.csdn.net/zhangjg_blog/article/details/18369201#0-qzone-1-28144-d020d2d2a4e8d1a374a433f596ad1440

```
1  protected native Object clone() throws CloneNotSupportedException;
```

clone顾名思义就是复制， 在Java语言中， clone方法被对象调用，所以会复制对象。所谓的复制对象，首先要分配一个和源对象同样大小的空间，在这个空间中创建一个新的对象。那么在java语言中，有几种方式可以创建对象呢?

- 使用new操作符创建一个对象
- 使用clone方法复制一个对象

那么这两种方式有什么相同和不同呢？ new操作符的本意是分配内存。程序执行到new操作符时， 首先去看new操作符后面的类型，因为知道了类型，才能知道要分配多大的内存空间。分配完内存之后，再调用构造函数，填充对象的各个域，这一步叫做对象的初始化，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

假设现在有一个Employee对象，Employee tobby =new Employee(“CMTobby”,5000)

通常我们会有这样的赋值Employee cindyelf=tobby，这个时候只是简单了copy了一下reference，cindyelf和tobby都指向内存中同一个object，这样cindyelf或者tobby的一个操作都可能影响到对方。打个比方，如果我们通过cindyelf.raiseSalary()方法改变了salary域的值，那么tobby通过getSalary()方法得到的就是修改之后的salary域的值，显然这不是我们愿意看到的。我们希望得到tobby的一个精确拷贝，同时两者互不影响，这时候， 我们就可以使用Clone来满足我们的需求。Employee cindy=tobby.clone()，这时会生成一个新的Employee对象，并且和tobby具有相同的属性值和方法。

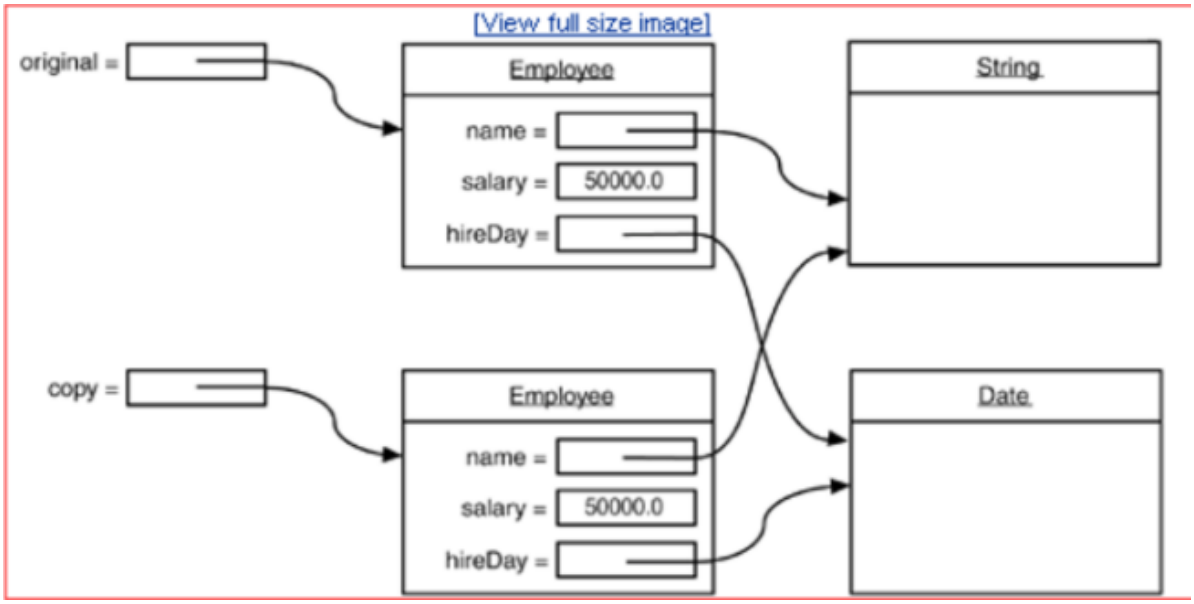
Shallow Clone与Deep Clone

主要是JAVA里除了8种基本类型传参数是值传递，其他的类对象传参数都是引用，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是如何完成的呢？Object在对某个对象实施Clone时对其是一无所知的，它仅仅是简单地执行域对域的copy，这就是Shallow Clone。这样，问题就来了咯。

以Employee为例，它里面有一个域hireDay不是基本数据类型的变量，而是一个reference变量，经过Clone之后就会产生一个新的Date型的reference，

它和原始对象中对应的域指向同一个Date对象，这样克隆类就和原始类共享了一部分信息，而这样显然是不利的，过程下图所示：



这个时候我们就需要进行deep Clone了，对那些非基本类型的域进行特殊的处理，例如本例中的hireDay。我们可以重新定义Clone方法，对hireDay做特殊处理，如下代码所示：

```
class Employee implements Cloneable  {
    public Object clone() throws CloneNotSupportedException  {
        Employee cloned = (Employee) super.clone();
        cloned.hireDay = (Date) hireDay.clone()
        return cloned;
    }
}
```

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，以Employee类为例，通过声明为protected，就可以保证只有Employee类里面才能“克隆”Employee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

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

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。

该方法用得比较多，**一般子类都有覆盖。**

我们推荐在学习阶段所有有属性的类都加上toString() 方法！

```
public static void main(String[] args){
    Object o1 = new Object();
    System.out.println(o1.toString());
}
```

3、getClass()方法

```
public final native Class<?> getClass();
```

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();

```
public static void main(String[] args) {
    Object o = new Object();
    System.out.println(o.getClass());
    //class java.lang.Object
}
```

4、finalize()方法

```
protected void finalize() throws Throwable { }
```

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。

Java允许在类中定义一个名为finalize()的方法。它的工作原理是：一旦垃圾回收器准备好释放对象占用的存储空间，将首先调用其finalize()方法。并且在下一次垃圾回收动作发生时，才会真正回收对象占用的内存。

关于垃圾回收，有三点需要记住：

1、对象可能不被垃圾回收。只要程序没有濒临存储空间用完的那一刻，对象占用的空间就总也得不到释放。

2、垃圾回收并不等于“析构”。

【科普：析构函数(destructor) 与构造函数相反，当对象结束其生命周期，如对象所在的函数已调用完毕时，系统自动执行析构函数。析构函数往往用来做“清理善后” 的工作（例如在建立对象时用new开辟了一片内存空间，delete会自动调用析构函数后释放内存）。】

3、垃圾回收只与内存有关。使用垃圾回收的唯一原因是为了回收程序不再使用的内存。

finalize()的用途：

无论对象是如何创建的，垃圾回收器都会负责释放对象占据的所有内存。

这就将对finalize()的需求限制到一种特殊情况，即通过某种创建对象方式以外的方式为对象分配了存储空间。不过这种情况一般发生在使用“本地方法”的情况下，本地方法是一种在Java中调用非Java代码的方式。

5、equals()方法

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

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hi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

如果希望不同内存但相同内容的两个对象equals时返回true,则我们需要重写父类的equal方法

String类已经重写了object中的equals方法（这样就是比较内容是否相等了）

【演示：查看String类源码equals方法】

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
    return false;
}
```

6、hashCode()方法

```
public native int hashCode();
```

返回该对象的哈希码值。

该方法用于哈希查找，可以减少在查找中使用equals的次数，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到。

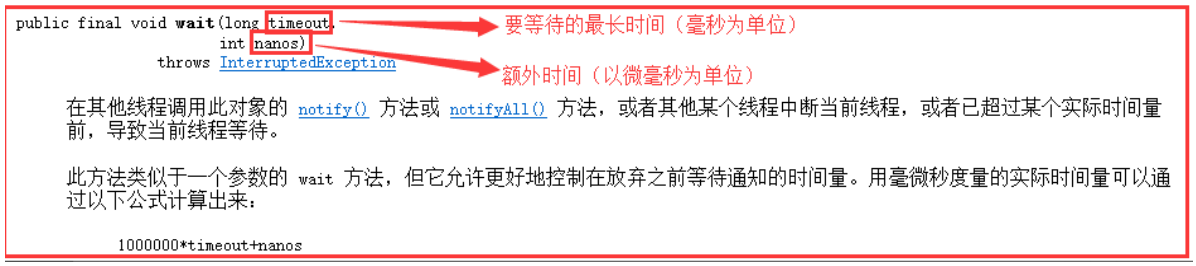
一般必须满足obj1.equals(obj2)==true。可以推出obj1.hash Code() == obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

7 wait()方法

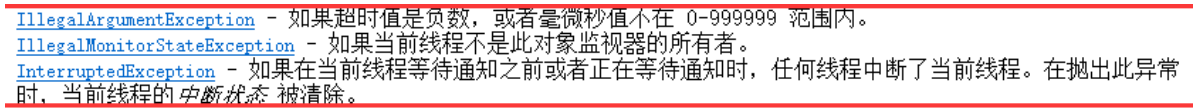
```
public final void wait() throws InterruptedException {
    wait(0);
}
```

