8-JavaSE之三个特殊的类

> *本节目标*
>
> 1. String类
> 2. Object类
> 3. 包装类

1.String类

String类在所有项目开发之中都会使用到

1.1 String类的两种实例化方式

- 直接赋值：

```
String str = "Hello Bit" ; // str是一个对象，那么"Hello Bit" 就应该保存在堆内存中
System.out.println(str) ;
```

这种赋值方式最为常用。但是String本身毕竟是一个类，既然是类，那么类中一定存在构造方法。String类也不例外，String类的其中一种构造方法如下：

public String(String str) ;

- 传统方法：

```
String str = new String("Hello Bit") ;
System.out.println(str) ;
```

该方法符合传统做法，使用关键字new进行对象实例化。

暂时不需要考虑这两者的区别以及使用，只需要清楚String类现在提供有两种对象实例化的模式。

1.2 字符串相等比较

如果现在有两个int型变量，判断其相等可以使用“==”完成。

范例：观察基本数据类型比较

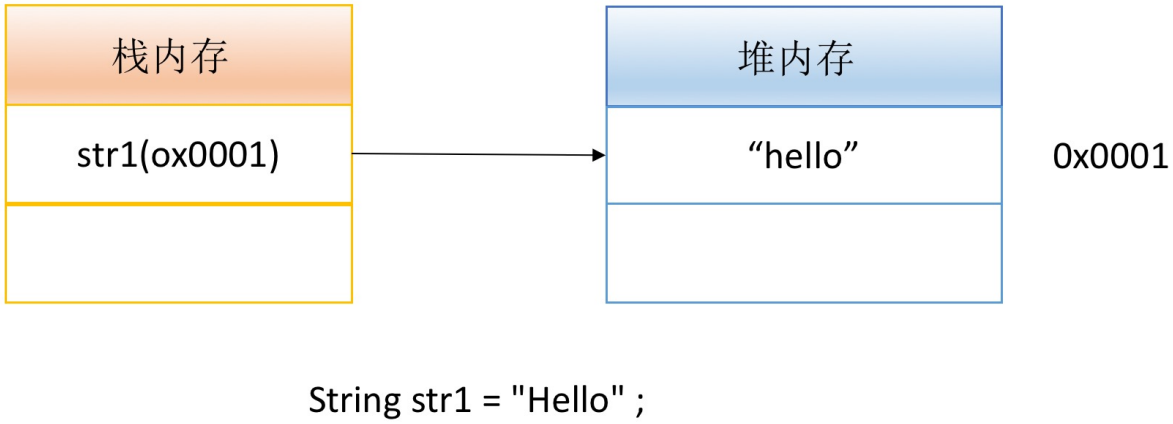
```
int x = 10 ;
int y = 10 ;
System.out.println(x==y); //true
```

如果说现在在String类对象上使用“==”？

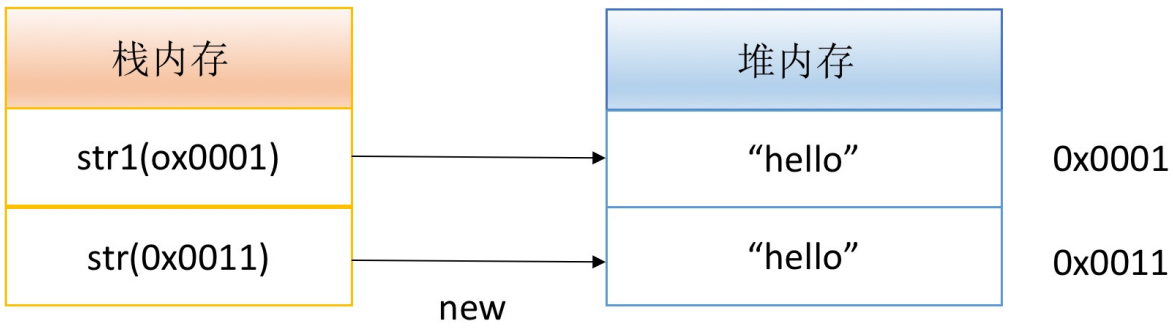
范例：观察字符串“==”比较

```
String str1 = "Hello" ;
String str = new String("Hello") ;
System.out.println(str1==str); // false
```

现在两个字符串内容相同，而使用“==“得到的结果不同。来看如下内存图分析：



String str1 = "Hello" ;



String str = new String("Hello") ;

"=="本身是进行数值比较的，如果现在用于对象比较，那么所比较的就应该是两个对象所保存的内存地址数值比较，而并没有比较对象的内容。

那么要想比较内容比较，则必须采用String类提供的equals方法。

范例：使用equals方法比较字符串内容

```
String str1 = "Hello" ;
String str = new String("Hello") ;
System.out.println(str1.equals(str));
```

面试题：请解释String类"=="与"equals"的区别

1. ”==”：进行的数值比较，比较的是两个字符串对象的内存地址数值。
2. “equals（）”：可以进行字符串内容的比较

1.3 字符串常量是String的匿名对象

在任何的语言的底层，都不会提供有直接的字符串类型。现在所谓的字符串只是高级语言提供给用户方便开发的支持而已。在java之中，本身也没有直接提供字符串常量的概念，所有使用“”定义的内容本质上来讲都是String的匿名对象。

范例：观察字符串常量

```
String str1 = "Hello" ;
String str = new String("Hello") ;
System.out.println(str1.equals(str));
System.out.println("Hello".equals(str));
```

