2、具体内容

在实际的开发之中没有一个项目不去使用String类，也就是说String是整个系统开发之中一个至关重要的组成类，在java程序里面所有的字符串都要求使用“”进行定义，同时也可以利用“+”实现字符串的连接处理，但是对于String类实际上还有其自身的特点，本次来进行String类的特点分析

■String类简介

字符串严格意义上来讲并不能算是一个基本数据类型，也就是说没有任何一门语言会提供有字符串这种数据类型的，而Java里面为了方便开发者进行项目的编写，所以利用其JVM的支持制造了一种可以简单实用的String类，并且可以像基本数据类型那样进行直接的赋值处理。

范例：String类对象实例化

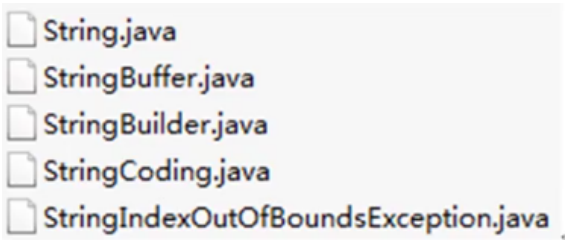
```
public class StringDemo {
    public static void main(String args[]){
        String str = "www.mldn.cn" ; // 直接赋值
        System.out.println(str) ;
    }
}
```

但是需要注意的是，String这个类里面之所以可以保存字符串主要的原因是其中定义了一个数组，也就是说在String里面所有的字符串中的每一个字符的数据都是保存在了数组之中。

提示：观察String类的源代码实现
·源代码目录：D:\Program Files\Java\jdk1.8.0_201**src.zip**;

JDK1.9之后实际上针对于字符串的定义形式是增多了，要比JDK1.8多，类的数量对比如下：

JDK1.8以前的String支持类：	JDK1.9及以后的String支持类：



从JDK1.9开始String类之中的数组类型采用了byte类型，而JDK1.9之后使用的是字符数组。

JDK1.8以前的String保存的是字符数组：	JDK1.9以后的String保存的是字节数组：
private final char value[];	private final byte value[];

原来所谓的字符串就是对数组的一种特殊包装应用，但是同时也应该清楚一点，既然包装的是数组，所以字符串里面的内容是无法改变的，这一点随后的部分会进行解释。

另外需要注意的是，在String类里面除了可以使用直接赋值的形式为对象进行实例化之外也可以按照传统的方法利用构造方法进行对象的实例化处理：public String(String str);

范例：利用构造方法进行实例化

```
public class StringDemo {
	public static void main(String args[]){
		String str = new String("www.mldn.cn") ; // 直接赋值
		System.out.println(str) ;
	}
}
```

String本身包装的是一个数组，并且其有两种对象的实例化形式：直接赋值、构造方法实例化。

■字符串比较

下面首先来回顾一下，如果说想要判断两个int型变量是否相等，那么肯定使用“==”来完成，这个是由程序直接提供的相等的运算符。

范例：进行“==”比较

```
public class StringDemo {
	public static void main(String args[]) {
		int x = 10 ;
		int y = 10 ;
		System.out.println(x == y) ;
	}
}
```

但是String类实际上也牵扯到一个相等的问题，但是对于String类相等的判断也可以使用“==”，只不过判断的不**准确**而已，下面通过代码来观察

范例：实现字符串的相等判断

```
public class StringDemo {
	public static void main(String args[]) {
		String strA = "mldn" ;
		String strB = new String("mldn") ;
		System.out.println(strA == strB) ;  // false
	}
}
```

此时的比较并没有成功，所以发现虽然两个字符串对象的内容是相同的，但是“==”也无法得到准确的相等判断，那么这种情况下如果想要实现准确的字符串相等判断，那么可以使用String类中所提供的一个比较方法：

- 字符串比较（有变形）：public boolean equals(String str);

范例：利用equals()实现字符串比较

```
public class StringDemo {
	public static void main(String args[]) {
		String strA = "mldn" ;
		String strB = new String("mldn") ;
		System.out.println(strA.equals(strB)) ;   // true
	}
}
```