```
4  public final native void wait(long timeout) throws InterruptedException;
5  public final void wait(long timeout, int nanos) throws InterruptedException
   {
6      if (timeout < 0) {
7          throw new IllegalArgumentException("timeout value is negative");
8      }
9
10     if (nanos < 0 || nanos > 999999) {
11         throw new IllegalArgumentException(
12                             "nanosecond timeout value out of range");
13     }
14
15     if (nanos > 0) {
16         timeout++;
17     }
18
19     wait(timeout);
20 }
```

可以看到有三种重载，wait什么意思呢?



方法中的异常：



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

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

该方法唤醒在该对象上等待的某个线程。

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

该方法唤醒在该对象上等待的所有线程。

包装类

1、包装类介绍

虽然 Java 语言是典型的面向对象编程语言，但其中的八种基本数据类型并不支持面向对象编程，基本类型的数据不具备“对象”的特性——不携带属性、没有方法可调用。 沿用它们只是为了迎合人类根深蒂固的习惯，并的确能简单、有效地进行常规数据处理。

这种借助于非面向对象技术的做法有时也会带来不便，比如引用类型数据均继承了 Object 类的特性，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。为解决此类问题 ，Java为每种基本数据类型分别设计了对应的类，称之为包装类(Wrapper Classes)，也有教材称为外覆类或数据类型类。

基本数据类型	对应的包装类
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

每个包装类的对象可以封装一个相应的基本类型的数据，并提供了其它一些有用的方法。包装类对象一经创建，其内容（所封装的基本类型数据值）不可改变。

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。

```
public static void main(String[] args) {
    int m = 500;
    Integer obj = new Integer(m); // 手动装箱
    int n = obj.intValue(); // 手动拆箱
    System.out.println("n = " + n);
    Integer obj1 = new Integer(500);
    System.out.println("obj 等价于 obj1? " + obj.equals(obj1));
}
```

【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，语法为：

```
parseInt(String s, int radix);
```

s 为要转换的字符串，radix 为进制，可选，默认为十进制。

下面的代码将会告诉你什么样的字符串可以转换为整数：

```
public static void main(String[] args) {
    String[] str = {"123", "123abc", "abc123", "abcxyz"};
    for(String str1 : str){
        try{
            int m = Integer.parseInt(str1, 10);
            System.out.println(str1 + " 可以转换为整数 " + m);
        }catch(Exception e){
            System.out.println(str1 + " 无法转换为整数");
        }
    }
}

//结果
123 可以转换为整数 123
123abc 无法转换为整数
abc123 无法转换为整数
abcxyz 无法转换为整数
```

【 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！

```
public static void main(String[] args) {
    int m = 500;
    String s = Integer.toString(m);
    String s2 = m+"";
    System.out.println("s = " + s);
}
```

3、自动拆箱和装箱

上面的例子都需要手动实例化一个包装类，称为手动拆箱装箱。Java 1.5(5.0) 之前必须手动拆箱装箱。

Java 1.5 之后可以自动拆箱装箱，也就是在进行基本数据类型和对应的包装类转换时，系统将自动进行，这将大大方便程序员的代码书写。

```
public static void main(String[] args) {
    int m = 500;
    Integer obj = m; // 自动装箱
    int n = obj; // 自动拆箱
    System.out.println("n = " + n);
    Integer obj1 = 500;
    System.out.println("obj 等价于 obj1? " + obj.equals(obj1));
}

//结果:
//    n = 500
//    obj 等价于 obj1? true
```

自动装箱与拆箱的功能事实上是编译器来帮您的忙，编译器在编译时期依您所编写的语法，决定是否进行装箱或拆箱动作。例如：

```
Integer i = 100;
相当于编译器自动为您作以下的语法编译：
Integer i = new Integer(100);
```

所以自动装箱与拆箱的功能是所谓的“编译器蜜糖”(Compiler Sugar)，虽然使用这个功能很方便，但在程序运行阶段您得了解Java的语义。例如下面的程序是可以通过编译的：

```
Integer i = null;
int j = i;
```

这样的语法在编译时期是合法的，但是在运行时期会有错误，因为这种写法相当于：

```
Integer i = null;
int j = i.intValue();
```

null表示i 没有参考至任何的对象实体，它可以合法地指定给对象参考名称。由于实际上i并没有参考至任何的对象，所以也就不可能操作intValue()方法，这样上面的写法在运行时会出现NullPointerException错误。

自动拆箱装箱是常用的一个功能，需要重点掌握。

一般地，当需要使用数字的时候，我们通常使用内置数据类型，如：**byte、int、long、double** 等。然而，在实际开发过程中，我们经常会遇到需要使用对象，而不是内置数据类型的情形。为了解决这个问题，Java 语言为每一个内置数据类型提供了对应的包装类。

所有的包装类（**Integer、Long、Byte、Double、Float、Short**）都是**抽象类 Number** 的子类。

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。

【演示：查看Math类的源码】

```
public final class Math{
    //数学方法
}
```

【常用值与函数】

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

狂神社群笔记资料，禁止外传，本人QQ：24736743

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

```
public static void main(String[] args) {
    /**
     *Math.sqrt()//计算平方根
     *Math.cbrt()//计算立方根
     *Math.pow(a, b)//计算a的b次方
     *Math.max( , );//计算最大值
     *Math.min( , );//计算最小值
     */
    System.out.println(Math.sqrt(16));  //4.0
    System.out.println(Math.cbrt(8));  //2.0
    System.out.println(Math.pow(3,2));   //9.0
    System.out.println(Math.max(2.3,4.5));//4.5
    System.out.println(Math.min(2.3,4.5));//2.3
    /**
     * abs求绝对值
     */
    System.out.println(Math.abs(-10.4));  //10.4
    System.out.println(Math.abs(10.1));   //10.1
    /**
     * ceil天花板的意思，就是返回大的值
     */
    System.out.println(Math.ceil(-10.1));  //-10.0
    System.out.println(Math.ceil(10.7));  //11.0
    System.out.println(Math.ceil(-0.7));  //-0.0
    System.out.println(Math.ceil(0.0));   //0.0
    System.out.println(Math.ceil(-0.0));  //-0.0
    System.out.println(Math.ceil(-1.7));  //-1.0
    /**
     * floor地板的意思，就是返回小的值
     */
    System.out.println(Math.floor(-10.1)); //-11.0
    System.out.println(Math.floor(10.7));  //10.0
    System.out.println(Math.floor(-0.7));  //-1.0
    System.out.println(Math.floor(0.0));  //0.0
    System.out.println(Math.floor(-0.0));  //-0.0
    /**
     * random 取得一个大于或者等于0.0小于不等于1.0的随机数 [0,1)
     */
    System.out.println(Math.random()); //小于1大于0的double类型的数
    System.out.println(Math.random()+1);//大于1小于2的double类型的数
    /**
     * rint 四舍五入，返回double值
     * 注意.5的时候会取偶数   异常的尴尬=。=
     */
    System.out.println(Math.rint(10.1));  //10.0
    System.out.println(Math.rint(10.7));  //11.0
    System.out.println(Math.rint(11.5));  //12.0
    System.out.println(Math.rint(10.5));  //10.0
    System.out.println(Math.rint(10.51));  //11.0
    System.out.println(Math.rint(-10.5));  //-10.0
```