那么在之前出现的”String str = ”hello““，本质上就是将一个匿名的String类对象设置有名字，而且匿名对象一定保存在堆内存中。

小\tips:在日后的开发过程之中，如果要判断用户输入的字符串是否等同于特定字符串，一定要将特定字符串写在前面。

- 比较方法1：

```
String str = null ; // 假设由用户输入
System.out.println(str.equals("Hello"));
```

在进行接收用户输入数据的时候一定要考虑到用户没有输入的问题，以上面代码为例，用户没有输入的时候，一定会出现“NullPointerException”问题。

- 任何的字符串常量都是String的匿名对象，所以该对象永远不会为null。

```
String str = null ; // 假设由用户输入System.out.println("Hello".equals(str));
```

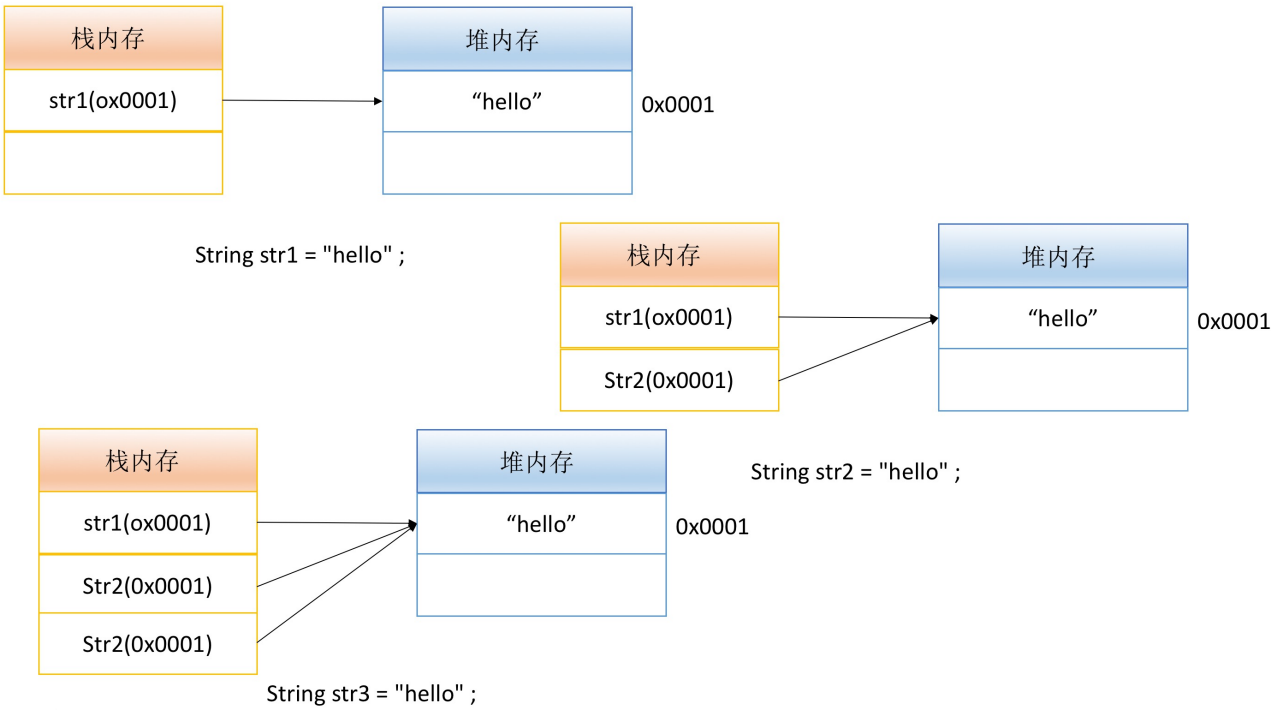
在以后进行比较的时候，强烈建议如上写法，把字符串写在前面。

1.4 String类两种实例化的区别

在第一节已经给出String类的两种实例化操作，在实际开发之中，使用哪一种更好以及彼此之间的区别有哪些呢?

1.采用直接赋值：

```
        String str1 = "hello" ;
        String str2 = "hello" ;
        String str3 = "hello" ;
        System.out.println(str1 == str2); // true
        System.out.println(str1 == str3); // true
        System.out.println(str2 == str3); // true
```



为什么现在并没有开辟新的堆内存空间呢?

String类的设计使用了共享设计模式

在JVM底层实际上会自动维护一个对象池（字符串对象池），如果现在采用了直接赋值的模式进行String类的对象实例化操作，那么该实例化对象（字符串内容）将自动保存到这个对象池之中。如果下次继续使用直接赋值的模式声明String类对象，此时对象池之中如若有指定内容，将直接进行引用；如若没有，则开辟新的字符串对象而后将其保存在对象池之中以供下次使用

