认识 String 类

本节目标

- 认识 String 类
- 了解 String 类的基本用法
- 熟练掌握 String 类的常见操作
- 认识字符串常量池
- 认识 StringBuffer 和 StringBuilder

1. 创建字符串

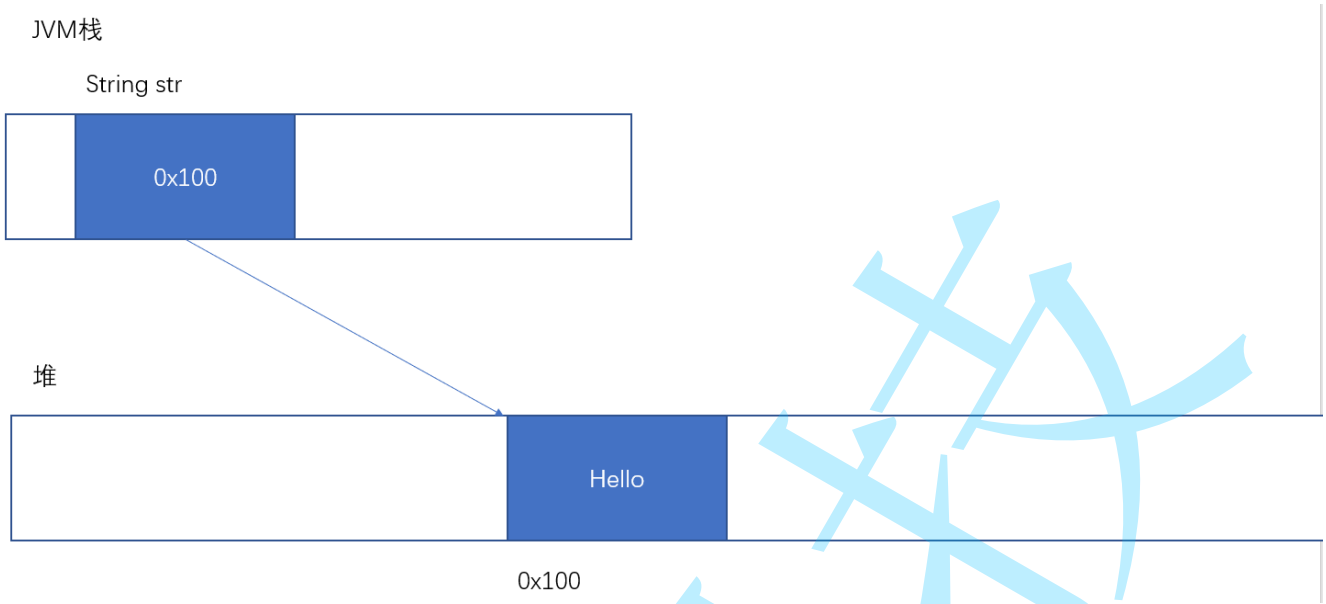
常见的构造 String 的方式

```
// 方式一
String str = "Hello Bit";

// 方式二
String str2 = new String("Hello Bit");

// 方式三
char[] array = {'a', 'b', 'c'};
String str3 = new String(array);
```

在官方文档上 (https://docs.oracle.com/javase/8/docs/api/index.html) 我们可以看到 String 还支持很多其他的构造方式, 我们用到的时候去查就可以了.

注意事项:

- "hello" 这样的字符串字面值常量, 类型也是 String.
- String 也是**引用类型**. `String str = "Hello";` 这样的代码内存布局如下

