

装箱拆箱

- 封装类
- Number类
- 基本类型转封装类
- 封装类转基本类型
- 自动装箱
- 自动拆箱
- int的最大值，最小值
- 练习-装箱拆箱

字符串转换

- 数字转字符串
- 字符串转数字
- 练习-字符串转换

数学方法

- 四舍五入, 随机数, 开方, 次方, π , 自然常数
- 练习-数学方法

格式化输出

- 1: 格式化输出
- 2: printf和format
- 3: 换行符
- 4: 总长度, 左对齐, 补0, 千位分隔符, 小数点位数, 本地化表达
- 5: Scanner

字符

- 保存一个字符的时候使用char
- char对应的封装类
- Character常见方法
- 常见转义
- 练习-character

字符串

- 创建字符串
- final
- immutable
- 字符串格式化
- 字符串长度
- 练习:
- 练习:

操作字符串

- 获取字符串
- 获取对应的字符数组
- 截取子字符串
- 分割
- 去掉首位空格
- 大小写
- 定位
- 替换

比较字符串

- 是否是同一个对象
- 是否是同一个对象-特例
- 内容是否相同
- 是否以子字符串开始或者结束
- 练习

Stringbuffer

- 追加, 删除, 插入, 反转
- 长度, 容量
- 练习:

装箱拆箱

封装类

所有的**基本类型**，都有对应的**类类型**

比如int对应的类是Integer

这种类就叫做封装类

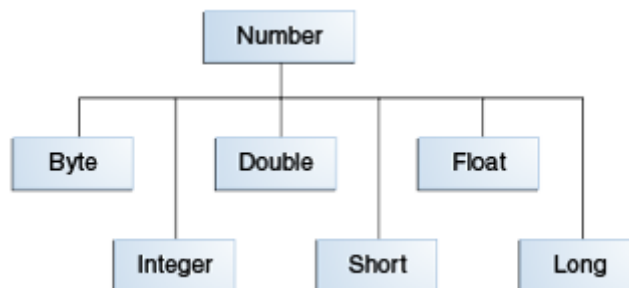
```
1 package digit;
2
3 public class TestNumber {
4     public static void main(String[] args) {
5         int i = 5;
6         //把一个基本类型的变量,转换为Integer对象
7         Integer it = new Integer(i);
8         //把一个Integer对象,转换为一个基本类型的int
9         int i2 = it.intValue();
10    }
11 }
12
```

Number类

数字封装类有

Byte,Short,Integer,Long,Float,Double

这些类都是抽象类Number的子类



```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6         int i = 5;
7
8         Integer it = new Integer(i);
9         //Integer是Number的子类, 所以打印true
10        System.out.println(it instanceof Number);
11    }
12 }
13
```

基本类型转封装类

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6         int i = 5;
7
8         //基本类型转换成封装类型
9         Integer it = new Integer(i);
10
11     }
12 }
13
```

封装类转基本类型

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6         int i = 5;
7
8         //基本类型转换成封装类型
9         Integer it = new Integer(i);
10
11         //封装类型转换成基本类型
12         int i2 = it.intValue();
13
14     }
15 }
16
```

自动装箱

不需要调用构造方法，通过 = 符号自动把 基本类型 转换为 类类型 就叫装箱

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6         int i = 5;
7
8         //基本类型转换成封装类型
9         Integer it = new Integer(i);
10
11         //自动转换就叫装箱
12         Integer it2 = i;
13
14     }
15 }
16
```

```
15 }  
16
```

自动拆箱

不需要调用Integer的intValue方法，通过=就自动转换成int类型，就叫拆箱

```
1  package digit;  
2  
3  public class TestNumber {  
4  
5      public static void main(String[] args) {  
6          int i = 5;  
7  
8          Integer it = new Integer(i);  
9  
10         //封装类型转换成基本类型  
11         int i2 = it.intValue();  
12  
13         //自动转换就叫拆箱  
14         int i3 = it;  
15  
16     }  
17 }  
18
```

int的最大值，最小值

int的最大值可以通过其对应的封装类Integer.MAX_VALUE获取