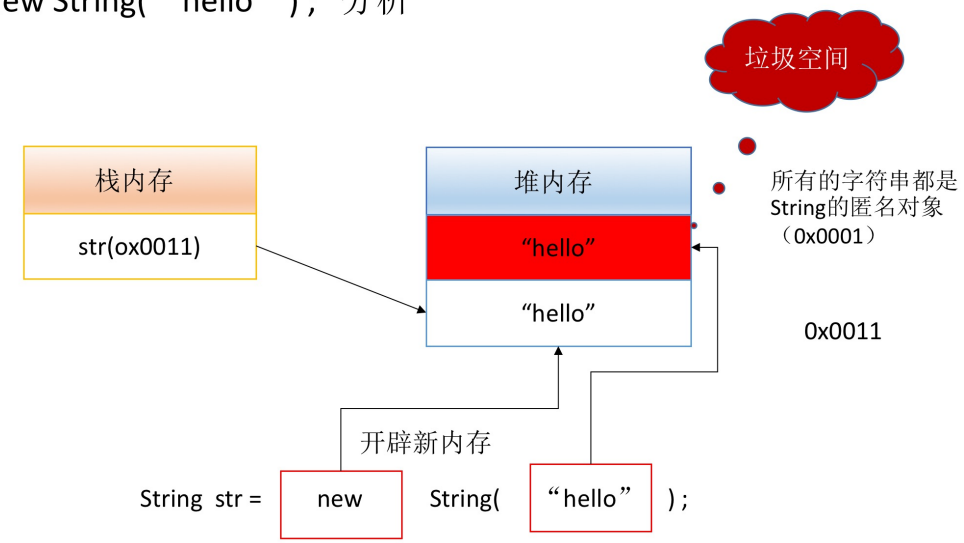
所谓的对象池就是一个对象数组（目的就是减少开销）

2.采用构造方法

类对象使用构造方法实例化是标准做法。分析如下程序:

```
String str = new String("hello") ;
```

String str = new String(“hello”)； 分析



通过分析可知，如果使用String构造方法就会开辟两块堆内存空间，并且其中一块堆内存将成为垃圾空间。除了这一缺点之外，也会对字符串共享产生问题。

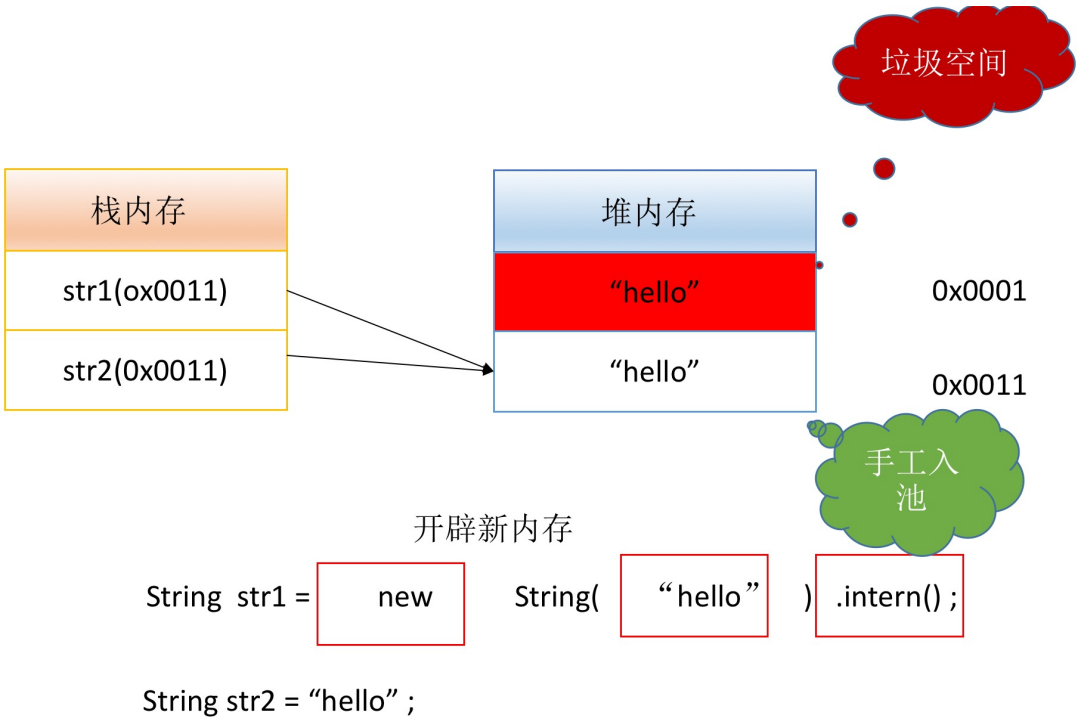
范例：观察字符串共享问题

```
// 该字符串常量并没有保存在对象池之中
String str1 = new String("hello") ;
String str2 = "hello" ;
System.out.println(str1 == str2); // false
```

在String类中提供有方法入池操作public String intern() ;

范例：观察入池操作

```
String str1 = new String("hello").intern() ;
String str2 = "hello" ;
System.out.println(str1 == str2); // true
```



面试题：请解释String类中两种对象实例化的区别

1. 直接赋值：只会开辟一块堆内存空间，并且该字符串对象可以自动保存在对象池中以供下次使用。
2. 构造方法：会开辟两块堆内存空间，其中一块成为垃圾空间，不会自动保存在对象池中，可以使用intern()方法手工入池。