```
51      System.out.println(Math.rint(-11.5));  //-12.0
52      System.out.println(Math.rint(-10.51)); //-11.0
53      System.out.println(Math.rint(-10.6));  //-11.0
54      System.out.println(Math.rint(-10.2));  //-10.0
55      /**
56       * round 四舍五入，float时返回int值，double时返回long值
57       */
58      System.out.println(Math.round(10.1));  //10
59      System.out.println(Math.round(10.7));  //11
60      System.out.println(Math.round(10.5));  //11
61      System.out.println(Math.round(10.51)); //11
62      System.out.println(Math.round(-10.5)); //-10
63      System.out.println(Math.round(-10.51)); //-11
64      System.out.println(Math.round(-10.6)); //-11
65      System.out.println(Math.round(-10.2)); //-10
66
67  }
```

Random类

Java中存在着两种Random函数：

一、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。例子如下：

```
1  public static void main(String[] args) {
2      // 结果是个double类型的值，区间为[0.0,1.0）
3      System.out.println("Math.random()=" + Math.random());
4      int num = (int) (Math.random() * 3);
5      // 注意不要写成(int)Math.random()*3，这个结果为0或1，因为先执行了强制转换
6      System.out.println("num=" + num);
7  }
8  //结果
9  //Math.random()=0.44938147153848396
10 //num=1
```

二、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。

你在创建一个Random对象的时候可以给定任意一个合法的种子数，种子数只是随机算法的起源数字，和生成的随机数的区间没有任何关系。

如下面的Java代码：

【演示一】

在没带参数构造函数生成的Random对象的种子缺省是当前系统时间的毫秒数。

rand.nextInt(100)中的100是随机数的上限,产生的随机数为0-100的整数,不包括100。

```
public static void main(String[] args) {
    Random rand =new Random();
    int i=rand.nextInt(100);
    System.out.println(i);
}
```

【演示二】

对于种子相同的Random对象，生成的随机数序列是一样的。

```
public static void main(String[] args) {
    Random ran1 = new Random(25);
    System.out.println("使用种子为25的Random对象生成[0,100)内随机整数序列: ");
    for (int i = 0; i < 10; i++) {
        System.out.print(ran1.nextInt(100) + " ");
    }
    System.out.println();
}
```

【方法摘要】

1. protected int next(int bits)：生成下一个伪随机数。
2. boolean nextBoolean()：返回下一个伪随机数，它是取自此随机数生成器序列的均匀分布的 boolean值。
3. void nextBytes(byte[] bytes)：生成随机字节并将其置于用户提供的 byte 数组中。
4. double nextDouble()：返回下一个伪随机数，它是取自此随机数生成器序列的、在0.0和1.0之间均匀分布的 double值。
5. float nextFloat()：返回下一个伪随机数，它是取自此随机数生成器序列的、在0.0和1.0之间均匀分布float值。
6. double nextGaussian()：返回下一个伪随机数，它是取自此随机数生成器序列的、呈高斯（“正态”）分布的double值，其平均值是0.0标准差是1.0。
7. int nextInt()：返回下一个伪随机数，它是此随机数生成器的序列中均匀分布的 int 值。
8. int nextInt(int n)：返回一个伪随机数，它是取自此随机数生成器序列的、在（包括和指定值（不包括）之间均匀分布的int值。
9. long nextLong()：返回下一个伪随机数，它是取自此随机数生成器序列的均匀分布的 long 值。
10. void setSeed(long seed)：使用单个 long 种子设置此随机数生成器的种子。

【例子】

1. 生成[0,1.0)区间的小数：double d1 = r.nextDouble();
2. 生成[0,5.0)区间的小数：double d2 = r.nextDouble() * 5;
3. 生成[1,2.5)区间的小数：double d3 = r.nextDouble() * 1.5 + 1;
4. 生成[0,10)区间的整数：int n2 = r.nextInt(10);

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。

Date 类提供两个构造函数来实例化 Date 对象。

第一个构造函数使用当前日期和时间来初始化对象。

```
Date()
```

第二个构造函数接收一个参数，该参数是从1970年1月1日起的毫秒数。

```
Date(long millisec)
```

Date对象创建以后，可以调用下面的方法。

序号	方法和描述
1	**boolean after(Date date)** 若当调用此方法的Date对象在指定日期之后返回true,否则返回false。
2	**boolean before(Date date)** 若当调用此方法的Date对象在指定日期之前返回true,否则返回false。
3	**Object clone( )** 返回此对象的副本。
4	**int compareTo(Date date)** 比较当调用此方法的Date对象和指定日期。两者相等时候返回0。调用对象在指定日期之前则返回负数。调用对象在指定日期之后则返回正数。
5	**int compareTo(Object obj)** 若obj是Date类型则操作等同于compareTo(Date) 。否则它抛出ClassCastException。
6	**boolean equals(Object date)** 当调用此方法的Date对象和指定日期相等时候返回true,否则返回false。
7	**long getTime( )** 返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。
8	**int hashCode( )** 返回此对象的哈希码值。
9	**void setTime(long time)** 用自1970年1月1日00:00:00 GMT以后time毫秒数设置时间和日期。
10	**String toString( )** 把此 Date 对象转换为以下形式的 String： dow mon dd hh:mm:ss zzz yyyy 其中： dow 是一周中的某一天 (Sun, Mon, Tue, Wed, Thu, Fri, Sat)。

【演示：获取当前日期时间】

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

如下所示：

```
public static void main(String args[]) {
    // 初始化 Date 对象
    Date date = new Date();

    // 使用 toString() 函数显示日期时间
    System.out.println(date.toString());
    //Sat Apr 27 15:09:43 CST 2019
}
```

【演示：日期比较】

- 使用 getTime() 方法获取两个日期（自1970年1月1日经历的毫秒数值），然后比较这两个值。

```
public static void main(String[] args) {
    // 初始化 Date 对象
    Date date = new Date();

    long time = date.getTime();
    long time2 = date.getTime();
    System.out.println(time==time2);
}
```

- 使用方法 before()，after() 和 equals()。例如，一个月的12号比18号早，则 new Date(99, 2, 12).before(new Date (99, 2, 18)) 返回true。

```
public static void main(String[] args) {
    boolean before = new Date(97, 01, 05).before(new Date(99, 11, 16));
    System.out.println(before);
}
```

2、SimpleDateFormat

【演示：使用 SimpleDateFormat 格式化日期】

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。例如：

```
public static void main(String args[]) {
    Date dNow = new Date( );
    SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd hh:mm:ss");
    System.out.println("当前时间为: " + ft.format(dNow));
}
```

其中 yyyy 是完整的公元年，MM 是月份，dd 是日期，HH:mm:ss 是时、分、秒。

注意:有的格式大写，有的格式小写，例如 MM 是月份，mm 是分；HH 是 24 小时制，而 hh 是 12 小时制。

时间模式字符串用来指定时间格式。在此模式中，所有的 ASCII 字母被保留为模式字母，定义如下：

字母	描述	示例
G	纪元标记	AD
y	四位年份	2001
M	月份	July or 07
d	一个月的日期	10
h	A.M./P.M. (1~12)格式小时	12
H	一天中的小时 (0~23)	22
m	分钟数	30
s	秒数	55
S	毫秒数	234
E	星期几	Tuesday
D	一年中的日子	360
F	一个月中第几周的周几	2 (second Wed. in July)
w	一年中第几周	40
W	一个月中第几周	1
a	A.M./P.M. 标记	PM
k	一天中的小时(1~24)	24
K	A.M./P.M. (0~11)格式小时	10
z	时区	Eastern Standard Time
'	文字定界符	Delimiter
"	单引号	`

【演示：使用printf格式化日期】

printf方法使用说明

printf 方法可以很轻松地格式化时间和日期。使用两个字母格式，它以 **%t** 开头并且以下面表格中的一个字母结尾。

```
public static void main(String args[]) {
    // 初始化 Date 对象
    Date date = new Date();

    //c的使用
    System.out.printf("全部日期和时间信息：%tc%n",date);
    //f的使用
    System.out.printf("年-月-日格式：%tF%n",date);
    //d的使用
    System.out.printf("月/日/年格式：%tD%n",date);
    //r的使用
    System.out.printf("HH:MM:SS PM格式（12时制）：%tr%n",date);
```

```
        //t的使用
        System.out.printf("HH:MM:SS格式（24时制）：%tT%n",date);
        //R的使用
        System.out.printf("HH:MM格式（24时制）：%tR",date);
    }