```
1  package digit;  
2  
3  public class TestNumber {  
4  
5      public static void main(String[] args) {  
6  
7          //int的最大值  
8          System.out.println(Integer.MAX_VALUE);  
9          //int的最小值  
10         System.out.println(Integer.MIN_VALUE);  
11  
12     }  
13 }  
14
```

练习-装箱拆箱

1. 对byte,short,float,double进行自动拆箱和自动装箱
2. byte和Integer之间能否进行自动拆箱和自动装箱
3. 通过Byte获取byte的最大值

```
1 |
```

字符串转换

数字转字符串

方法1：使用String类的静态方法valueOf

方法2：先把基本类型装箱为对象，然后调用对象的toString

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6         int i = 5;
7
8         //方法1
9         String str = String.valueOf(i);
10
11        //方法2
12        Integer it = i;
13        String str2 = it.toString();
14
15    }
16 }
17
```

字符串转数字

调用Integer的静态方法parseInt

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6
7         String str = "999";
8
9         int i = Integer.parseInt(str);
10
11        System.out.println(i);
12
13    }
14 }
15
```

练习-字符串转换

参考上述步骤

把浮点数 3.14 转换为 字符串 "3.14"

再把字符串 "3.14" 转换为 浮点数 3.14

如果字符串是 3.1a4，转换为浮点数会得到什么？

数学方法

java.lang.Math提供了一些常用的数学运算方法，并且都是以静态方法的形式存在

四舍五入, 随机数, 开方, 次方, π , 自然常数

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6         float f1 = 5.4f;
7         float f2 = 5.5f;
8         //5.4四舍五入即5
9         System.out.println(Math.round(f1));
10        //5.5四舍五入即6
11        System.out.println(Math.round(f2));
12
13        //得到一个0-1之间的随机浮点数（取不到1）
14        System.out.println(Math.random());
15
16        //得到一个0-10之间的随机整数（取不到10）
17        System.out.println((int)(Math.random()*10));
18        //开方
19        System.out.println(Math.sqrt(9));
20        //次方（2的4次方）
21        System.out.println(Math.pow(2,4));
22
23        //π
24        System.out.println(Math.PI);
25
26        //自然常数
27        System.out.println(Math.E);
28    }
29 }
30
```

练习-数学方法

这个图是自然对数的计算方式。

借助Math的方法，把自然对数计算出来，看看经过自己计算的自然对数和Math.E的区别有多大

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = e.$$

格式化输出

1: 格式化输出

如果不使用格式化输出，就需要进行字符串连接，如果变量比较多，拼接就会显得繁琐

使用格式化输出，就可以简洁明了

%s 表示字符串

%d 表示数字

%n 表示换行

```

1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6
7         String name ="盖伦";
8         int kill = 8;
9         String title="超神";
10
11         //直接使用+进行字符串连接，编码感觉会比较繁琐，并且维护性差,易读性差
12         String sentence = name+ " 在进行了连续 " + kill + " 次击杀后，获得了 " +
title +" 的称号";
13
14         System.out.println(sentence);
15
16         //使用格式化输出
17         //%s表示字符串，%d表示数字，%n表示换行
18         String sentenceFormat ="%s 在进行了连续 %d 次击杀后，获得了 %s 的称号%n";
19         System.out.printf(sentenceFormat,name,kill,title);
20
21     }
22 }
23

```

2: printf和format

printf和format能够达到一模一样的效果

```

public PrintStream printf(String format, Object ... args) {
    return format(format, args);
}

```

```

1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6
7         String name ="盖伦";
8         int kill = 8;
9         String title="超神";
10
11         String sentenceFormat ="%s 在进行了连续 %d 次击杀后，获得了 %s 的称号%n";
12         //使用printf格式化输出
13         System.out.printf(sentenceFormat,name,kill,title);
14         //使用format格式化输出
15         System.out.format(sentenceFormat,name,kill,title);
16
17     }
18 }
19

```

3: 换行符

换行符就是另起一行 --- '\n' 换行 (newline)

回车符就是回到一行的开头 --- '\r' 回车 (return)