因此，我们一般会采取第一种方式即直接赋值。

1.5 字符串常量不可变更

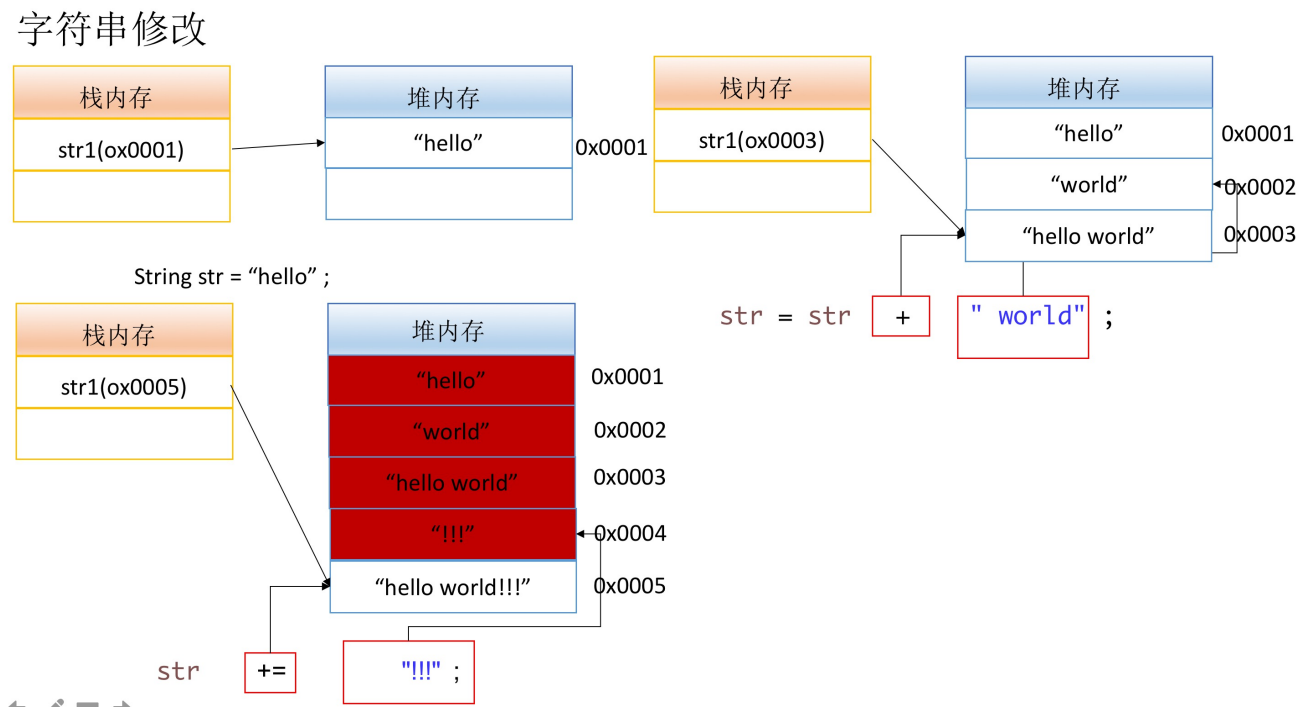
字符串一旦定义不可改变。

所有的语言对于字符串的底层实现，都是字符数组，数组的最大缺陷就是长度固定。在定义字符串常量时，它的内容不可改变。

范例：观察如下代码

```
String str = "hello" ;
str = str + " world" ;
str += "!!!" ;
System.out.println(str); // hello world!!!
```

以上字符串的变更是字符串对象的变更而非字符串常量。



可以发现字符串上没有发生任何变化，但是字符串对象的引用一直在改变，而且会形成大量的垃圾空间。正是因为String的特点，所以如下代码不应该在你的开发中出现：

```
String str = "hello" ;
for(int x = 0 ; x<1000 ; x++) {
    str += x ;
}
System.out.println(str);
```

如果有很多用户都使用了同样的操作，那么产生的垃圾数量就相当可观了。

原则:

1. 字符串使用就采用直接赋值。
2. 字符串比较就使用equals()实现。
3. 字符串别改变太多。

1.6 字符与字符串

字符串就是一个字符数组，所以在String类里面支持有字符数组转换为字符串以及字符串变为字符的操作方法。

No.	方法名称	类型	描述
1.	public String(char value[])	构造	将字符数组中的所有内容变为字符串
2.	public String(char value[], int offset, int count)	构造	将部分字符数组中的内容变为字符串
3.	public char charAt(int index)	普通	取得指定索引位置的字符，索引从 0 开始
4.	public char[] toCharArray()	普通	将字符串变为字符数组返回

范例:观察charAt()方法

```
String str = "hello" ;
System.out.println(str.charAt(0));
// 如果现在超过了字符串长度，则会产生StringIndexOutOfBoundsException异常
System.out.println(str.charAt(10));
```

字符串和字符数组的互相转换才是重点内容

范例：字符串与字符数组的转换

```
    String str = "helloworld" ;
        // 将字符串变为字符数组
        char[] data = str.toCharArray() ;
        for (int i = 0; i < data.length; i++) {
            data[i] -= 32 ;
            System.out.print(data[i]+"、 ");
        }
        // 字符数组转为字符串
        System.out.println(new String(data)); // 全部转换
        System.out.println(new String(data,5,5)); // 部分转换
```

范例：现在有一个字符串判断其是否由数字所组成

因为现在不知道字符串的长度以及包含的内容，所以最好的做法是将字符串变为字符数组而后判断每一位字符是否是"0"~"9"之间的内容，如果是则为数字。

```
public static void main(String[] args) {
        String str = "1a23456" ;
        System.out.println(isNumber(str)? "字符串由数字所组成！" : "字符串中有非数字成员！");
    }
    // 一般而言，如果方法返回boolean类型，往往以isXXX()命名
    public static boolean isNumber(String str) {
        char[] data = str.toCharArray() ;
        for (int i = 0; i < data.length; i++) {
            if (data[i]<'0'||data[i]>'9') { // 不是数字，停止遍历
                return false ;
            }
```

```
        }
        return true ;
    }
```