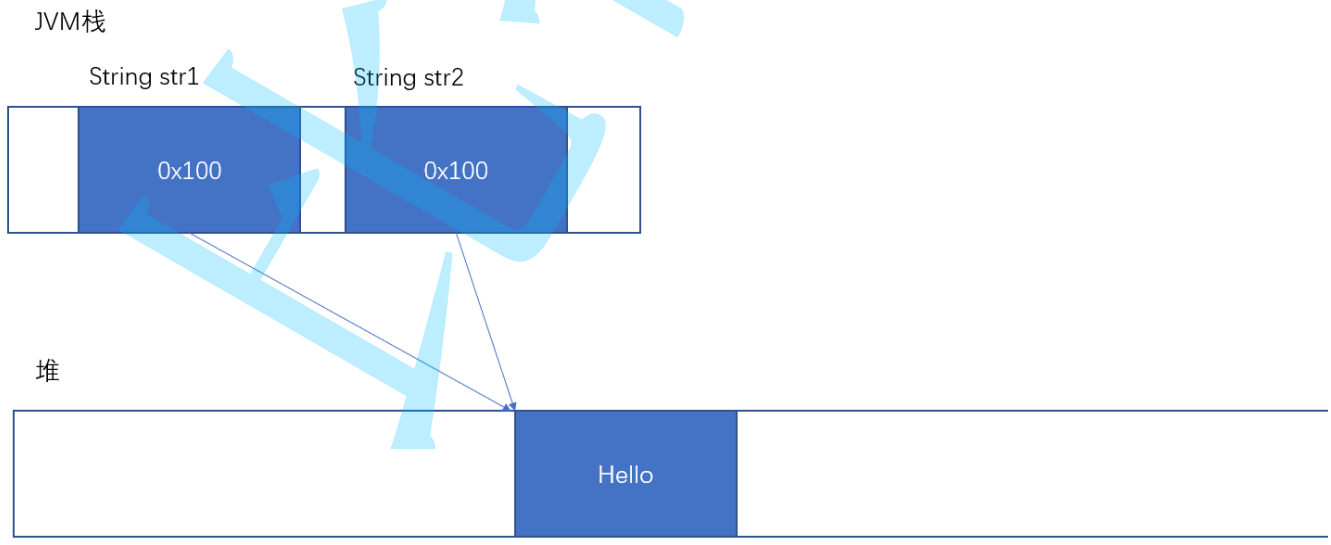


> **回忆 "引用"**
>
> 我们曾经在讲数组的时候就提到了引用的概念.
>
> 引用类似于 C 语言中的指针, 只是在栈上开辟了一小块内存空间保存一个地址. 但是引用和指针又不太相同, 指针能进行各种数字运算(指针+1)之类的, 但是引用不能, 这是一种 "没那么灵活" 的指针.
>
> 另外, 也可以把引用想象成一个标签, "贴" 到一个对象上. 一个对象可以贴一个标签, 也可以贴多个. 如果一个对象上面一个标签都没有, 那么这个对象就会被 JVM 当做垃圾对象回收掉.
>
> Java 中数组, String, 以及自定义的类都是引用类型.

由于 String 是引用类型, 因此对于以下代码

```
String str1 = "Hello";
String str2 = str1;
```

内存布局如图

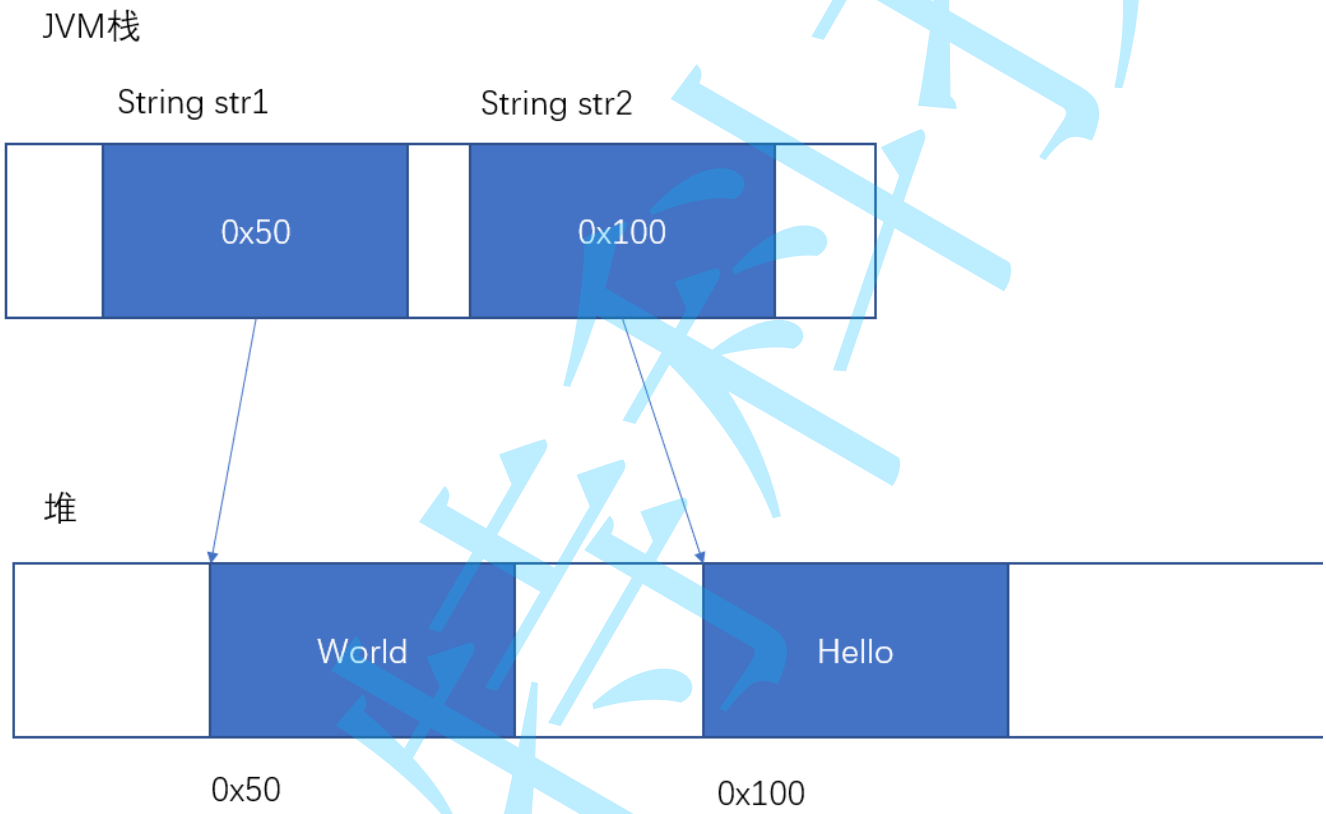


那么有同学可能会说, 是不是修改 str1 , str2 也会随之变化呢?

```
str1 = "world";
System.out.println(str2);

// 执行结果
Hello
```

我们发现, "修改" str1 之后, str2 也没发生变化, 还是 hello?