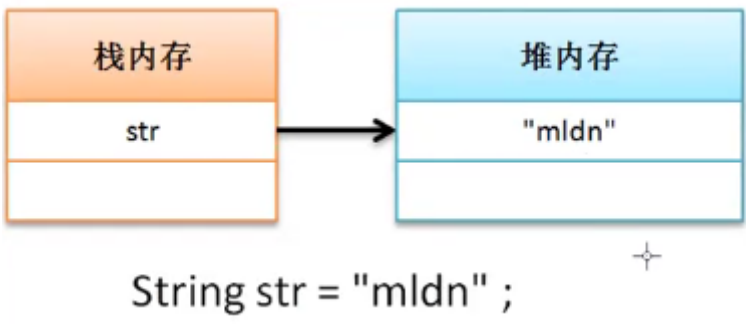
面试题：请解释String比较中”==“与equals()区别？

- “==”：进行的是数值比较，如果用于对象比较上，比较的是两个内存的地址数值；
- equals()：是类所提供的一个比较方法，可以直接进行字符串内容的判断（比较）。

■字符串常量是String类的匿名对象

现在已经清楚了String类的基本操作形式，但是需要作出一个明确的定义，在程序的开发之中任何的一个整数是int型，任意的小数默认都是double，但是对于字符串而言，首先恒旭之中不会提供有字符串这样的基本数据类型，那么可以提供的只是String类，所以任何使用“”定义的字符串常量实际上描述的都是一个String类的匿名对象。

```
String strA = "mldn" ;
```



String str = "mldn" ;

所谓的String类对象的直接赋值直接描述的是，将一个匿名对象设置一个具体的引用名字。

范例：观察匿名对象的存在

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "mldn" ;
		System.out.println("mldn".equals(str)) ; // true
	}
}
```

此时可以发现字符串常量已经可以明确的调用equals()方法实现对象相等的判断，所以可以得出的结论：程序中的确没有字符串常量这种基本类型，有的只是String类的匿名对象。

关于对象相等判断的小技巧：

在以后进行项目开发的时候，如果现在某些数据是由用户输入，并且要求这些数据位一个指定内容的情况下建议将字符串常量写在前面。

接收用户输入数据的字符串调用方法：

```
public class StringDemo {
	public static void main(String args[]) {
		String input = null ;//用户输入的内容

System.out.println(input.equals("mldn")) ;
// true
	}
}//Exception in thread "main"
java.lang.NullPointerException
```

换个方式，将字符串常量写在前面：

```
public class JavaDemo {
	public static void main(String[] args) {
		String str = null;

System.out.println("hh".equals(str));//false
	}
}
```

equals()方法里面提供有一个可以回避null的判断，所以如果将字符串常量写在前面，那么调用equals()方法的时候永远都不会出现“NullPointerException”，字符串是一个匿名对象，匿名对象一定是开辟好堆内存空间的对象。

■String类两种实例化方法区别

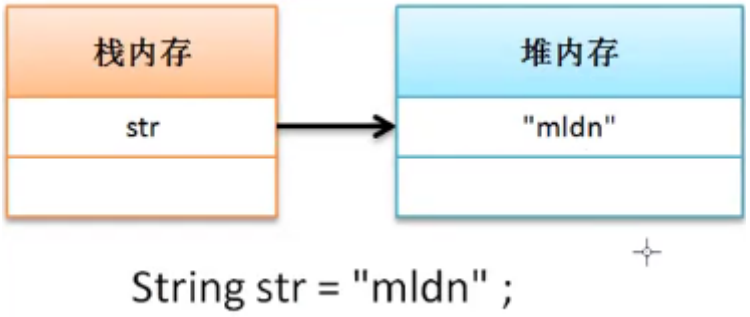
现在已经清楚了在String类对象实例化的过程之中一共提供有两种处理模式，那么现在就需要区分好这两种处理模式到底是使用哪种会更好。

1、分析直接赋值的对象实例化模式

在程序之中只需要将一个字符串赋值给String类的对象就可以实现对象的实例化处理，现在假设只有如下一行代码：

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "mldn" ;
	}
}
```