1.7 字节与字符串

字节常用于数据传输以及编码转换的处理之中，在String中提供有对字节的支持。

No.	方法名称	类型	描述
1.	public String(byte bytes[])	构造	将字节数组变为字符串
2.	public String(byte bytes[], int offset, int length)	构造	将部分字节数组中的内容变为字符串
3.	public byte[] getBytes()	普通	将字符串以字节数组的形式返回
4.	public byte[] getBytes(String charsetName)throws UnsupportedEncodingException	普通	编码转换处理

范例：实现字符串与字节数组的转换处理

```
String str = "helloworld" ;
byte[] data = str.getBytes() ;
for (int i = 0; i < data.length; i++) {
    data[i] -= 32 ;
    System.out.print(data[i]+"、 ");
}
System.out.println(new String(data));
```

通过程序可以发现，字节并不适合处理中文，只有字符适合处理中文。按照程序的概念来讲，一个字符等于两个字节。字节只适合处理二进制数据。

1.8 字符串比较

上面使用过String类提供的equals()方法，该方法本身是可以进行区分大小写的相等判断。除了这个方法之外，String类还提供有如下的比较操作：

No.	方法名称	类型	描述
1.	public boolean equals(Object anObject)	普通	区分大小写的比较
2.	public boolean equalsIgnoreCase(String anotherString)	普通	不区分大小写的比较
3.	public int compareTo(String anotherString)	普通	比较两个字符串大小关系

范例：不区分大小写比较

```
String str1 = "hello" ;
String str2 = "Hello" ;
System.out.println(str1.equals(str2)); // false
System.out.println(str1.equalsIgnoreCase(str2)); // true
```

在String类中compareTo()方法是一个非常重要的方法，该方法返回一个整型，该数据会根据大小关系返回三类内容：

1. 相等：返回0.
2. 小于：返回内容小于0.
3. 大于：返回内容大于0。

范例：观察compareTo()比较

```
System.out.println("A".compareTo("a")); // -32
System.out.println("a".compareTo("A")); // 32
System.out.println("A".compareTo("A")); // 0
System.out.println("AB".compareTo("AC")); // -1
System.out.println("刘".compareTo("杨"));
```

compareTo()是一个可以区分大小关系的方法，是String方法里是一个非常重要的方法。

1.9 字符串查找

从一个完整的字符串之中可以判断指定内容是否存在，对于查找方法有如下定义：

No.	方法名称	类型	描述
1.	public boolean contains(CharSequence s)	普通	判断一个子字符串是否存在
2.	public int indexOf(String str)	普通	从头开始查找指定字符串的位置，查到了返回位置的开始索引，如果查不到返回-1
3.	public int indexOf(String str, int fromIndex)	普通	从指定位置开始查找子字符串位置
4.	public int lastIndexOf(String str)	普通	由后向前查找子字符串位置
5.	public int lastIndexOf(String str, int fromIndex)	普通	从指定位置由后向前查找
6.	public boolean startsWith(String prefix)	普通	判断是否以指定字符串开头
7.	public boolean startsWith(String prefix, int toffset)	普通	从指定位置开始判断是否以指定字符串开头
8.	public boolean endsWith(String suffix)	普通	判断是否以指定字符串结尾

范例：字符串查找，最好用最方便的就是contains()

```
String str = "helloworld" ;
System.out.println(str.contains("world")); // true
```

该判断形式是从JDK1.5之后开始追加的，在JDK1.5以前要想实现与之类似的功能，就必须借助、indexOf()方法完成。

范例：使用indexOf()方法进行位置查找

```
String str = "helloworld" ;
System.out.println(str.indexOf("world")); // 5,w开始的索引
System.out.println(str.indexOf("bit")); // -1，没有查到
if (str.indexOf("hello") != -1) {
    System.out.println("可以查到指定字符串！");
}
```

现在基本都是用contains()方法完成。

使用indexOf()需要注意的是，如果内容重复，它只能返回查找的第一个位置

范例：使用indexOf()的注意点

```
String str = "helloworld" ;
System.out.println(str.indexOf("l")); // 2
System.out.println(str.indexOf("l",5)); // 8
System.out.println(str.lastIndexOf("l")); // 8
```

在进行查找的时候往往会判断开头或结尾。

范例：判断开头或结尾

```
String str = "**@@helloworld!!" ;
System.out.println(str.startsWith("**")); // true
System.out.println(str.startsWith("@@",2)); // ture
System.out.println(str.endsWith("!!")); // true
```

很多时候一些参数利用标记做一些处理，这时候就利用startsWith()与endsWith()来判断。

1.10 字符串替换

使用一个指定的新的字符串替换掉已有的字符串数据，可用的方法如下：

No.	方法名称	类型	描述
1.	public String replaceAll(String regex, String replacement)	普通	替换所有的指定内容
2.	public String replaceFirst(String regex, String replacement)	普通	替换首个内容

范例：字符串的替换处理

```
String str = "helloworld" ;
System.out.println(str.replaceAll("l", "_"));
System.out.println(str.replaceFirst("l", "_"));
```

字符串的替换操作与正则表达式有关，后续还会再讲。

1.11 字符串拆分

可以将一个完整的字符串按照指定的分隔符划分为若干个子字符串。

可用方法如下：

No.	方法名称	类型	描述
1.	public String[] split(String regex)	普通	将字符串全部拆分
2.	public String[] split(String regex, int limit)	普通	将字符串部分拆分，该数组长度就是 limit 极限

范例：实现字符串的拆分处理

```
String str = "hello world hello bit" ;
String[] result = str.split(" ") ; // 按照空格拆分
for(String s: result) {
    System.out.println(s);
}
```

范例：字符串的部分拆分

```
String str = "hello world hello bit" ;
String[] result = str.split(" ",2) ;
for(String s: result) {
    System.out.println(s);
}
```

以上的拆分形式都很容易。如果发现有些内容无法拆分开就需要使用"\"转义

范例：拆分IP地址

```
String str = "192.168.1.1" ;
String[] result = str.split("\\.") ;
for(String s: result) {
    System.out.println(s);
}
```

在以后实际开发之中，经常会出现这样的拆分模式：姓名：年龄|姓名：年龄|..