事实上, `str1 = "world"` 这样的代码并不算 "修改" 字符串, 而是让 str1 这个引用指向了一个新的 String 对象.



2. 字符串比较相等

如果现在有两个int型变量，判断其相等可以使用 == 完成。

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

// 执行结果
true
```

如果说现在在String类对象上使用 `==` ?

代码1

```
String str1 = "Hello";
String str2 = "Hello";
System.out.println(str1 == str2);

// 执行结果
true
```

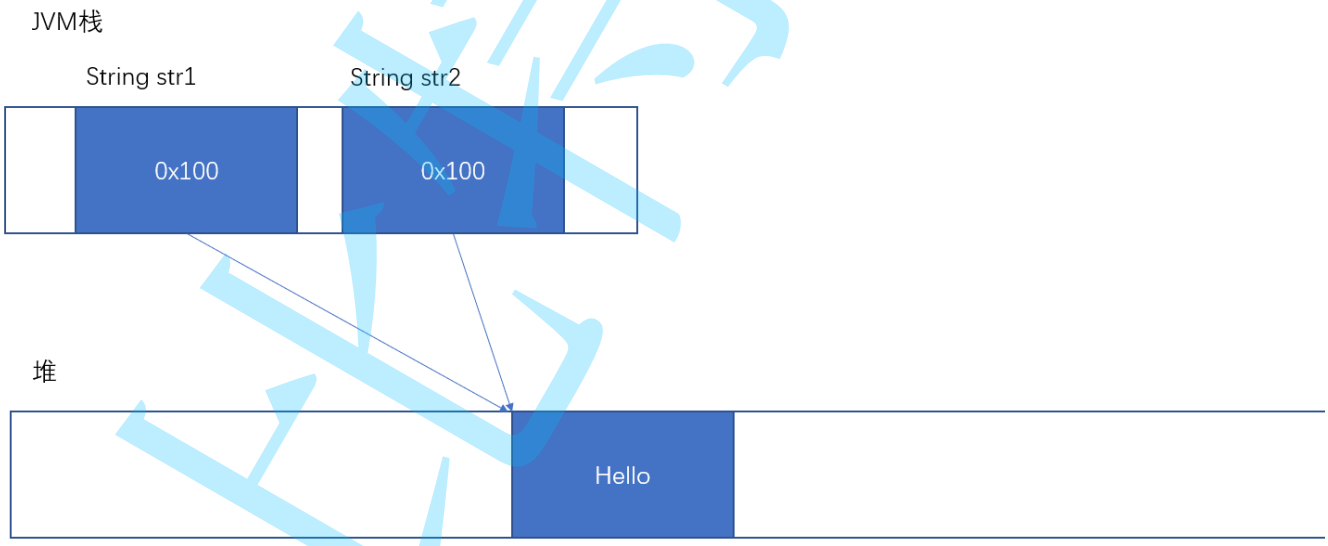
看起来貌似没啥问题, 再换个代码试试, 发现情况不太妙.

代码2

```
String str1 = new String("Hello");
String str2 = new String("Hello");
System.out.println(str1 == str2);