这种情况下肯定只会开辟出一个堆内存空间，此时的内存关系图如下：



除了这种内存模式之外，利用直接赋值实例化String的形式还可以实现同一个字符串对象数据的共享操作。

范例：观察String直接赋值时的数据共享

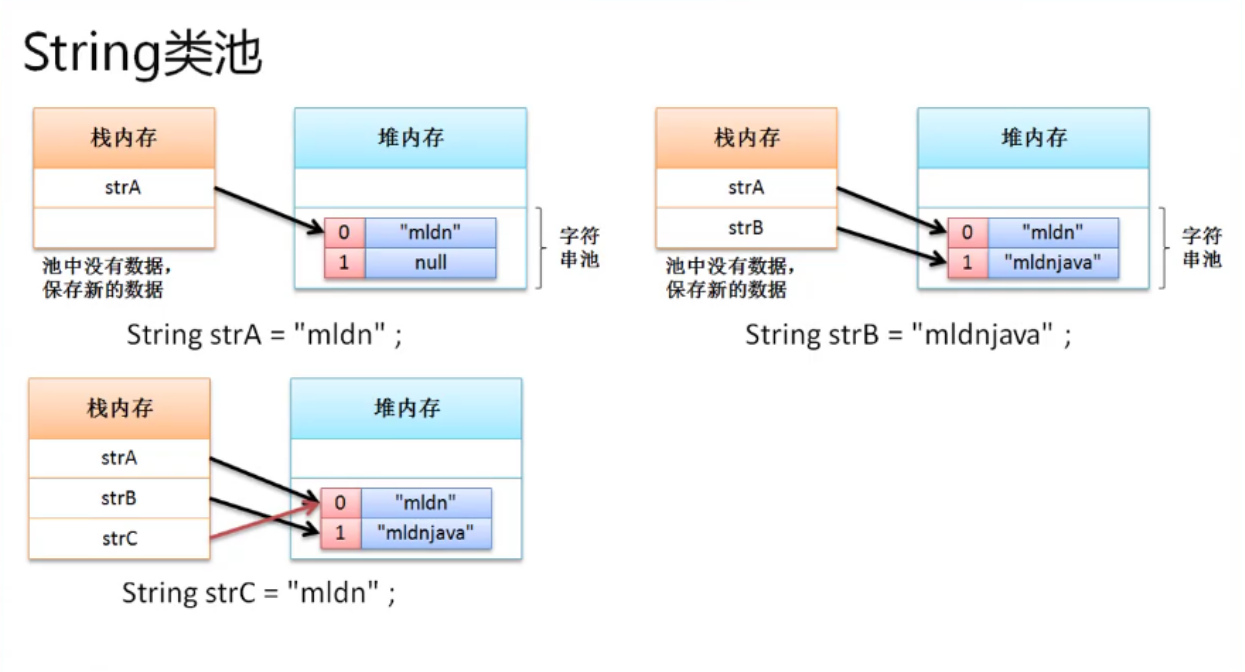
```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = "mldn" ;
        System.out.println(strA == strB) ;  // 地址判断
    }
}
```