    //结果:
    全部日期和时间信息：星期六 四月 27 15:23:45 CST 2019
    年-月-日格式：2019-04-27
    月/日/年格式：04/27/19
    HH:MM:SS PM格式（12时制）：03:23:45 下午
    HH:MM:SS格式（24时制）：15:23:45
    HH:MM格式（24时制）：15:23
```

【时间休眠：休眠(sleep)】

sleep()使当前线程进入停滞状态（阻塞当前线程），让出CPU的使用、目的是不让当前线程独自霸占该进程所获的CPU资源，以留一定时间给其他线程执行的机会。

你可以让程序休眠一毫秒的时间或者到您的计算机的寿命长的任意段时间。例如，下面的程序会休眠3秒：

```
public static void main(String args[]) {
    try {
        System.out.println(new Date( ) + "\n");
        Thread.sleep(1000*3);   // 休眠3秒
        System.out.println(new Date( ) + "\n");
    } catch (Exception e) {
        System.out.println("Got an exception!");
    }
}
```

3、Calendar类

我们现在已经能够格式化并创建一个日期对象了，但是我们如何才能设置和获取日期数据的特定部分呢，比如说小时，日，或者分钟? 我们又如何在日期的这些部分加上或者减去值呢? 答案是使用Calendar类。Date中有很多方法都已经废弃了!

Calendar类的功能要比Date类强大很多，而且在实现方式上也比Date类要复杂一些。

Calendar类是一个抽象类，在实际使用时实现特定的子类的对象，创建对象的过程对程序员来说是透明的，只需要使用getInstance方法创建即可。

创建一个代表系统当前日期的Calendar对象

```
public static void main(String args[]) {
    Calendar c = Calendar.getInstance();//默认是当前日期
    System.out.println(c);
}
//输出
java.util.GregorianCalendar[time=1556350818634,areFieldsSet=true,areAllFields
Set=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Asia/Shanghai",offs
et=28800000,dstSavings=0,useDaylight=false,transitions=29,lastRule=null],firs
tDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2019,MONTH=3,WEEK_OF_YEAR=17
,WEEK_OF_MONTH=4,DAY_OF_MONTH=27,DAY_OF_YEAR=117,DAY_OF_WEEK=7,DAY_OF_WEEK_IN
_MONTH=4,AM_PM=1,HOUR=3,HOUR_OF_DAY=15,MINUTE=40,SECOND=18,MILLISECOND=634,ZO
NE_OFFSET=28800000,DST_OFFSET=0]
```

创建一个指定日期的Calendar对象

使用Calendar类代表特定的时间，需要首先创建一个Calendar的对象，然后再设定该对象中的年月日参数来完成。

```
//创建一个代表2019年4月27日的Calendar对象
Calendar c1 = Calendar.getInstance();
c1.set(2019, 4 - 1, 27);
```

Calendar类对象字段类型

Calendar类中用以下这些常量表示不同的意义，jdk内的很多类其实都是采用的这种思想

常量	描述
Calendar.YEAR	年份
Calendar.MONTH	月份
Calendar.DATE	日期
Calendar.DAY_OF_MONTH	日期，和上面的字段意义完全相同
Calendar.HOUR	12小时制的小时
Calendar.HOUR_OF_DAY	24小时制的小时
Calendar.MINUTE	分钟
Calendar.SECOND	秒
Calendar.DAY_OF_WEEK	星期几

```
// 获得年份
int year = c1.get(Calendar.YEAR);
// 获得月份
int month = c1.get(Calendar.MONTH) + 1;
// 获得日期
int date = c1.get(Calendar.DATE);
// 获得小时
int hour = c1.get(Calendar.HOUR_OF_DAY);
// 获得分钟
int minute = c1.get(Calendar.MINUTE);
// 获得秒
int second = c1.get(Calendar.SECOND);
// 获得星期几（注意（这个与Date类是不同的）：1代表星期日、2代表星期1、3代表星期二，以此类推）
int day = c1.get(Calendar.DAY_OF_WEEK);
```

【演示：设置完整日期】

```
c1.set(2009, 6 - 1, 12);//把Calendar对象c1的年月日分别设这为：2009、6、12
```

【演示：设置某个字段】

```
c1.set(Calendar.DATE,10);
c1.set(Calendar.YEAR,2008);
//其他字段属性set的意义以此类推
```

【add设置】

```
//把c1对象的日期加上10，也就是c1也就表示为10天后的日期，其它所有的数值会被重新计算
c1.add(Calendar.DATE, 10);
//把c1对象的日期减去10，也就是c1也就表示为10天前的日期，其它所有的数值会被重新计算
c1.add(Calendar.DATE, -10);
```

【演示：GregorianCalendar】

```
public static void main(String args[]) {
    String months[] = {
        "Jan", "Feb", "Mar", "Apr",
        "May", "Jun", "Jul", "Aug",
        "Sep", "Oct", "Nov", "Dec"};

    int year;
    // 初始化 Gregorian 日历
    // 使用当前时间和日期
    // 默认为本地时间和时区
    GregorianCalendar gcalendar = new GregorianCalendar();
    // 显示当前时间和日期的信息
    System.out.print("Date: ");
    System.out.print(months[gcalendar.get(Calendar.MONTH)]);
    System.out.print(" " + gcalendar.get(Calendar.DATE) + " ");
    System.out.println(year = gcalendar.get(Calendar.YEAR));
    System.out.print("Time: ");
    System.out.print(gcalendar.get(Calendar.HOUR) + ":");
    System.out.print(gcalendar.get(Calendar.MINUTE) + ":");
    System.out.println(gcalendar.get(Calendar.SECOND));

    // 测试当前年份是否为闰年
    if(gcalendar.isLeapYear(year)) {
        System.out.println("当前年份是闰年");
    }
    else {
        System.out.println("当前年份不是闰年");
    }
}
//输出：
Date: Apr 27 2019
Time: 3:56:20
当前年份不是闰年
```

注意：Calender的月份是从0开始的，但日期和年份是从1开始的

【演示】

```
public static void main(String[] args) {
    Calendar c1 = Calendar.getInstance();
    c1.set(2017, 1, 1);
    System.out.println(c1.get(Calendar.YEAR)
                       +"-"+c1.get(Calendar.MONTH)
                       +"-"+c1.get(Calendar.DATE));
    c1.set(2017, 1, 0);
    System.out.println(c1.get(Calendar.YEAR)
                       +"-"+c1.get(Calendar.MONTH)
                       +"-"+c1.get(Calendar.DATE));
}
```

```
//输出
2017-1-1
2017-0-31
```

可见，将日期设为0以后，月份变成了上个月，但月份可以为0，把月份改为2试试：

```
public static void main(String[] args) {
    Calendar c1 = Calendar.getInstance();
    c1.set(2017, 2, 1);
    System.out.println(c1.get(Calendar.YEAR)
                       +"-"+c1.get(Calendar.MONTH)
                       +"-"+c1.get(Calendar.DATE));
    c1.set(2017, 2, 0);
    System.out.println(c1.get(Calendar.YEAR)
                       +"-"+c1.get(Calendar.MONTH)
                       +"-"+c1.get(Calendar.DATE));
}
//输出
2017-2-1
2017-1-28
```

可以看到上个月的最后一天是28号，所以Calendar.MONTH为1的时候是2月 。

【作业：在控制台输出windows日历效果】

String类

1、String概述

在API中是这样描述：

String 类代表字符串。Java 程序中的所有字符串字面值（如 "abc" ）都作为此类的实例实现。 字符串是常量；它们的值在创建之后不能更改。字符串缓冲区支持可变的字符串。因为 String 对象是不可变的，所以可以共享。

【演示：查看String源码】

```
public final class String
implements java.io.Serializable, Comparable<String>, CharSequence {

}
```

【String的成员变量】

```
//String的属性值
private final char value[];

//数组被使用的开始位置
private final int offset;

//String中元素的个数
private final int count;

//String类型的hash值
private int hash; // Default to 0

```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
private static final long serialVersionUID = -6849794470754667710L;

private static final ObjectStreamField[] serialPersistentFields =
    new ObjectStreamField[0];
```