在eclipse里敲一个回车，实际上是**回车换行符**

Java是跨平台的编程语言，同样的代码，可以在不同的平台使用，比如Windows,Linux,Mac

然而在不同的操作系统，换行符是不一样的

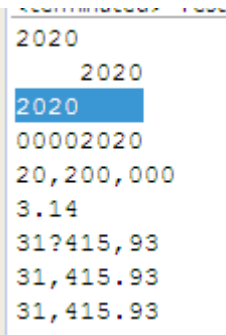
- (1) 在DOS和Windows中，每行结尾是 "\r\n";
- (2) Linux系统里，每行结尾只有 "\n";
- (3) Mac系统里，每行结尾是只有 "\r"。

为了使得同一个java程序的换行符在所有的操作系统中都有一样的表现，使用%n，就可以做到平台无关的换行

```
1 package digit;
2
3 public class TestNumber {
4
5     public static void main(String[] args) {
6
7         System.out.printf("这是换行符%n");
8         System.out.printf("这是换行符%n");
9
10    }
11 }
12
```

4：总长度，左对齐，补0，千位分隔符，小数点位数，本地化表达

其他常用的格式化方式



```
1 package digit;
2
3 import java.util.Locale;
4
5 public class TestNumber {
6
7     public static void main(String[] args) {
8         int year = 2020;
9         //总长度，左对齐，补0，千位分隔符，小数点位数，本地化表达
10
11        //直接打印数字
12        System.out.format("%d\n", year);
13        //总长度是8，默认右对齐
14        System.out.format("%8d\n", year);
15        //总长度是8，左对齐
16        System.out.format("%-8d\n", year);
17    }
18 }
```



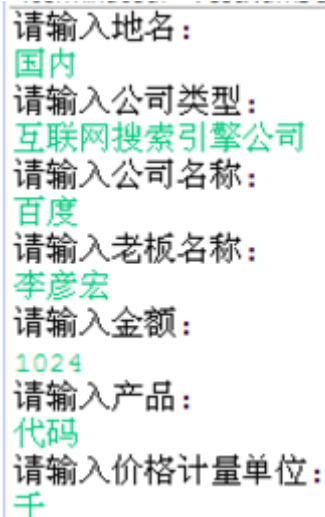
```

17 //总长度是8,不够补0
18 System.out.format("%08d\n",year);
19 //千位分隔符
20 System.out.format("%,8d\n",year*10000);
21
22 //小数点位数
23 System.out.format("%.2f\n",Math.PI);
24
25 //不同国家的千位分隔符
26 System.out.format(Locale.FRANCE,"%,.2f\n",Math.PI*10000);
27 System.out.format(Locale.US,"%,.2f\n",Math.PI*10000);
28 System.out.format(Locale.UK,"%,.2f\n",Math.PI*10000);
29
30 }
31 }
32

```

5: Scanner

借助 Scanner 读取字符串数据，然后用格式化输出任意一段文字，类似以下



```

请输入地名：
国内
请输入公司类型：
互联网搜索引擎公司
请输入公司名称：
百度
请输入老板名称：
李彦宏
请输入金额：
1024
请输入产品：
代码
请输入价格计量单位：
千

```

1 |

字符

保存一个字符的时候使用char

```

1 package character;
2
3 public class TestChar {
4
5     public static void main(String[] args) {
6         char c1 = 'a';
7         char c2 = '1';//字符1,而非数字1
8         char c3 = '中';//汉字字符
9         char c4 = 'ab'; //只能放一个字符
10
11     }
12 }
13

```

char对应的封装类

char对应的封装类是Character 装箱拆箱概念, 参考 装箱装箱

```

1 package character;
2
3 public class TestChar {
4
5     public static void main(String[] args) {
6         char c1 = 'a';
7         Character c = c1; //自动装箱
8         c1 = c;//自动拆箱
9
10    }
11 }
12