此时程序的判断结构返回了“true”，那么可以得出结论，这两个对象所指向的堆内存是同一个，也就是说这个时候的内存关系如图所示。

之所以现在会有这样的特点，主要的原因是因为在Java程序的底层里面提供有一个专门的字符串次（字符串数组）。

范例：分析字符串池

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = "mldnjava" ;
        String strC = "mldn" ;
        System.out.println(strA == strB) ;  // 地址判断
    }
}
```



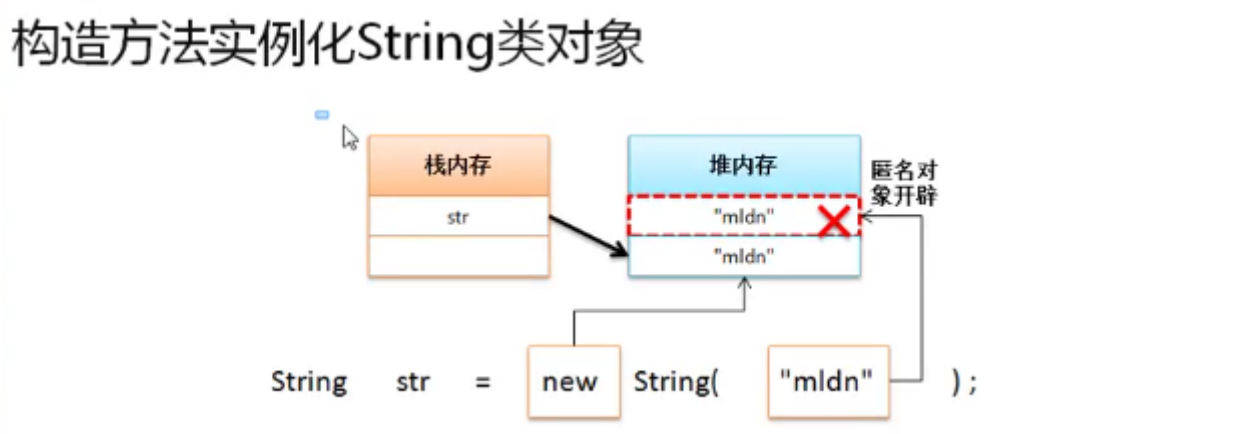
在采用直接赋值的处理过程之中，对于字符串而言可以实现池数据的自动保存，这样如果再有相同数据定义时可以减少对象的产生，以提升操作性能。

2、分析构造方法实例化

构造方法进行对象实例化可以说是在进行对象定义的时候的一种常见做法，String类为了满足于设计的结构要求也提供有构造方法实例化的做法。

```
public class StringDemo {
	public static void main(String args[]) {
		String str = new String("mldn") ;
	}
}
```

那么此时对于本程序而言，可以通过内存关系图进行观察。

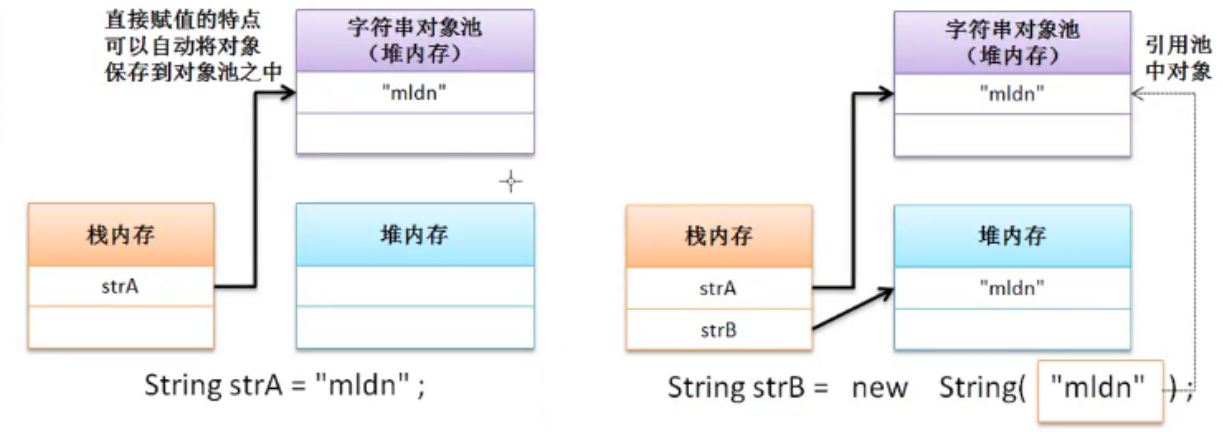


此时会开辟两块堆内存空间，而后只会使用一块，而另外一个由于字符串常量所定义的匿名对象将成为垃圾空间。

范例：更换一种形式

```
public class StringDemo {
	public static void main(String args[]) {
		String strA = "mldn" ;
```

```
        String strB = new String("mldn") ;
    }
}
```



除了以上的特点之外，在使用构造方法实例化String类对象时不会自动出现保存到字符串池的处理。

范例：观察构造方法实例化对象时的池操作

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = new String("mldn") ;
        System.out.println(strA == strB) ;  // false
    }
}
```

可以发现构造方法实例化对象实际上是属于一种自己专用的内存空间，但是在String类里面也提供有帮助开发者实现手工入池的处理情况，这个方法：public String intern();