从源码看出String底层使用一个字符数组来维护的。

成员变量可以知道String类的值是final类型的，不能被改变的，所以只要一个值改变就会生成一个新的String类型对象，存储String数据也不一定从数组的第0个元素开始的，而是从offset所指的元素开始。

【String的构造方法】

```
String()
    //初始化一个新创建的 String 对象，使其表示一个空字符序列。
String(byte[] bytes)
    //通过使用平台的默认字符集解码指定的 byte 数组，构造一个新的 String。
String(byte[] bytes, Charset charset)
    //通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。
String(byte[] bytes, int offset, int length)
    //通过使用平台的默认字符集解码指定的 byte 子数组，构造一个新的 String。
String(byte[] bytes, int offset, int length, Charset charset)
    //通过使用指定的 charset 解码指定的 byte 子数组，构造一个新的 String。
String(byte[] bytes, int offset, int length, String charsetName)
    //通过使用指定的字符集解码指定的 byte 子数组，构造一个新的 String。
String(byte[] bytes, String charsetName)
    //通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。
String(char[] value)
    //分配一个新的 String，使其表示字符数组参数中当前包含的字符序列。
String(char[] value, int offset, int count)
    //分配一个新的 String，它包含取自字符数组参数一个子数组的字符。
String(int[] codePoints, int offset, int count)
    //分配一个新的 String，它包含 Unicode 代码点数组参数一个子数组的字符。
String(String original)
    //初始化一个新创建的 String 对象，使其表示一个与参数相同的字符序列；换句话说，新创建
的字符串是该参数字符串的副本。
String(StringBuffer buffer)
    //分配一个新的字符串，它包含字符串缓冲区参数中当前包含的字符序列。
String(StringBuilder builder)
    //分配一个新的字符串，它包含字符串生成器参数中当前包含的字符序列。
```

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

```
String str="hello";//直接赋值的方式
```

通过构造方法创建字符串对象是在堆内存

```
String str=new String("hello");//实例化的方式
```

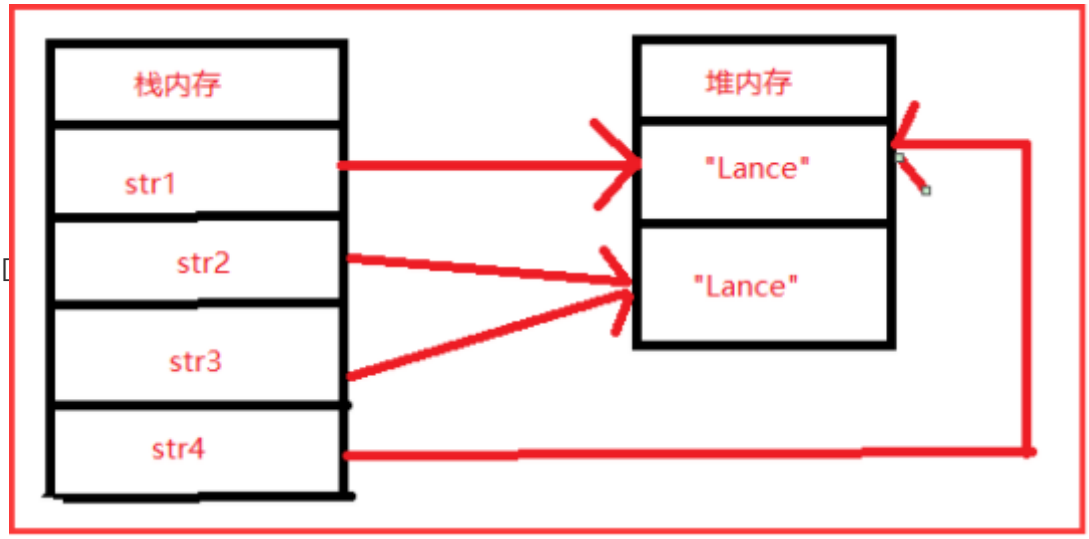
【两种实例化方式的比较】

1. 编写代码比较

```
public static void main(String[] args) {
    String str1 = "Lance";
    String str2 = new String("Lance");
    String str3 = str2; //引用传递，str3直接指向st2的堆内存地址
```

```
5       String str4 = "Lance";
6       /**
7            *  ==:
8            * 基本数据类型：比较的是基本数据类型的值是否相同
9            * 引用数据类型：比较的是引用数据类型的地址值是否相同
10           * 所以在这里的话：String类对象==比较，比较的是地址，而不是内容
11           */
12      System.out.println(str1==str2);//false
13      System.out.println(str1==str3);//false
14      System.out.println(str3==str2);//true
15      System.out.println(str1==str4);//true
16  }
```

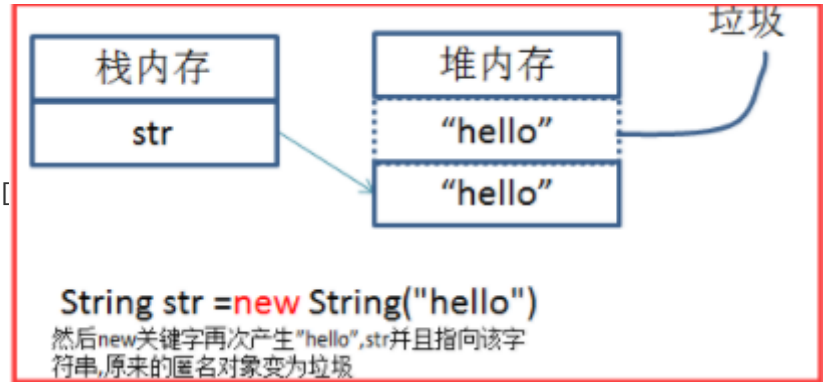
1. 内存图分析



可能这里还是不够明显，构造方法实例化方式的内存图：String str = new String("Hello");

首先：

当我们再一次的new一个String对象时：



狂神社群笔记资料，禁止外传，本人QQ：24736743

【字符串常量池】

在字符串中，如果采用直接赋值的方式（String str="Lance"）进行对象的实例化，则会将匿名对象“Lance”放入对象池，每当下一次对不同的对象进行直接赋值的时候会直接利用池中原有的匿名对象，我们可以用对象手工入池；

```
public static void main(String args[]){
    String str =new String("Lance").intern();//对匿名对象"hello"进行手工入池操作
    String str1="Lance";
    System.out.println(str==str1);//true
}
```

【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。

【避免空指向】

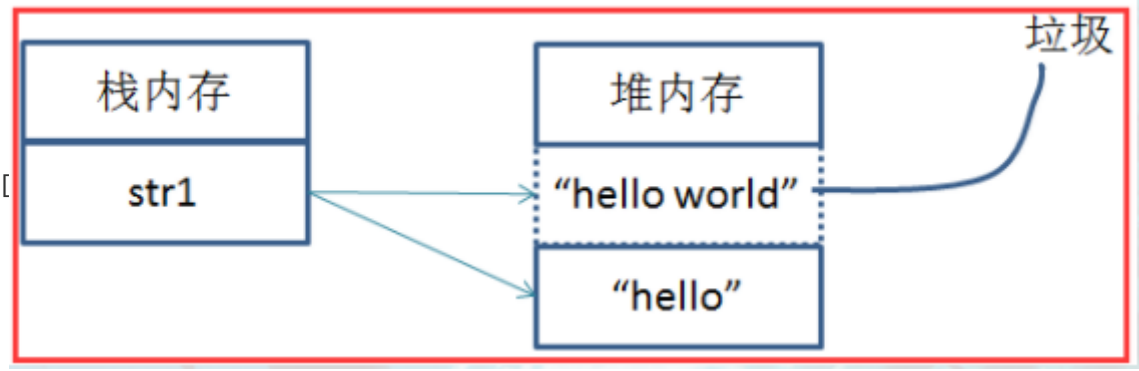
首先了解： == 和public boolean equals()比较字符串的区别

==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。

```
String str = null;
if(str.equals("hello")){//此时会出现空指向异常
    ...
}
if("hello".equals(str)){//此时equals会处理null值，可以避免空指向异常
    ...
}
```