// 执行结果
false
```

我们来分析两种创建 String 方式的差异.

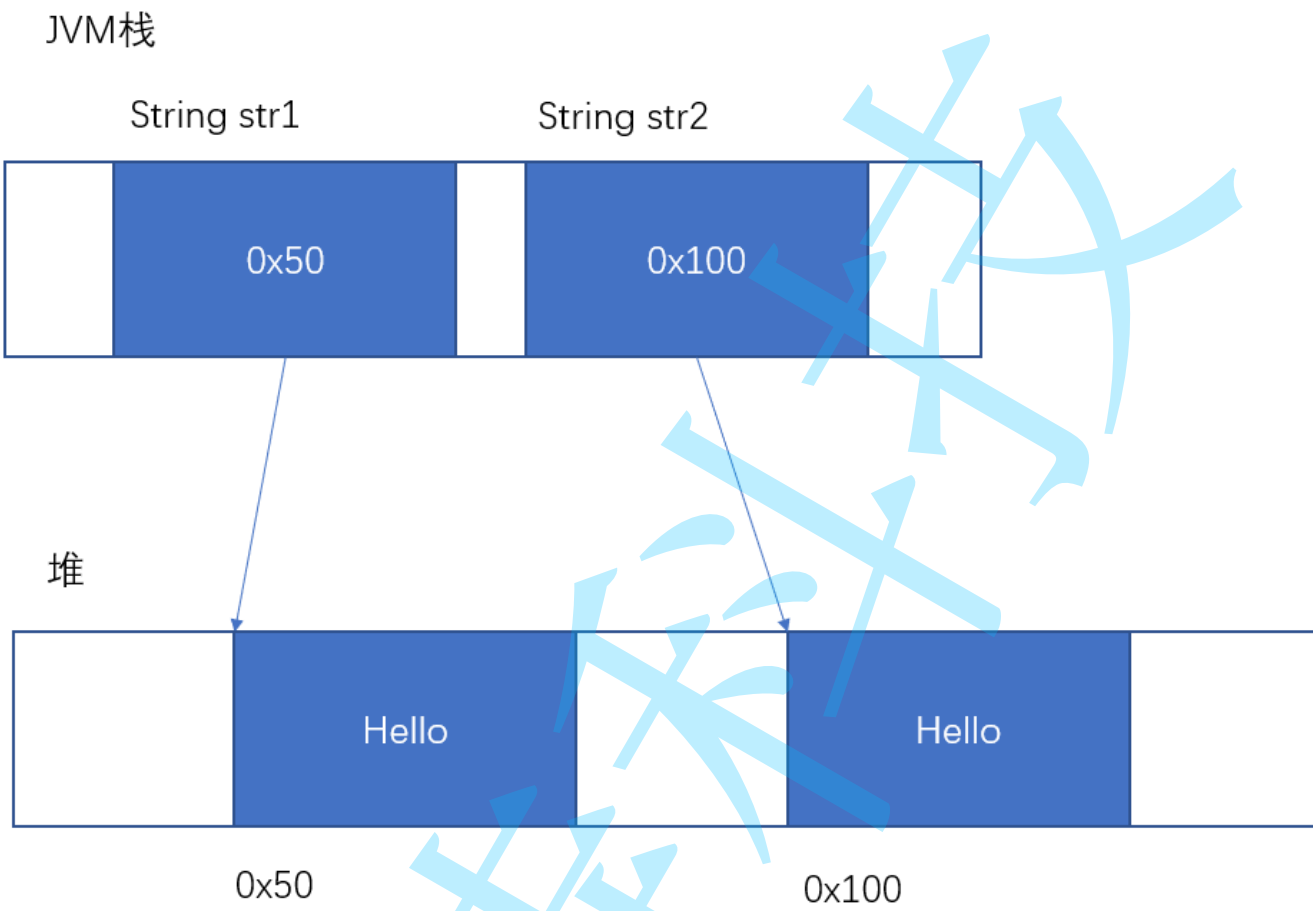
代码1内存布局



我们发现, str1 和 str2 是指向同一个对象的. 此时如 "Hello" 这样的字符串常量是在 **字符串常量池** 中.

> 关于字符串常量池
>
> 如 "Hello" 这样的字符串字面值常量, 也是需要一定的内存空间来存储的. 这样的常量具有一个特点, 就是不需要修改(常量嘛). 所以如果代码中有多个地方引用都需要使用 "Hello" 的话, 就直接引用到常量池的这个位置就行了, 而没必要把 "Hello" 在内存中存储两次.

代码2内存布局



通过 `String str1 = new String("Hello");` 这样的方式创建的 String 对象相当于再堆上另外开辟了空间来存储 "Hello" 的内容, 也就是内存中存在两份 "Hello".

String 使用 == 比较并不是在比较字符串内容, 而是比较两个引用是否是指向同一个对象.

> **关于对象的比较**
>
> 面向对象编程语言中, 涉及到对象的比较, 有三种不同的方式, 比较身份, 比较值, 比较类型.
>
> 在大部分编程语言中 == 是用来比较比较值的. 但是 Java 中的 == 是用来比较身份的.
>
> 如何理解比较值和比较身份呢?
>
> 可以想象一个场景, 现在取快递, 都有包裹储物柜. 上面有很多的格子. 每个格子里面都放着东西.

例如, "第二行, 左数第五列" 这个柜子和 "第二行, 右数第二列" 这个柜子是同一个柜子, 就是 **身份相同**. 如果身份相同, 那么里面放的东西一定也相同 (值一定也相同).

例如, "第一行, 左数第一列" 这个柜子和 "第一行, 左数第二列" 这两个柜子不是同一个柜子, 但是柜子打开后发现里面放着的是完全一模一样的两双鞋子. 这个时候就是 **值相同**.

Java 中要想比较字符串的内容, 必须采用String类提供的equals方法.

```
String str1 = new String("Hello");
String str2 = new String("Hello");
System.out.println(str1.equals(str2));
// System.out.println(str2.equals(str1)); // 或者这样写也行

// 执行结果
true
```

equals 使用注意事项

现在需要比较 str 和 "Hello" 两个字符串是否相等, 我们该如何来写呢?

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

// 方式一
System.out.println(str.equals("Hello"));
// 方式二
System.out.println("Hello".equals(str));
```

在上面的代码中, 哪种方式更好呢?