范例：多次拆分

```
String str = "yuisama:27|yui:25" ;
String[] result = str.split("\\|") ;
for (int i = 0; i < result.length; i++) {
    String[] temp = result[i].split(":") ;
    System.out.println(temp[0]+" = "+temp[1]);
}
```

这种代码在以后的开发之中会经常出现，该程序必须会操作。（后续在框架讲解中很常见）

1.12 字符串截取

从一个完整的字符串之中截取出部分内容。可用方法如下：

No.	方法名称	类型	描述
1.	public String substring(int beginIndex)	普通	从指定索引截取到结尾
2.	public String substring(int beginIndex, int endIndex)	普通	截取部分内容

范例:观察字符串截取

```
String str = "helloworld" ;
System.out.println(str.substring(5));
System.out.println(str.substring(0, 5));
```

索引从0开始。

1.13 字符串其他操作方法

No.	方法名称	类型	描述
1.	public String trim()	普通	去掉字符串中的左右空格，保留中间空格
2.	public String toUpperCase()	普通	字符串转大写
3.	public String toLowerCase()	普通	字符串转小写
4.	public native String intern()	普通	字符串入池操作
5.	public String concat(String str)	普通	字符串连接，等同于“+”，不入池
6.	public int length()	普通	取得字符串长度
7.	public boolean isEmpty()	普通	判断是否为空字符串，但不是 null，而是长度为 0

范例：观察trim()方法的使用

```
String str = "   hello  world   " ;
System.out.println("["+str+"]");
System.out.println("["+str.trim()+"]");
```

范例：大小写转换

```
        String str = "   hello%$$%@#$%world 哈哈哈  " ;
        System.out.println(str.toUpperCase());
        System.out.println(str.toLowerCase());
```

这两个函数只转换字母。

范例：字符串length()

```
        String str = "   hello%$$%@#$%world 哈哈哈  " ;
        System.out.println(str.length());
```

注意：数组长度使用数组名称.length属性，而String中使用的是length()方法

范例：观察isEmpty()方法

```
        System.out.println("hello".isEmpty());
        System.out.println("".isEmpty());
        System.out.println(new String().isEmpty());
```

String类并没有提供首字母大写操作，需要自己实现。

范例：首字母大写

```
public static void main(String[] args) {
        System.out.println(fistUpper("yuisama"));
        System.out.println(fistUpper(""));
        System.out.println(fistUpper("a"));
    }
    public static String fistUpper(String str) {
        if ("".equals(str)||str==null) {
            return str ;
        }
        if (str.length()>1) {
            return str.substring(0, 1).toUpperCase()+str.substring(1) ;
        }
        return str.toUpperCase() ;
    }
```

1.14 StringBuffer类

首先来回顾下String类的特点：

任何的字符串常量都是String对象，而且String的常量一旦声明不可改变，如果改变对象内容，改变的是其引用的指向而已。

通常来讲String的操作比较简单，但是由于String的不可更改特性，为了方便字符串的修改，提供StringBuffer类。

在String中使用"+"来进行字符串连接，但是这个操作在StringBuffer类中需要更改为append()方法：

```
public synchronized StringBuffer append(各种数据类型 b)
```

范例：观察StringBuffer使用

```
public class Test{
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        sb.append("Hello").append("World");
        fun(sb);
        System.out.println(sb);
    }
    public static void fun(StringBuffer temp) {
        temp.append("\n").append("www.bit.com.cn");
    }
}
```

String和StringBuffer最大的区别在于：String的内容无法修改，而StringBuffer的内容可以修改。频繁修改字符串的情况考虑使用StingBuffer。

为了更好理解String和StringBuffer，我们来看这两个类的继承结构:

String类	**StringBuffer类**
public final class String implements java.io.Serializable, Comparable, CharSequence	public final class StringBuffer extends AbstractStringBuilder implements java.io.Serializable, CharSequence

可以发现两个类都是"CharSequence"接口的子类。这个接口描述的是一系列的字符集。所以字符串是字符集的子类，如果以后看见CharSequence，最简单的联想就是字符串。

注意：String和StringBuffer类不能直接转换。如果要想互相转换，可以采用如下原则:

- String变为StringBuffer:利用StringBuffer的构造方法或append()方法
- StringBuffer变为String:调用toString()方法。

除了append()方法外，StringBuffer也有一些String类没有的方法:

1. 字符串反转：

```
public synchronized StringBuffer reverse()
```

范例：观察字符串反转:

```
        StringBuffer sb = new StringBuffer("helloworld");
        System.out.println(sb.reverse());
```

2.删除指定范围的数据:

```
public synchronized StringBuffer delete(int start, int end)
```

范例：观察删除操作

```
        StringBuffer sb = new StringBuffer("helloworld");
        System.out.println(sb.delete(5, 10));
```