【String类对象一旦声明则不可以改变；而改变的只是地址，原来的字符串还是存在的，并且产生垃圾】

[



3、String常用的方法

狂神社群笔记资料，禁止外传，本人QQ：24736743

[

1、String的判断

【常用方法】

```
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
```

【演示】

```
public static void main(String[] args) {
    // 创建字符串对象
    String s1 = "hello";
    String s2 = "hello";
    String s3 = "Hello";

    // boolean equals(Object obj):比较字符串的内容是否相同
    System.out.println(s1.equals(s2));  //true
    System.out.println(s1.equals(s3));  //false
    System.out.println("-----------");

    // boolean equalsIgnoreCase(String str):比较字符串的内容是否相同,忽略大小写
    System.out.println(s1.equalsIgnoreCase(s2));  //true
    System.out.println(s1.equalsIgnoreCase(s3));  //true
    System.out.println("-----------");

    // boolean startsWith(String str):判断字符串对象是否以指定的str开头
    System.out.println(s1.startsWith("he"));  //true
    System.out.println(s1.startsWith("ll"));  //false
}
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

2、String的截取

【常用方法】

```
  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
```

【演示】

```
public static void main(String args[]) {
    // 创建字符串对象
    String s = "helloworld";

    // int length():获取字符串的长度，其实也就是字符个数
    System.out.println(s.length());  //10
    System.out.println("--------");

    // char charAt(int index):获取指定索引处的字符
    System.out.println(s.charAt(0));  //h
    System.out.println(s.charAt(1));  //e
    System.out.println("--------");

    // int indexOf(String str):获取str在字符串对象中第一次出现的索引
    System.out.println(s.indexOf("l"));  //2
    System.out.println(s.indexOf("owo"));  //4
    System.out.println(s.indexOf("ak"));  //-1
    System.out.println("--------");

    // String substring(int start):从start开始截取字符串
    System.out.println(s.substring(0));  //helloworld
    System.out.println(s.substring(5));  //world
    System.out.println("--------");

    // String substring(int start,int end):从start开始，到end结束截取字符串
    System.out.println(s.substring(0, s.length()));  //helloworld
    System.out.println(s.substring(3, 8));  //lowor
}
```

3、String的转换

【常用方法】

```
   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
```

【演示】

```
public static void main(String args[]) {
    // 创建字符串对象
```

```
    String s = "abcde";

    // char[] toCharArray():把字符串转换为字符数组
    char[] chs = s.toCharArray();
    for (int x = 0; x < chs.length; x++) {
        System.out.println(chs[x]);
    }

    System.out.println("-----------");

    // String toLowerCase():把字符串转换为小写字符串
    System.out.println("HelloWorld".toLowerCase());
    // String toUpperCase():把字符串转换为大写字符串
    System.out.println("HelloWorld".toUpperCase());
}
```

4、其他方法

【常用方法】

```
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)
```

【演示】

```
public static void main(String args[]) {
    // 创建字符串对象
    String s1 = "helloworld";
    String s2 = " helloworld ";
    String s3 = " hello world ";
    System.out.println("---" + s1 + "---");
    System.out.println("---" + s1.trim() + "---");
    System.out.println("---" + s2 + "---");
    System.out.println("---" + s2.trim() + "---");
    System.out.println("---" + s3 + "---");
    System.out.println("---" + s3.trim() + "---");
    System.out.println("--------------------");

    // String[] split(String str)
    // 创建字符串对象
    String s4 = "aa,bb,cc";
    String[] strArray = s4.split(",");
    for (int x = 0; x < strArray.length; x++) {
        System.out.println(strArray[x]);
    }
}
```

4、String的不可变性

当我们去阅读源代码的时候，会发现有这样的一句话：

```
Strings are constant; their values cannot be changed after they are created.
```

意思就是说：String是个常量，从一出生就注定不可变。

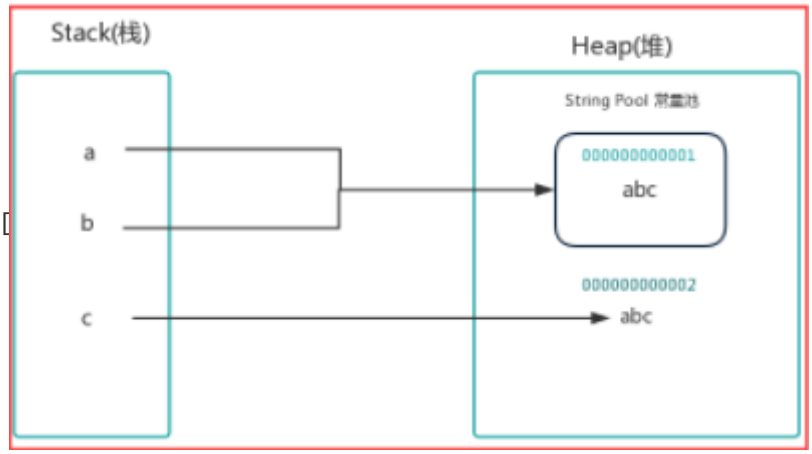
狂神社群笔记资料，禁止外传，本人QQ：24736743

我想大家应该就知道为什么String不可变了，String类被final修饰，官方注释说明创建后不能被改变，但是为什么String要使用final修饰呢?

【了解一个经典的面试题】

```
public static void main(String[] args) {
    String a = "abc";
    String b = "abc";
    String c = new String("abc");
    System.out.println(a==b);  //true
    System.out.println(a.equals(b));  //true
    System.out.println(a==c);  //false
    System.out.println(a.equals(c));  //true
}
```

内存图分析:



【分析】

因为String太过常用，JAVA类库的设计者在实现时做了个小小的变化，即采用了享元模式,每当生成一个新内容的字符串时，他们都被添加到一个共享池中，当第二次再次生成同样内容的字符串实例时，就共享此对象，而不是创建一个新对象，但是这样的做法仅仅适合于通过=符号进行的初始化。

需要说明一点的是，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，这也就是为什么a.equals(c)返回true的原因了。

【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。

5、字符串常量池

【字符串常量池概述】

1. 常量池表（Constant_Pool table）

Class文件中存储所有常量（包括字符串）的table。这是Class文件中的内容，还不是运行时的内容，不要理解它是个池子，其实就是Class文件中的字节码指令。

1. 运行时常量池（Runtime Constant Pool）

JVM内存中方法区的一部分，这是运行时的内容。这部分内容（绝大部分）是随着JVM运行时候，从常量池转化而来，每个Class对应一个运行时常量池。上一句中说绝大部分是因为：除了 Class中常量池内容，还可能包括动态生成并加入这里的内容。

1. 字符串常量池（String Pool）

这部分也在方法区中，但与Runtime Constant Pool不是一个概念，String Pool是JVM实例全局共享的，全局只有一个。JVM规范要求进入这里的String实例叫“被驻留的interned string”，各个JVM可以有不同的实现，HotSpot是设置了一个哈希表StringTable来引用堆中的字符串实例，被引用就是被驻留。

【享元模式】

其实字符串常量池这个问题涉及到一个设计模式，叫“享元模式”，顾名思义 - - - > 共享元素模式
也就是说：一个系统中如果有多处用到了相同的一个元素，那么我们应该只存储一份此元素，而让所有地方都引用这一个元素
Java中String部分就是根据享元模式设计的，而那个存储元素的地方就叫做“字符串常量池 - String Pool”

【详细分析】

```
int x  = 10;
String y = "hello";
```

1. 首先，`10`和`"hello"`会在经过javac（或者其他编译器）编译过后变为Class文件中`constant_pool table`的内容
2. 当我们的程序运行时，也就是说JVM运行时，每个Class`constant_pool table`中的内容会被加载到JVM内存中的方法区中各自Class的`Runtime Constant Pool。`
3. 一个没有被String Pool包含的Runtime Constant Pool中的字符串（这里是"hello"）会被加入到String Pool中（HosSpot使用hashtable引用方式），步骤如下：
    1. 在Java Heap（堆）中根据"hello"字面量create一个字符串对象
    2. 将字面量"hello"与字符串对象的引用在hashtable中关联起来键 - 值

    形式是："hello" = 对象的引用地址。

另外来说，当一个新的字符串出现在Runtime Constant Pool中时怎么判断需不需要在Java Heap中创建新对象呢?

策略是这样：会先去根据equals来比较Runtime Constant Pool中的这个字符串是否和String Pool中某一个是相等的（也就是找是否已经存在），如果有那么就不创建，直接使用其引用；反之，就如同上面的第三步。

如此，就实现了享元模式，提高的内存利用效率。

举例：