我们更推荐使用 "方式二". 一旦 str 是 null, 方式一的代码会抛出异常, 而方式二不会.

```
String str = null;

// 方式一
System.out.println(str.equals("Hello"));  // 执行结果 抛出 java.lang.NullPointerException 异常
// 方式二
System.out.println("Hello".equals(str));  // 执行结果 false
```

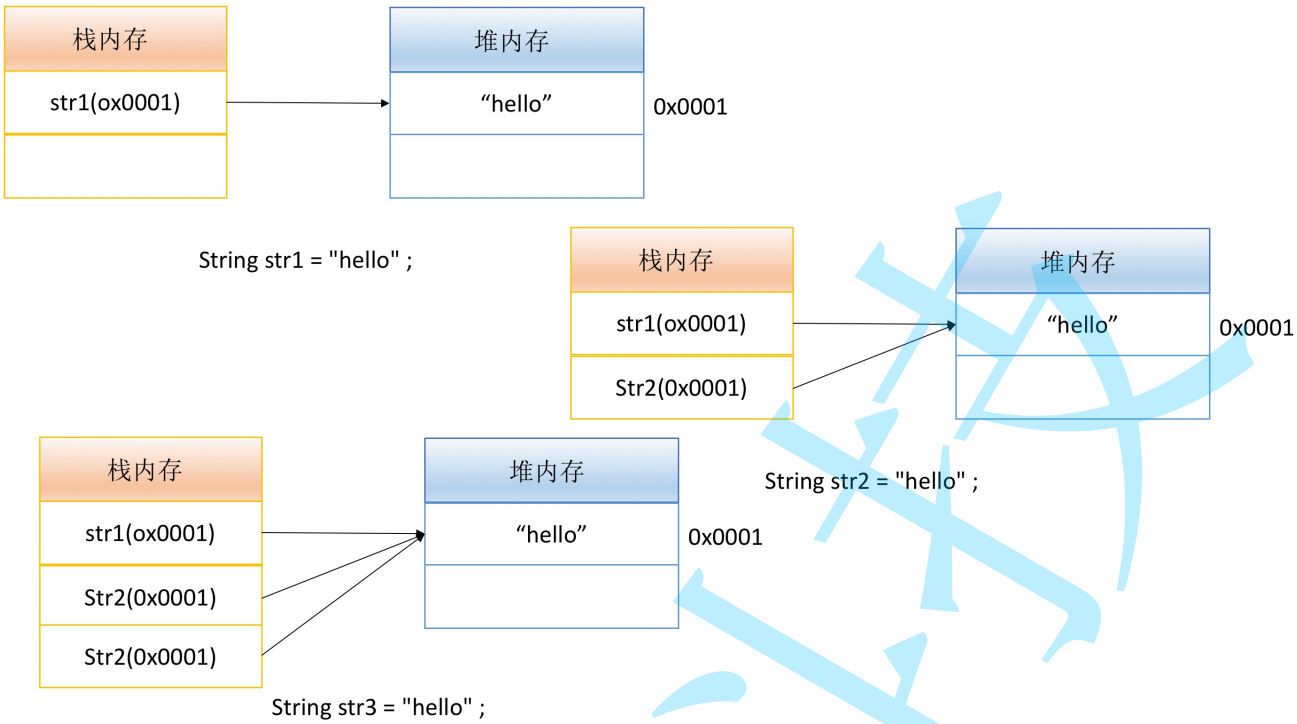
注意事项: "Hello" 这样的字面值常量, 本质上也是一个 String 对象, 完全可以使用 equals 等 String 对象的方法.

3. 字符串常量池

在上面的例子中, String类的两种实例化操作, 直接赋值和 new 一个新的 String.

a) 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（字符串常量池）

- 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

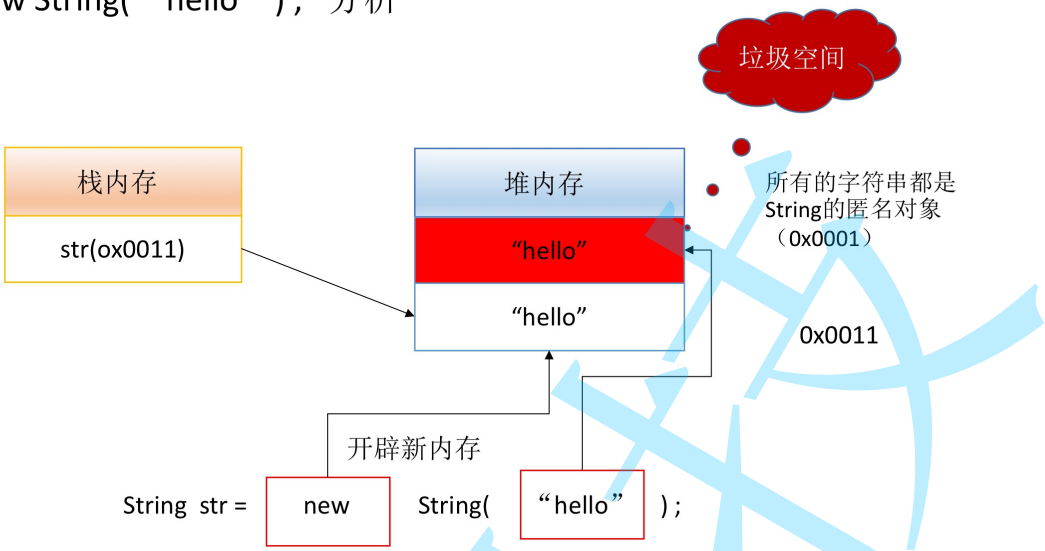
> **理解 "池" (pool)**
>
> "池" 是编程中的一种常见的, 重要的提升效率的方式, 我们会在未来的学习中遇到各种 "内存池", "线程池", "数据库连接池" ....
>
> 然而池这样的概念不是计算机独有, 也是来自于生活中. 举个栗子:
>
> 现实生活中有一种女神, 称为 "绿茶", 在和高富帅谈着对象的同时, 还可能和别的屌丝搞暧昧. 这时候这个屌丝被称为 "备胎". 那么为啥要有备胎? 因为一旦和高富帅分手了, 就可以立刻找备胎接盘, 这样 **效率比较高**.
>
> 如果这个女神, 同时在和很多个屌丝搞暧昧, 那么这些备胎就称为 **备胎池**.