3.插入数据:

```
public synchronized StringBuffer insert(int offset, 各种数据类型 b)
```

范例：观察插入操作

```
        StringBuffer sb = new StringBuffer("helloworld");
        System.out.println(sb.delete(5, 10).insert(0, "你好"));
```

面试题：请解释String、StringBuffer、StringBuilder的区别:

1. String的内容不可修改，StringBuffer与StringBuilder的内容可以修改.
2. StringBuffer采用同步处理，属于线程安全操作；而StringBuilder采用异步处理，属于线程不安全操作。

2.Object类

2.1 Object类简介

Object是Java默认提供的一个类。Java里面除了Object类，所有的类都是存在继承关系的。默认会继承Object父类。

即，所有类的对象都可以使用Object进行接收。

范例：使用Object接收所有类的对象

```
class Person{}
class Student{}
public class Test {
    public static void main(String[] args) {
        fun(new Person() );
        fun(new Student() );
    }
    public static void fun(Object obj) {
        System.out.println(obj);
    }
}
```

所以在开发之中，Object类是参数的最高同一类型。但是Object类也存在有定义好的一些方法。如下:

No.	方法名称	类型	描述
1.	public Object()	构造	无参构造为子类服务
2.	public String toString()	普通	取得对象信息
3.	public boolean equals(Object obj)	普通	对象比较

对于整个Object类中的方法需要时间全部掌握。

2.2 取得对象信息

在使用对象直接输出的时候，默认输出的是一个地址编码。如果现在使用的是String类，该类对象直接输出的是内容，主要原因就是toString()方法的问题。

范例：观察String与Object类输出

```
class Person{
    private String name ;
    private int age ;
    public Person(String name, int age) {
        this.age = age ;
        this.name = name ;
    }
}
class Student{}
public class Test {
    public static void main(String[] args) {
        fun(new Person("yuisama",25) );
        fun("Hello");
    }
    public static void fun(Object obj) {
        System.out.println(obj.toString()); // 默认输出对象调用的就是toString()方法
    }
}
```

通过以上代码发现，默认Object类提供的toString()方法只能够得到一个对象的地址（而这是所有对象都共同具备的特征）。如若觉得默认给出的toString()方法功能不足，就在需要的子类上覆写toString()方法。

范例：覆写toString()方法

```
class Person{
    private String name ;
    private int age ;
    public Person(String name, int age) {
        this.age = age ;
        this.name = name ;
    }
    @Override
    public String toString() {
```

```
        return "姓名: "+this.name+",年龄: "+this.age ;
    }
}
class Student{}
public class Test {
    public static void main(String[] args) {
        fun(new Person("yuisama",25) );
        fun("Hello");
    }
    public static void fun(Object obj) {
        System.out.println(obj.toString()); // 默认输出对象调用的就是toString()方法
    }
}
```

toString()的核心目的在于取得对象信息。

String作为信息输出的重要数据类型，在Java中所有的数据类型只要遇见了String并且执行了"+",那么都要求将其变为字符串后连接，而所有对象要想变为字符串就默认使用toString()方法

范例：观察toString()方法

```
class Person{
    private String name ;
    private int age ;
    public Person(String name, int age) {
        this.age = age ;
        this.name = name ;
    }
    @Override
    public String toString() {
        return "姓名: "+this.name+",年龄: "+this.age ;
    }
}
class Student{}
public class Test {
    public static void main(String[] args) {
        String msg = "Hello " + new Person("yuisama", 25) ;
        System.out.println(msg);
    }
    public static void fun(Object obj) {
        System.out.println(obj.toString()); // 默认输出对象调用的就是toString()方法
    }
}
```

结论：字符串是爸爸！

2.3 对象比较

String类对象的比较使用的是equals()方法，实际上String类的equals()方法就是覆写的Object类中的equals()方法。

范例：实现对象比较

```
class Person{
    private String name ;
```

```
    private int age ;
    public Person(String name, int age) {
        this.age = age ;
        this.name = name ;
    }
    @Override
    public String toString() {
        return "姓名: "+this.name+",年龄: "+this.age ;
    }
    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false ;
        }
        if(this == obj) {
            return true ;
        }
        // 不是Person类对象
        if (!(obj instanceof Person)) {
            return false ;
        }
        Person person = (Person) obj ; // 向下转型，比较属性值
        return this.name.equals(person.name) && this.age==person.age ;

    }
}
class Student{}
public class Test {
    public static void main(String[] args) {
        Person per1 = new Person("yuisama", 20) ;
        Person per2 = new Person("yuisama", 20) ;
        System.out.println(per1.equals(per2));
    }
}
```

2.4 接收引用数据类型

在之前已经分析了Object可以接收任意的对象，因为Object是所有类的父类，但是Obejct并不局限于此，它可以接收所有数据类型，包括：类、数组、接口。

范例：使用Object来接受数组对象

```
public static void main(String[] args) {
        // Object接收数组对象，向上转型
        Object obj = new int[]{1,2,3,4,5,6} ;
        // 向下转型，需要强转
        int[] data = (int[]) obj ;
        for(int i :data){
            System.out.println(i+"、");
        }
    }
```

而Object可以接收接口是Java中的强制要求，因为接口本身不能继承任何类。

范例：使用Object接收接口对象

```
interface IMessage {
    public void getMessage() ;
}
class MessageImpl implements IMessage {
    @Override
    public String toString() {
        return "I am small biter" ;
    }
    public void getMessage() {
        System.out.println("比特欢迎你");
    }
}
public class Test {
    public static void main(String[] args) {
        IMessage msg  = new MessageImpl() ; // 子类向父接口转型
        Object obj = msg ; // 接口向Obejct转型
        System.out.println(obj);
        IMessage temp = (IMessage) obj ; // 强制类型转换
        temp.getMessage();
    }
}
```

Object真正达到了参数的统一，如果一个类希望接收所有的数据类型，就是用Object完成。

3.包装类

Object类可以接收所有引用数据类型。然而在Java中，数据类型分为基本数据类型和引用数据类型，那么基本数据类型如何处理呢?