```

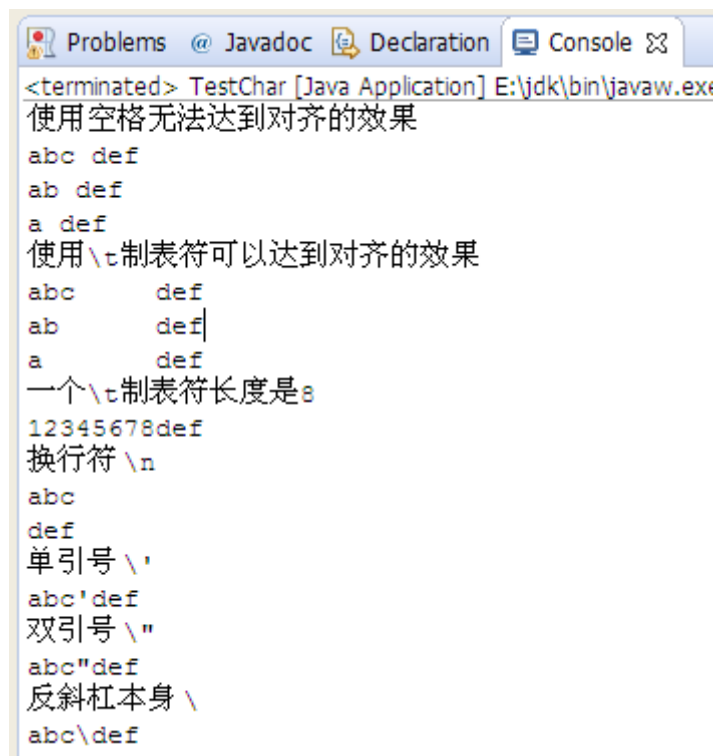
Character常见方法

```

1 package character;
2
3 public class TestChar {
4
5     public static void main(String[] args) {
6
7         System.out.println(Character.isLetter('a'));//判断是否为字母
8         System.out.println(Character.isDigit('a')); //判断是否为数字
9         System.out.println(Character.isWhitespace(' ')); //是否是空白
10        System.out.println(Character.isUpperCase('a')); //是否是大写
11        System.out.println(Character.isLowerCase('a')); //是否是小写
12
13        System.out.println(Character.toUpperCase('a')); //转换为大写
14        System.out.println(Character.toLowerCase('A')); //转换为小写
15
16        String a = 'a'; //不能够直接把一个字符转换成字符串
17        String a2 = Character.toString('a'); //转换为字符串
18
19    }
20 }

```

常见转义



```

<terminated> TestChar [Java Application] E:\jdk\bin\javaw.exe
使用空格无法达到对齐的效果
abc def
ab def
a def
使用\t制表符可以达到对齐的效果
abc      def
ab      def
a      def
一个\t制表符长度是8
12345678def
换行符 \n
abc
def
单引号 \'
abc'def
双引号 \"
abc"def
反斜杠本身 \
abc\def

```

```

1 package character;
2
3 public class TestChar {
4
5     public static void main(String[] args) {
6         System.out.println("使用空格无法达到对齐的效果");
7         System.out.println("abc def");
8         System.out.println("ab def");
9         System.out.println("a def");
10
11         System.out.println("使用\t制表符可以达到对齐的效果");
12         System.out.println("abc\tdef");
13         System.out.println("ab\tdef");
14         System.out.println("a\tdef");
15
16         System.out.println("一个\t制表符长度是8");
17         System.out.println("12345678def");
18
19         System.out.println("换行符 \n");
20         System.out.println("abc\ndef");
21
22         System.out.println("单引号 \'");
23         System.out.println("abc'def");
24         System.out.println("双引号 \"");
25         System.out.println("abc\"def");
26         System.out.println("反斜杠本身 \\");
27         System.out.println("abc\\def");
28     }
29 }
30

```

练习-character

通过Scanner从控制台读取字符串，然后把字符串转换为字符数组

参考的转换方式:

```
String str = "abc123"; char[] cs = str.toCharArray();
```

转换为字符数组后，筛选出控制台读取到的字符串中的大写字母和数字，并打印出来

```
<terminated> TestChar [Java A;  
Hf1860JghlabfiU57Dv  
H860JU57D
```