b) 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

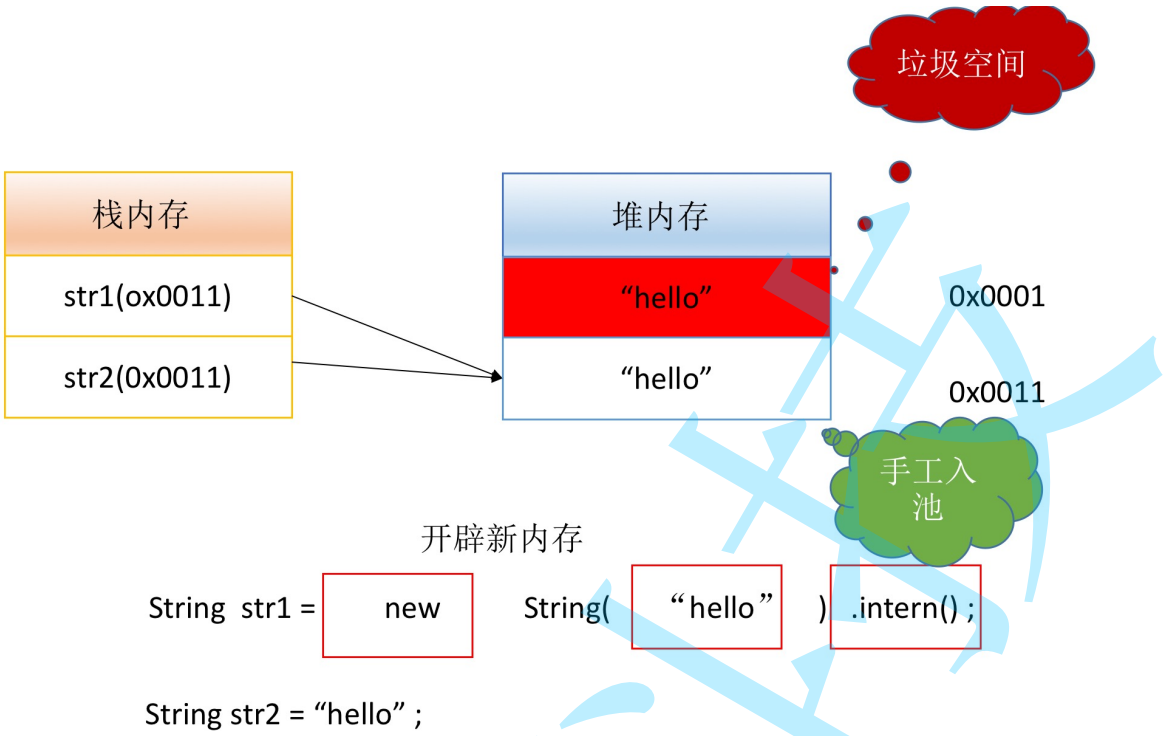


这样的做法有两个缺点:

1. 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(字符串常量 "hello" 也是一个匿名对象, 用了一次之后就不再使用了, 就成为垃圾空间, 会被 JVM 自动回收掉).
2. 字符串共享问题. 同一个字符串可能会被存储多次, 比较浪费空间.