3.1 包装类基本原理

包装类就是将基本数据类型封装到类中。

范例：自己定义一个包装类

```
class IntDemo {
    private int num ;
    public IntDemo(int num) {
        this.num = num ;
    }
    public int intValue() {
        return this.num ;
    }
}
```

这个时候的IntDemo实际上就是int数据类型的包装类，利用IntValue就可以实现基本数据类型变为对象的需求。

范例：IntDemo使用

```
public static void main(String[] args) {
        // 子类对象向上转型
        Object obj = new IntDemo(55) ;
        IntDemo temp = (IntDemo) obj ; // 向下转型
        System.out.println(temp.intValue()); // 取出里面的基本数据类型操作
    }
```

结论：将基本数据类型包装为一个类对象的本质就是使用Object进行接收处理。

Java中有8种数据类型，如果每种数据类型都按照以上方式编写，存在以下问题：

1. 开发中代码重复
2. 在进行数学计算的时候，必须利用明确的方法将包装的数据取出后才可以进行运算。

所以Java为了方便开发，专门提供了包装类的使用，而对于包装类的使用，提供了两种类型。

- 对象型(Object的直接子类):Boolean、Character(char)；
- 数值型(Number的直接子类):Byte、Double、Short、Long、Integer(int)、Float；

说明：关于Number类

1. Number类的定义:public abstract class Number implements java.io.Serializable.
2. 在Number类里面实际定义有六种重要方法。

3.2 装箱与拆箱

在包装类与基本数据类型处理之中存在有两个概念：

- 装箱：将基本数据类型变为包装类对象，利用每一个包装类提供的构造方法实现装箱处理。
- 拆箱：将包装类中包装的基本数据类型取出。利用Number类中提供的六种方法。

范例：以int和Integer举例

```
        Integer num = new Integer(55) ; // 装箱
        int data = num.intValue() ; // 拆箱
        System.out.println(data);
```

以上操作采用的是手工的装箱和拆箱。在JDK1.5之后，提供了自动拆装箱的机制，最为重要的是，由于此类机制的存在，可以直接利用包装类的对象进行各种数学计算。

范例：自动装箱与拆箱处理

```
        // 自动装箱
        Integer x = 55 ;
        // 可以直接利用包装类对象操作
        System.out.println(++x * 5 );
```

这个时候依然存在有"==" 和 "equals" 问题

范例：观察问题

```
Integer num1 = new Integer(10) ;
Integer num2 = new Integer(10) ;
System.out.println(num1 == num2);
System.out.println(num1 == new Integer(10));
System.out.println(num1.equals(new Integer(10)));
```

阿里编码规范:所有的相同类型的包装类对象之间值的比较，全部使用 equals 方法比较。

说明:对于 Integer var = ? 在-128 至 127 范围内的赋值，Integer 对象是在IntegerCache.cache 产生，会复用已有对象，这个区间内的 Integer 值可以直接使用==进行判断，但是这个区间之外的所有数据，都会在堆上产生，并不会复用已有对象，这是一个大坑，推荐使用 equals 方法进行判断。

阿里编码规范：使用int还是Integer?

关于基本数据类型与包装数据类型的使用标准如下： 1） 【强制】所有的 POJO 类属性必须使用包装数据类型。 2） 【强制】RPC 方法的返回值和参数必须使用包装数据类型。 3） 【推荐】所有的局部变量使用基本数据类型。

3.3 字符串与基本数据类型转换

以后要进行各种数据的输入，一定都是字符串类型的接收。所以在开发之中，如何将字符串变为各个数据类型，这个时候就需要包装类支持。

1. String变为int 类型（Integer类）： public static int parseInt(String s) throws NumberFormatException
2. String变为double类型（Double类）： public static double parseDouble(String s) throws NumberFormatException
3. String变为Boolean类型（Boolean类）： public static boolean parseBoolean(String s)

范例：将字符串变为int型

```
String str = "123" ; // String类型
int num = Integer.parseInt(str) ;
System.out.println(num);
```

范例：将字符串变为double

```
String str = "123" ; // String类型
double num = Double.parseDouble(str) ;
System.out.println(num);
```

需要注意的是，将字符串转为数字的时候，字符串的组成有非数字，那么转换就会出现错误（NumberFormatException），以后就因为这个异常，我们要写一堆程序来回避。

范例：观察非数字错误

```
String str = "123aassa" ; // String类型
double num = Double.parseDouble(str) ;
System.out.println(num);
```

而字符串与boolean转换就不受此影响。

```
        String str = "truesds" ; // String类型
        boolean num = Boolean.parseBoolean(str) ;
        System.out.println(num);
```

那么如果现在要将基本数据类型变为字符串：

1. 任何数据类型使用了"+"连接空白字符串就变为了字符串类型。
2. 使用String类中提供的valueOf()方法，此方法不产生垃圾。

```
        String str = String.valueOf(100) ;
        System.out.println(str);
```

总结：以后开发之中一定会用到字符串类型与基本数据类型的转换。