字符串

创建字符串

字符串即字符的组合，在Java中，字符串是一个类，所以我们见到的字符串都是对象
常见创建字符串手段：

1. 每当有一个**字面值**出现的时候，虚拟机就会创建一个字符串
2. 调用String的构造方法创建一个字符串对象
3. 通过+加号进行字符串拼接也会创建新的字符串对象

```
1 package character;  
2  
3 public class TestString {  
4  
5     public static void main(String[] args) {  
6         String garen = "盖伦"; //字面值,虚拟机碰到字面值就会创建一个字符串对象  
7  
8         String teemo = new String("提莫"); //创建了两个字符串对象  
9  
10        char[] cs = new char[]{'崔','斯','特'};  
11  
12        String hero = new String(cs); // 通过字符数组创建一个字符串对象  
13  
14        String hero3 = garen + teemo; // 通过+加号进行字符串拼接  
15    }  
16 }  
17
```

final

String 被修饰为final,所以是不能被继承的

代码比较复制代码

```
1 package character;  
2  
3 public class TestString {  
4  
5     public static void main(String[] args) {
```

```

6         MyString str = new MyString();
7
8     }
9
10    /*这里会报错，因为String不能被继承*/
11    static class MyString extends String{
12
13    }
14
15 }
16

```

immutable

immutable 是指不可改变的

比如创建了一个字符串对象

String garen ="盖伦";

不可改变的具体含义是指：

不能增加长度

不能减少长度

不能插入字符

不能删除字符

不能修改字符

一旦创建好这个字符串，里面的内容 **永远** 不能改变

```

1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6         String garen ="盖伦";
7
8     }
9 }
10

```

字符串格式化

如果不使用字符串格式化，就需要进行字符串连接，如果变量比较多，拼接就会显得繁琐

使用**字符串格式化**，就可以简洁明了

```

1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String name ="盖伦";
8         int kill = 8;
9         String title="超神";
10
11        /*直接使用+进行字符串连接，编码感觉会比较繁琐，并且维护性差，易读性差
12        String sentence = name+ " 在进行了连续 " + kill + " 次击杀后，获得了 " +
13        title +" 的称号";
14
15    }
16 }
17

```

```

14         System.out.println(sentence);
15
16         //格式化字符串
17         // %s表示字符串, %d表示数字, %n表示换行
18         String sentenceFormat = "%s 在进行了连续 %d 次击杀后, 获得了 %s 的称号%n";
19
20         String sentence2 = String.format(sentenceFormat, name, kill, title);
21
22         System.out.println(sentence2);
23
24     }
25 }
26

```

字符串长度

length方法返回当前字符串的长度可以有长度为0的字符串,即空字符串

```

1  package character;
2
3  public class TestString {
4
5      public static void main(String[] args) {
6
7          String name = "盖伦";
8
9          System.out.println(name.length());
10
11         String unknowHero = "";
12
13         //可以有长度为0的字符串,即空字符串
14         System.out.println(unknowHero.length());
15
16     }
17 }
18

```

练习:

创建一个长度是5的随机字符串, 随机字符有可能是数字, 大写字母或者小写字母

给点提示: 数字和字符之间可以通过互相转换

```

1  char c = 'A';
2  short s = (short) c;

```

通过这个手段就能够知道字符 a-z A-Z 0-9 所对应的数字的区间了

练习:

创建一个长度是8的字符串数组

使用8个长度是5的随机字符串初始化这个数组

对这个数组进行排序, 按照每个字符串的首字母排序(无视大小写)

注1: 不能使用Arrays.sort() 要自己写

注2: 无视大小写, 即 Axxxx 和 axxxx 没有先后顺序

操作字符串

获取字符串

charAt(int index)获取指定位置的字符

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String sentence = "盖伦,在进行了连续8次击杀后,获得了 超神 的称号";
8
9         char c = sentence.charAt(0);
10
11         System.out.println(c);
12
13     }
14 }
15
```

获取对应的字符数组

toCharArray() 获取对应的字符数组

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String sentence = "盖伦,在进行了连续8次击杀后,获得了超神 的称号";
8
9         char[] cs = sentence.toCharArray(); //获取对应的字符数组
10
11         System.out.println(sentence.length() == cs.length);
12
13     }
14 }
15
```

截取子字符串

substring 截取子字符串

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String sentence = "盖伦,在进行了连续8次击杀后,获得了 超神 的称号";
8
9     }
10 }
11
```