范例：观察手工入池

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "mldn" ;
        String strB = new String("mldn")。intern() ;
        System.out.println(strA == strB) ;  // true
    }
}
```

在使用构造方法定义对象之后由于使用了intern()方法，所以即便是构造出来的String类对象的内容也可以实现对象池的统一管理，但是这种做法太啰嗦。

面试题：请解释String类两种对象实例化方法的区别？

- 直接赋值：只会产生一个实例化对象，并且可以自动保存到对象池之中，以实现该字符串实例的重用；
- 构造方法：会产生两个实例化对象，并且不会自动入池，无法实现对象重用，但是可以利用intern()方法手工入池处理。

■String对象（常量）池

对象池的主要目的是实现数据的共享处理。以String对象池为例，里面的内容主要就是为了重用，而重用的实际上就属于共享设计，但是在Java之中对象（常量）池实际上可以分为两种：

• 静态常量池：指的是程序（*.class）在加载的时候会自动将此程序之中保存的字符串、普通的常量、类和方法的信息等等，全部进行分配；

• 运行时常量池：当一个程序（*.class）加载之后，里面可能有一些变量，这个时候提供的常量池。

范例：观察一个程序（静态常量池）

```
public class StringDemo {
    public static void main(String args[]) {
        String strA = "www.mldn.cn" ;
        String strB = "www." + "mldn" + ".cn" ;
        System.out.println(strA == strB) ;  // true
    }
}
```

本程序之中给出的内容全部都是常量数据（字符串的常量都是匿名对象），所以最终在程序加载的时候会自动帮助开发者处理好相应的连接。

String常量池

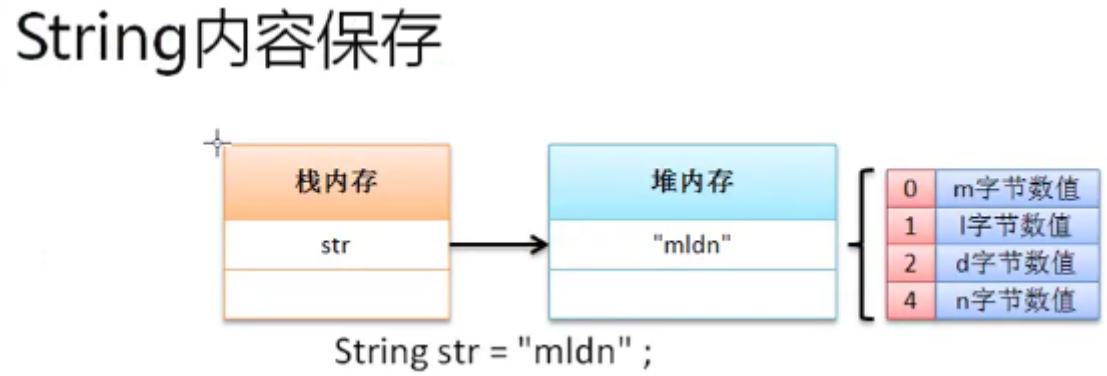
范例：观察另外一种情况