我们可以使用 String 的 intern 方法来手动把 String 对象加入到字符串常量池中

```
// 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);

// 执行结果
false

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

// 执行结果
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，不会自动保存在对象池中，可以使用intern()方法手工入池。

综上, 我们一般采取直接赋值的方式创建 String 对象.

4. 理解字符串不可变

字符串是一种不可变对象. 它的内容不可改变.

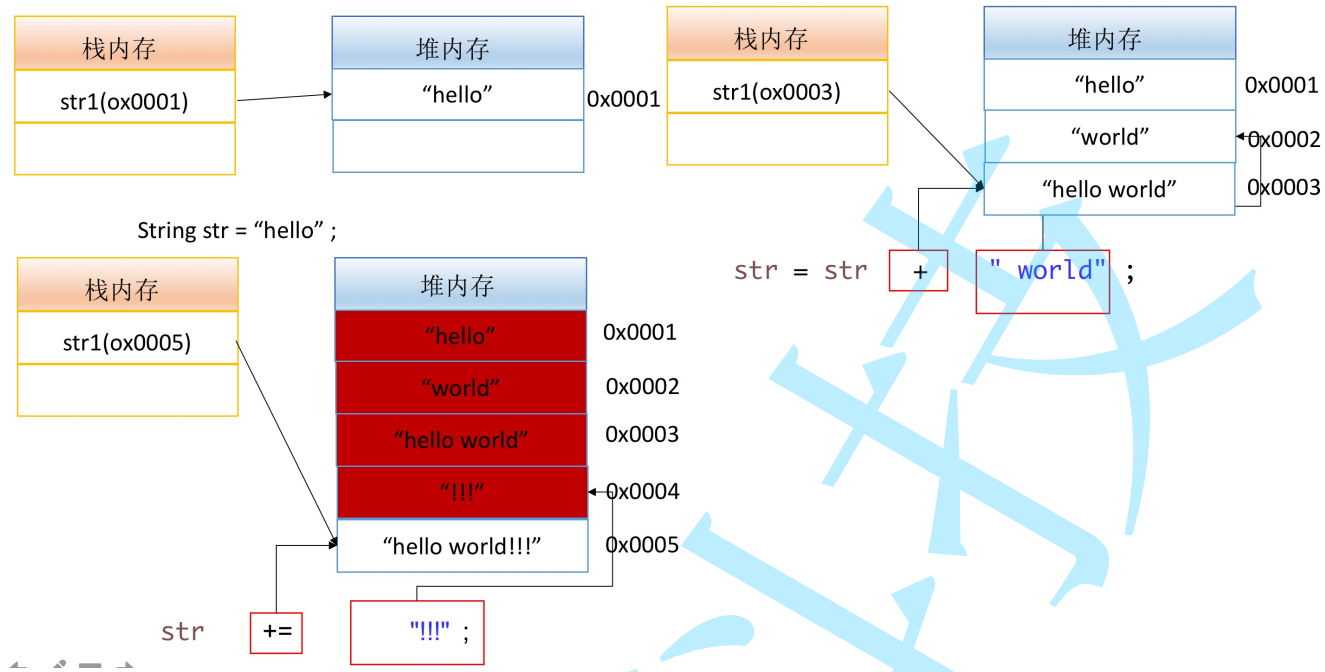
String 类的内部实现也是基于 char[] 来实现的, 但是 String 类并没有提供 set 方法之类的来修改内部的字符数组.

感受下形如这样的代码:

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

// 执行结果
hello world!!!
```

形如 += 这样的操作, 表面上好像是修改了字符串, 其实不是. 内存变化如下:

字符串修改



+= 之后 str 打印的结果却是变了, 但是不是 String 对象本身发生改变, 而是 str 引用到了其他的对象.

> **回顾引用**
>
> 引用相当于一个指针, 里面存的内容是一个地址. 我们要区分清楚当前修改到底是修改了地址对应内存的内容发生改变了, 还是引用中存的地址改变了.

那么如果实在需要修改字符串, 例如, 现有字符串 `str = "Hello"`, 想改成 `str = "hello"`, 该怎么办?

a) **常见办法**: 借助原字符串, 创建新的字符串

```
String str = "Hello";
str = "h" + str.substring(1);
System.out.println(str);

// 执行结果
hello
```

b) **特殊办法(选学)**: 使用 "反射" 这样的操作可以破坏封装, 访问一个类内部的 private 成员.

IDEA 中 ctrl + 左键 跳转到 String 类的定义, 可以看到内部包含了一个 char[] , 保存了字符串的内容.

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];
```

```
String str = "Hello";
```

```
// 获取 String 类中的 value 字段. 这个 value 和 String 源码中的 value 是匹配的.
Field valueField = String.class.getDeclaredField("value");
// 将这个字段的访问属性设为 true
valueField.setAccessible(true);
// 把 str 中的 value 属性获取到.
char[] value = (char[]) valueField.get(str);
// 修改 value 的值
value[0] = 'h';

System.out.println(str);

// 执行结果
hello
```

> **关于反射**
>
> 反射是面向对象编程的一种重要特性, 有些编程语言也称为 "自省".
>
> 指的是程序运行过程中, 获取/修改某个对象的详细信息(类型信息, 属性信息等), 相当于让一个对象更好的 "认清自己" .
>
> Java 中使用反射比较麻烦一些. 我们后面的课程中会详细介绍反射的具体用法.

> **为什么 String 要不可变?**(不可变对象的好处是什么?) (选学)
>
> 1. 方便实现字符串对象池. 如果 String 可变, 那么对象池就需要考虑何时深拷贝字符串的问题了.
> 2. 不可变对象是线程安全的.
> 3. 不可变对象更方便缓存 hash code, 作为 key 时可以更高效的保存到 HashMap 中.

注意事项: 如下代码不应该在你的开发中出, 会产生大量的临时对象, 效率比较低.

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

5. 字符, 字节与字符串

5.1 字符与字符串

字符串内部包含一个字符数组, String 可以和 char[] 相互转换.

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

代码示例: 获取指定位置的字符