```

9      //截取从第3个开始的字符串 （基0）
10     String subString1 = sentence.substring(3);
11
12     System.out.println(subString1);
13
14     //截取从第3个开始的字符串 （基0）
15     //到5-1的位置的字符串
16     //左闭右开
17     String subString2 = sentence.substring(3,5);
18
19     System.out.println(subString2);
20
21 }
22 }
23

```

分割

split 根据分隔符进行分隔

```

1  package character;
2
3  public class TestString {
4
5      public static void main(String[] args) {
6
7          String sentence = "盖伦,在进行了连续8次击杀后,获得了 超神 的称号";
8
9          //根据,进行分割,得到3个子字符串
10         String subSentences[] = sentence.split(",");
11         for (String sub : subSentences) {
12             System.out.println(sub);
13         }
14
15     }
16 }
17

```

去掉首位空格

trim 去掉首尾空格

```

1  package character;
2
3  public class TestString {
4
5      public static void main(String[] args) {
6
7          String sentence = "      盖伦,在进行了连续8次击杀后,获得了 超神 的称号
8          ";
9
10         System.out.println(sentence);
11         //去掉首尾空格
12         System.out.println(sentence.trim());
13     }
14 }

```


大小写

toLowerCase 全部变成小写

toUpperCase 全部变成大写

```

1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String sentence = "Garen";
8
9         //全部变成小写
10        System.out.println(sentence.toLowerCase());
11        //全部变成大写
12        System.out.println(sentence.toUpperCase());
13
14    }
15 }
16

```

定位

indexOf 判断字符或者子字符串出现的位置

contains 是否包含子字符串

```

1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String sentence = "盖伦,在进行了连续8次击杀后,获得了超神 的称号";
8
9         System.out.println(sentence.indexOf('8')); //字符第一次出现的位置
10
11        System.out.println(sentence.indexOf("超神")); //字符串第一次出现的位置
12
13        System.out.println(sentence.lastIndexOf("了")); //字符串最后出现的位置
14
15        System.out.println(sentence.indexOf(',',5)); //从位置5开始,出现的第一
16        次,的位置
17
18        System.out.println(sentence.contains("击杀")); //是否包含字符串"击杀"
19
20    }
21 }

```

替换

replaceAll 替换所有的
replaceFirst 只替换第一个

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String sentence = "盖伦,在进行了连续8次击杀后,获得了超神 的称号";
8
9         String temp = sentence.replaceAll("击杀", "被击杀"); //替换所有的
10
11         temp = temp.replaceAll("超神", "超鬼");
12
13         System.out.println(temp);
14
15         temp = sentence.replaceFirst(",", ""); //只替换第一个
16
17         System.out.println(temp);
18
19     }
20 }
21
```

比较字符串

是否是同一个对象

str1和str2的内容一定是一样的!
但是, 并不是同一个字符串对象

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String str1 = "the light";
8
9         String str2 = new String(str1);
10
11         //==用于判断是否是同一个字符串对象
12         System.out.println( str1 == str2);
13
14     }
15
16 }
17
```

是否是同一个对象-特例

```
str1 = "the light";
```

```
str3 = "the light";
```

一般说来，编译器每碰到一个字符串的字面值，就会创建一个新的对象

所以在第6行会创建了一个新的字符串"the light"

但是在第7行，编译器发现已经存在现成的"the light"，那么就拿来使用，而没有进行重复创建

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6         String str1 = "the light";
7         String str3 = "the light";
8         System.out.println( str1 == str3);
9     }
10
11 }
12
```

内容是否相同

使用equals进行字符串内容的比较，必须大小写一致

equalsIgnoreCase，忽略大小写判断内容是否一致

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6
7         String str1 = "the light";
8
9         String str2 = new String(str1);
10
11        String str3 = str1.toUpperCase();
12
13        //==用于判断是否是同一个字符串对象
14        System.out.println( str1 == str2);
15
16        System.out.println(str1.equals(str2)); //完全一样返回true
17
18        System.out.println(str1.equals(str3)); //大小写不一样，返回false
19        System.out.println(str1.equalsIgnoreCase(str3)); //忽略大小写的比较，返回
20        true
21    }
22
23 }
24
```

是否以子字符串开始或者结束

startsWith //以...开始

endsWith //以...结束

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6         String str1 = "the light";
7
8         String start = "the";
9         String end = "Ight";
10
11         System.out.println(str1.startsWith(start)); //以...开始
12         System.out.println(str1.endsWith(end)); //以...结束
13
14     }
15
16 }
17
```