```
public class StringDemo {
    public static void main(String args[]) {
        String info = "mldn"
        String strA = "www.mldn.cn" ;
        String strB = "www." + info + ".cn" ;
        System.out.println(strA == strB) ;  // false
    }
}
```

这个时候之所以是一个false，是因为程序在加载的时候并不确定info是什么内容。因为在进行字符串连接的时候info采用的是一个变量，变量的内容是可以修改的，所以它不认为最终的strB的结果就是一个所需要的最终的结果。

■字符串内容不可修改

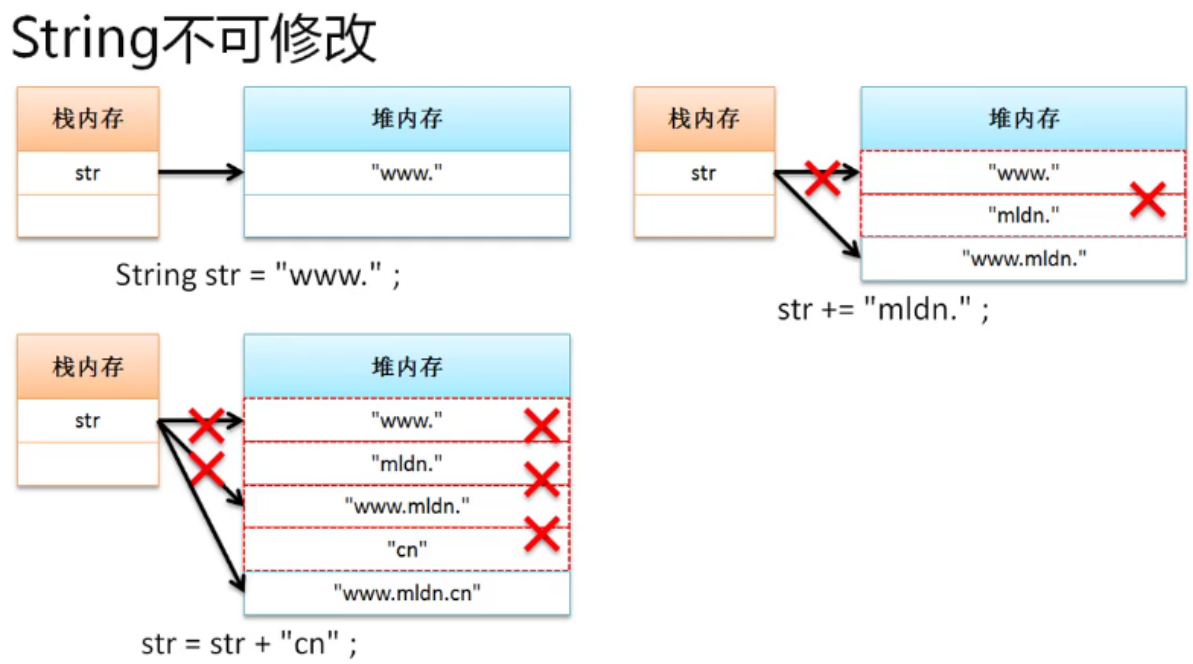
在String类之中包含的是一个数组，数组的最大缺点在于长度的不可改变，当设置了一个字符串之后，会自动的进行一个数组空间的开辟，开辟的内容长度是固定的。



范例：观察一个程序

```
public class StringDemo {
	public static void main(String args[]) {
		String str = "www." ;
		str += "mldn." ;
		str = str + "cn" ;
		System.out.println(str) ;
	}
}
```

下面一起来分析一下本程序所进行的内存处理操作。



通过此时的程序可以发现，在整个处理过程之中，字符串常量的内容并没有发生任何改变，改变的只是一个String类对象的引用，并且这种改变将有可能带来大量的垃圾空间。

范例：观察另外一种程序

```
public class StringDemo {
	public static void main(String args[]) {
```

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

如果本程序真的出现在了你的代码之中，那么将会产生1000多个垃圾空间，并且String对象的指向要修改1000次，这样程序的性能非常的差，String类在以后的开发之中不要进行内容的频繁修改。

■Java中的主方法

Java中的主方法组成是非常复杂的，而且单次很多：public static void main(String[] args)，那么下面来对该组成分析：

- public：描述的是一种访问权限，主方法是一切的开始点，开始点一定是公共的；
- static：程序的执行是通过类名称完成的，所以表示此方法是由类直接调用；
- void：主方法是一切的起点，起点一旦开始就没有返回的可能；
- main：是一个系统定义好的方法名称
- String args[]：字符串的数组，可以实现程序启动参数的接收

范例：输出启动参数

```
public class StringDemo {
    public static void main(String args[]) {
        for (String arg : args) {
            System.out.println(arg) ;
        }
    }
}
```

在程序执行的时候可以设置参数，每一个参数之间使用空格分隔；

```
D:\>javac JavaDemo.java

D:\>java JavaDemo
D:\>java JavaDemo first second
first
second
```

但是千万要记住几点，如果你的参数本身拥有空格，则必须使用“”包装。

```
D:\>java JavaDemo "hello first" "hello second"
hello first
hello second
```

以后可以暂时通过这种启动参数实现数据的输入的模拟。