```
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.
```

StringBuilder 和 StringBuffer

1、概述

狂神社群笔记资料，禁止外传，本人QQ：24736743

【演示：查看源码及API文档】

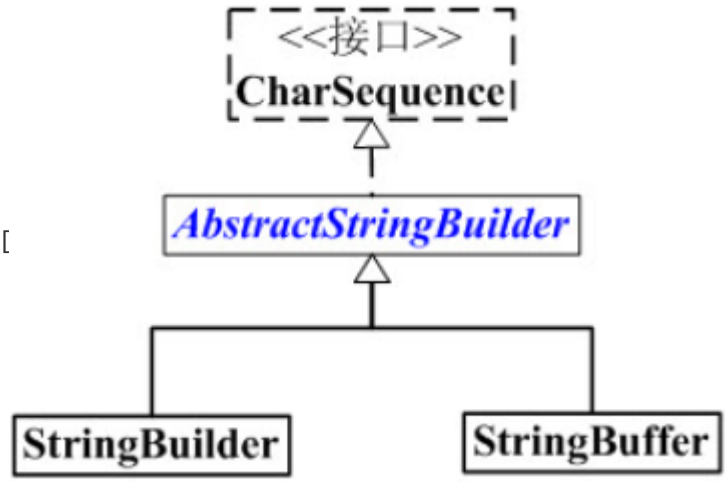
```
public final class StringBuilder
    extends AbstractStringBuilder
    implements java.io.Serializable, CharSequence{

    }
```

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。

StringBuilder 和 CharSequence之间的关系图如下：

[



【源码概览】

```
package java.lang;

public final class StringBuilder
    extends AbstractStringBuilder
    implements java.io.Serializable, CharSequence {

    static final long serialVersionUID = 4383685877147921099L;

    // 构造函数。默认的字符数组大小是16。
    public StringBuilder() {
        super(16);
    }

    // 构造函数。指定StringBuilder的字符数组大小是capacity。
    public StringBuilder(int capacity) {
        super(capacity);
    }

    // 构造函数。指定字符数组大小=str长度+15，且将str的值赋值到当前字符数组中。
    public StringBuilder(String str) {
        super(str.length() + 16);
        append(str);
    }

    // 构造函数。指定字符数组大小=seq长度+15，且将seq的值赋值到当前字符数组中。
    public StringBuilder(CharSequence seq) {
        this(seq.length() + 16);
        append(seq);
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
    }

    // 追加“对象obj对应的字符串”。String.valueOf(obj)实际上是调用obj.toString()
    public StringBuilder append(Object obj) {
        return append(String.valueOf(obj));
    }

    // 追加“str”。
    public StringBuilder append(String str) {
        super.append(str);
        return this;
    }

    // 追加“sb的内容”。
    private StringBuilder append(StringBuilder sb) {
        if (sb == null)
            return append("null");
        int len = sb.length();
        int newcount = count + len;
        if (newcount > value.length)
            expandCapacity(newcount);
        sb.getChars(0, len, value, count);
        count = newcount;
        return this;
    }

    // 追加“sb的内容”。
    public StringBuilder append(StringBuffer sb) {
        super.append(sb);
        return this;
    }

    // 追加“s的内容”。
    public StringBuilder append(CharSequence s) {
        if (s == null)
            s = "null";
        if (s instanceof String)
            return this.append((String)s);
        if (s instanceof StringBuffer)
            return this.append((StringBuffer)s);
        if (s instanceof StringBuilder)
            return this.append((StringBuilder)s);
        return this.append(s, 0, s.length());
    }

    // 追加“s从start(包括)到end(不包括)的内容”。
    public StringBuilder append(CharSequence s, int start, int end) {
        super.append(s, start, end);
        return this;
    }

    // 追加“str字符数组对应的字符串”
    public StringBuilder append(char[] str) {
        super.append(str);
        return this;
    }

    // 追加“str从offset开始的内容，内容长度是len”
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
    public StringBuilder append(char[] str, int offset, int len) {
        super.append(str, offset, len);
        return this;
    }

    // 追加“b对应的字符串”
    public StringBuilder append(boolean b) {
        super.append(b);
        return this;
    }

    // 追加“c”
    public StringBuilder append(char c) {
        super.append(c);
        return this;
    }

    // 追加“i”
    public StringBuilder append(int i) {
        super.append(i);
        return this;
    }

    // 追加“lng”
    public StringBuilder append(long lng) {
        super.append(lng);
        return this;
    }

    // 追加“f”
    public StringBuilder append(float f) {
        super.append(f);
        return this;
    }

    // 追加“d”
    public StringBuilder append(double d) {
        super.append(d);
        return this;
    }

    // 追加“codePoint”
    public StringBuilder appendCodePoint(int codePoint) {
        super.appendCodePoint(codePoint);
        return this;
    }

    // 删除“从start(包括)到end的内容”
    public StringBuilder delete(int start, int end) {
        super.delete(start, end);
        return this;
    }

    // 删除“位置index的内容”
    public StringBuilder deleteCharAt(int index) {
        super.deleteCharAt(index);
        return this;
    }
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
145
146     // “用str替换StringBuilder中从start(包括)到end(不包括)的内容”
147     public StringBuilder replace(int start, int end, String str) {
148         super.replace(start, end, str);
149         return this;
150     }
151
152     // “在StringBuilder的位置index处插入‘str中从offset开始的内容’，插入内容长度是
    len”
153     public StringBuilder insert(int index, char[] str, int offset,
154                                 int len)
155     {
156         super.insert(index, str, offset, len);
157         return this;
158     }
159
160     // “在StringBuilder的位置offset处插入obj对应的字符串”
161     public StringBuilder insert(int offset, Object obj) {
162         return insert(offset, String.valueOf(obj));
163     }
164
165     // “在StringBuilder的位置offset处插入str”
166     public StringBuilder insert(int offset, String str) {
167         super.insert(offset, str);
168         return this;
169     }
170
171     // “在StringBuilder的位置offset处插入str”
172     public StringBuilder insert(int offset, char[] str) {
173         super.insert(offset, str);
174         return this;
175     }
176
177     // “在StringBuilder的位置dstOffset处插入s”
178     public StringBuilder insert(int dstOffset, CharSequence s) {
179         if (s == null)
180             s = "null";
181         if (s instanceof String)
182             return this.insert(dstOffset, (String)s);
183         return this.insert(dstOffset, s, 0, s.length());
184     }
185
186     // “在StringBuilder的位置dstOffset处插入's中从start到end的内容'”
187     public StringBuilder insert(int dstOffset, CharSequence s,
188                                 int start, int end)
189     {
190         super.insert(dstOffset, s, start, end);
191         return this;
192     }
193
194     // “在StringBuilder的位置Offset处插入b”
195     public StringBuilder insert(int offset, boolean b) {
196         super.insert(offset, b);
197         return this;
198     }
199
200     // “在StringBuilder的位置Offset处插入c”
201     public StringBuilder insert(int offset, char c) {
```

```
202         super.insert(offset, c);
203         return this;
204     }
205 
206     // “在StringBuilder的位置Offset处插入i”
207     public StringBuilder insert(int offset, int i) {
208         return insert(offset, String.valueOf(i));
209     }
210 
211     // “在StringBuilder的位置Offset处插入l”
212     public StringBuilder insert(int offset, long l) {
213         return insert(offset, String.valueOf(l));
214     }
215 
216     // “在StringBuilder的位置Offset处插入f”
217     public StringBuilder insert(int offset, float f) {
218         return insert(offset, String.valueOf(f));
219     }
220 
221     // “在StringBuilder的位置Offset处插入d”
222     public StringBuilder insert(int offset, double d) {
223         return insert(offset, String.valueOf(d));
224     }
225 
226     // 返回“str”在StringBuilder的位置
227     public int indexOf(String str) {
228         return indexOf(str, 0);
229     }
230 
231     // 从fromIndex开始查找，返回“str”在StringBuilder的位置
232     public int indexOf(String str, int fromIndex) {
233         return String.indexOf(value, 0, count,
234                               str.toCharArray(), 0, str.length(),
fromIndex);
235     }
236 
237     // 从后向前查找，返回“str”在StringBuilder的位置
238     public int lastIndexOf(String str) {
239         return lastIndexOf(str, count);
240     }
241 
242     // 从fromIndex开始，从后向前查找，返回“str”在StringBuilder的位置
243     public int lastIndexOf(String str, int fromIndex) {
244         return String.lastIndexOf(value, 0, count,
245                               str.toCharArray(), 0, str.length(),
fromIndex);
246     }
247 
248     // 反转StringBuilder
249     public StringBuilder reverse() {
250         super.reverse();
251         return this;
252     }
253 
254     public String toString() {
255         // Create a copy, don't share the array
256         return new String(value, 0, count);
257     }
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
258
259     // 序列化对应的写入函数
260     private void writeObject(java.io.ObjectOutputStream s)
261         throws java.io.IOException {
262         s.defaultWriteObject();
263         s.writeInt(count);
264         s.writeObject(value);
265     }
266
267     // 序列化对应的读取函数
268     private void readObject(java.io.ObjectInputStream s)
269         throws java.io.IOException, ClassNotFoundException {
270         s.defaultReadObject();
271         count = s.readInt();
272         value = (char[]) s.readObject();
273     }
274 }
```

2、常用方法

1、insert