```
String str = "hello" ;
System.out.println(str.charAt(0));  // 下标从 0 开始

// 执行结果
h

System.out.println(str.charAt(10));

// 执行结果
产生 StringIndexOutOfBoundsException 异常
```

代码示例: 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++) {
    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
```

代码示例: 给定字符串一个字符串, 判断其是否全部由数字所组成.

思路: 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! ");
```

```
    }

    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') {
                return false ;
            }
        }
        return true ;
    }
```

5.2 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，String 也能方便的和 byte[] 相互转换.

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

代码示例: 实现字符串与字节数组的转换处理

```
String str = "helloworld" ;
// String 转 byte[]
byte[] data = str.getBytes() ;
for (int i = 0; i < data.length; i++) {
    System.out.print(data[i]+" ");
}

// byte[] 转 String
System.out.println(new String(data));
```

5.3 小结

那么何时使用 byte[], 何时使用 char[] 呢?

- byte[] 是把 String 按照一个字节一个字节的方式处理, 这种适合在网络传输, 数据存储这样的场景下使用. 更适合针对二进制数据来操作.
- char[] 是吧 String 按照一个字符一个字符的方式处理, 更适合针对文本数据来操作, 尤其是包含中文的时候.

> 回忆概念: 文本数据 vs 二进制数据
>
> 一个简单粗暴的区分方式就是用记事本打开能不能看懂里面的内容.
>
> 如果看的懂, 就是文本数据(例如 .java 文件), 如果看不懂, 就是二进制数据(例如 .class 文件).

6. 字符串常见操作

6.1 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

代码示例: 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。

> 字符串的比较大小规则, 总结成三个字 "字典序" 相当于判定两个字符串在一本词典的前面还是后面. 先比较第一个字符的大小(根据 unicode 的值来判定), 如果不分胜负, 就依次比较后面的内容

6.2 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

代码示例: 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。

代码示例: 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

代码示例: 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。

代码示例: 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
```

6.3 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

代码示例: 字符串的替换处理

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

注意事项: 由于字符串是不可变对象, 替换不修改当前字符串, 而是产生一个新的字符串.

6.4 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

代码示例: 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);
}
```

代码示例: 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);
}
```

拆分是特别常用的操作. 一定要重点掌握. 另外有些特殊字符作为分割符可能无法正确切分, 需要加上转义.

代码示例: 拆分IP地址

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

注意事项:

1. 字符"|","*","+"都得加上转义字符，前面加上"\".
2. 而如果是""，那么就得写成"\\".
3. 如果一个字符串中有多个分隔符，可以用"|"作为连字符.

代码示例: 多次拆分

```
String str = "name=zhangsan&age=18" ;
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

6.5 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

代码示例: 观察字符串截取

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

注意事项:

1. 索引从0开始
2. 注意前闭后开区间的写法, substring(0, 5) 表示包含 0 号下标的字符, 不包含 5 号下标

6.6 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

代码示例: 观察trim()方法的使用

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

trim 会去掉字符串开头和结尾的空白字符(空格, 换行, 制表符等).

代码示例: 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。

代码示例: 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

代码示例: 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.

代码示例: 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() ;
    }
```

7. StringBuffer 和 StringBuilder

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和StringBuilder类。

StringBuffer 和 StringBuilder 大部分功能是相同的，我们课件上主要介绍 StringBuffer

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<String>, 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:

- 字符串反转：

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

代码示例: 字符串反转

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

删除指定范围的数据:

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

代码示例: 观察删除操作

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

插入数据

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

代码示例: 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:

- String的内容不可修改，StringBuffer与StringBuilder的内容可以修改.
- StringBuffer与StringBuilder大部分功能是相似的
- StringBuffer采用同步处理，属于线程安全操作；而StringBuilder未采用同步处理，属于线程不安全操作

小结

字符串操作是我们以后工作中非常常用的操作. 使用起来都非常简单方便, 一定要使用熟练.

指的注意的点:

1. 字符串的比较, ==, equals, compareTo 之间的区别.
2. 了解字符串常量池, 体会 "池" 的思想.
3. 理解字符串不可变
4. split 的应用场景
5. StringBuffer 和 StringBuilder 的功能.

作业

创建一个 MyString 类, 模拟实现字符串的基本操作

1. equals
2. compareTo
3. toCharArray
4. contains
5. indexOf
6. lastIndexOf
7. replaceAll
8. replaceFirst
9. split
10. subString
11. trim
12. isEmpty
13. length

注意, replaceAll 等方法中的参数不必考虑正则表达式的情况. 只要进行简单的查找替换就行.