练习

创建一个长度是100的字符串数组

使用长度是2的随机字符填充该字符串数组

统计这个字符串数组里重复的字符串有多少种

```
Ri Ud ey iO a5 Zw BF ZV gV JM Ye ZL 6N mz bC p5 IJ yr KY SH
lt QT fs 3G 5J Oq do aV Wp 2C ca wp WK Ux do kl 2X FK 4H c8
1V 74 7j 3w o3 eC w0 vK 9f 8V Sg Om FM x3 YF YT em AX 7V EV
IOG rM W2 SU Ao fy On IQ T9 qw lT EN za 9j 2w SK jA Ck 8H Oj
kk tg Hq 5k oz sG Rh Dc WM fT Zn OD aS MI xb Ip Ke RK RK cI
总共有 5种重复的字符串
分别是:
lt do Wp Sg RK
```

1 |

Stringbuffer

追加，删除，插入，反转

append追加

delete 删除

insert 插入

reverse 反转

```
1 package character;
2
3 public class TestString {
4
5     public static void main(String[] args) {
6         String str1 = "let there ";
7
```

```

8      StringBuffer sb = new StringBuffer(str1); //根据str1创建一个
StringBuffer对象
9      sb.append("be light"); //在最后追加
10
11     System.out.println(sb);
12
13     sb.delete(4, 10); //删除4-10之间的字符
14
15     System.out.println(sb);
16
17     sb.insert(4, "there "); //在4这个位置插入 there
18
19     System.out.println(sb);
20
21     sb.reverse(); //反转
22
23     System.out.println(sb);
24
25 }
26
27 }
28

```

长度，容量

为什么StringBuffer可以变长?

和String内部是一个字符数组一样，StringBuffer也维护了一个字符数组。但是，这个字符数组，留有冗余长度

比如说new StringBuffer("the")，其内部的字符数组的长度，是19，而不是3，这样调用插入和追加，在现成的数组的基础上就可以完成了。

如果追加的长度超过了19，就会分配一个新的数组，长度比原来多一些，把原来的数据复制到新的数组中，看上去 数组长度就变长了

length: "the"的长度 3

capacity: 分配的总空间 19

注：19这个数量，不同的JDK数量是不一样的

```

1  package character;
2
3  public class TestString {
4
5      public static void main(String[] args) {
6          String str1 = "the";
7
8          StringBuffer sb = new StringBuffer(str1);
9
10         System.out.println(sb.length()); //内容长度
11
12         System.out.println(sb.capacity()); //总空间
13
14     }
15
16 }

```

练习:

String与StringBuffer的性能区别?

生成10位长度的随机字符串

然后,先使用String的+,连接10000个随机字符串,计算消耗的时间

然后,再使用StringBuffer连接10000个随机字符串,计算消耗的时间

提示: 使用System.currentTimeMillis() 获取当前时间(毫秒)

使用字符串连接+的方式, 连接10000次, 耗时1625毫秒
使用StringBuffer的方式, 连接1000000次, 耗时187毫秒

练习

根据接口IStringBuffer , 自己做一个MyStringBuffer

```

1  package character;
2
3  public interface IStringBuffer {
4      public void append(String str); //追加字符串
5      public void append(char c); //追加字符
6      public void insert(int pos,char b); //指定位置插入字符
7      public void insert(int pos,String b); //指定位置插入字符串
8      public void delete(int start); //从开始位置删除剩下的
9      public void delete(int start,int end); //从开始位置删除结束位置-1
10     public void reverse(); //反转
11     public int length(); //返回长度
12 }
13 package character;
14 public class MyStringBuffer implements IStringBuffer{
15 }
16

```