```
1  private static void testInsertAPIs(){
2      System.out.println("---------- testInsertAPIs -----------");
3
4      StringBuilder sbuilder = new StringBuilder();
5
6      // 在位置0处插入字符数组
7      sbuilder.insert(0, new char[]{'a', 'b', 'c', 'd', 'e'});
8      // 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
9      sbuilder.insert(0, new char[]{'A', 'B', 'C', 'D', 'E'}, 0, 3);
10     // 在位置0处插入float
11     sbuilder.insert(0, 1.414f);
12     // 在位置0处插入double
13     sbuilder.insert(0, 3.14159d);
14     // 在位置0处插入boolean
15     sbuilder.insert(0, true);
16     // 在位置0处插入char
17     sbuilder.insert(0, '\n');
18     // 在位置0处插入int
19     sbuilder.insert(0, 100);
20     // 在位置0处插入long
21     sbuilder.insert(0, 12345L);
22     // 在位置0处插入StringBuilder对象
23     sbuilder.insert(0, new StringBuilder("StringBuilder"));
24     // 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)
25     sbuilder.insert(0, new StringBuilder("STRINGBUILDER"), 6, 13);
26     // 在位置0处插入StringBuffer对象。
27     sbuilder.insert(0, new StringBuffer("StringBuffer"));
28     // 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)
29     sbuilder.insert(0, new StringBuffer("STRINGBUFFER"), 6, 12);
30     // 在位置0处插入String对象。
31     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);
}
```

2、append

```
/**
 * StringBuilder 的append()示例
 */
private static void testAppendAPIs() {

    System.out.println("------------------- testAppendAPIs ----------------------");

    StringBuilder sbuilder = new StringBuilder();

    // 追加字符数组
    sbuilder.append(new char[]{'a','b','c','d','e'});
    // 追加字符数组。0表示字符数组起始位置，3表示长度
    sbuilder.append(new char[]{'A','B','C','D','E'}, 0, 3);
    // 追加float
    sbuilder.append(1.414f);
    // 追加double
    sbuilder.append(3.14159d);
    // 追加boolean
    sbuilder.append(true);
    // 追加char
    sbuilder.append('\n');
    // 追加int
    sbuilder.append(100);
    // 追加long
    sbuilder.append(12345L);
    // 追加StringBuilder对象
    sbuilder.append(new StringBuilder("StringBuilder"));
    // 追加StringBuilder对象。6表示被追加对象的起始位置(包括)，13是结束位置(不包括)
    sbuilder.append(new StringBuilder("STRINGBUILDER"), 6, 13);
    // 追加StringBuffer对象。
    sbuilder.append(new StringBuffer("StringBuffer"));
    // 追加StringBuffer对象。6表示被追加对象的起始位置(包括)，12是结束位置(不包括)
    sbuilder.append(new StringBuffer("STRINGBUFFER"), 6, 12);
    // 追加String对象。
    sbuilder.append("String");
    // 追加String对象。1表示被追加对象的起始位置(包括)，6是结束位置(不包括)
    sbuilder.append("0123456789", 1, 6);
```

```
    sbuilder.append('\n');

    // 追加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.append(map);
    sbuilder.append('\n');

    // 追加unicode编码
    sbuilder.appendCodePoint(0x5b57);    // 0x5b57是“字”的unicode编码
    sbuilder.appendCodePoint(0x7b26);    // 0x7b26是“符”的unicode编码
    sbuilder.appendCodePoint(0x7f16);    // 0x7f16是“编”的unicode编码
    sbuilder.appendCodePoint(0x7801);    // 0x7801是“码”的unicode编码

    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();
    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

```
private static void testDeleteAPIs() {

    System.out.println("------------------- testDeleteAPIs -----------------
--");

    StringBuilder sbuilder = new StringBuilder("0123456789");
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
    // 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
 */
private static void testIndexAPIs() {
    System.out.println("-------------------------------- testIndexAPIs --------------------------------");

    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

```
/**
 * StringBuilder 的其它API示例
 */
```


狂神社群笔记资料，禁止外传，本人QQ：24736743

```
4  private static void testOtherAPIs() {
5
6      System.out.println("----------- testOtherAPIs -----------");
7      StringBuilder sbuilder = new StringBuilder("0123456789");
8
9      int cap = sbuilder.capacity();
10     System.out.printf("cap=%d\n", cap);
11
12     /*
13         capacity()返回的是字符串缓冲区的容量
14         StringBuffer( ); 分配16个字符的缓冲区
15         StringBuffer( int len ); 分配len个字符的缓冲区
16         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
17         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。
18     */
19
20     char c = sbuilder.charAt(6);
21     System.out.printf("c=%c\n", c);
22
23     char[] carr = new char[4];
24     sbuilder.getChars(3, 7, carr, 0);
25     for (int i=0; i<carr.length; i++){
26         System.out.printf("carr[%d]=%c ", i, carr[i]);
27     }
28     System.out.println();
29 }
```

3、StringBuffer

和StringBulider用法差不多，不过多介绍，主要看一下三者的区别

4、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

解释：

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。

为什么是大多数情况呢?

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”;

大部分情况下StringBuilder的速度要大于StringBuffer：

java.lang.StringBuilder一个可变的字符序列是5.0新增的。（大多数情况下就是我们是在单线程下进行的操作，所以大多数情况下是建议用StringBuilder而不用StringBuffer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。

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

5、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。

File类

1、File类的基本用法

1. java.io.File类：文件和目录路径名的抽象表示形式。

File类的常见构造方法：

```
public File(String pathname)
```

以pathname为路径创建File对象，如果pathname是相对路径，则默认的当前路径在系统属性user.dir中存储。

1. File的静态属性String separator存储了当前系统的路径分隔符。
2. 通过File对象可以访问文件的属性。

```
public boolean canRead()
public boolean exists()
public boolean isFile()
public long lastModified()
public String getName()
public boolean canWrite()
public boolean isDirectory()
public boolean isHidden()
public long length()
public String getPath()
```

1. 通过File对象创建空文件或目录（在该对象所指的文件或目录不存在的情况下）。

```
public boolean createNewFile()throws IOException
public boolean delete()
public boolean mkdir(),  mkdirs()
```

1. 常见构造器，方法

【演示】

```
import java.io.File;
import java.io.IOException;

public class TestFile {
    /**
     * File文件类 1.代表文件 2.代表目录
     */
    public static void main(String[] args) {
        File f = new File("d:/src3/TestObject.java");
        File f2 = new File("d:/src3");
        File f3 = new File(f2, "TestFile.java");
        File f4 = new File(f2, "TestFile666.java");
        File f5 = new File("d:/src3/aa/bb/cc/dd");
        //f5.mkdirs();
        f5.delete();

        try {
            f4.createNewFile();
            System.out.println("文件创建成功！");
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (f.isFile()) {
            System.out.println("是一个文件！");
        }
        if (f2.isDirectory()) {
            System.out.println("是一个目录！");
        }
        if (f3.isFile()) {
            System.out.println("是一个文件奥");
        }
```

```
    }
}
```


狂神社群笔记资料，禁止外传，本人QQ：24736743