# day02

## StringBuilder类

由于String是不变对象,每次修改内容都要创建新对象,因此String不适合做频繁修改操作.为了解决这个问题,java提供了StringBuilder类.

```
package string;

/**
 * 频繁修改字符串带来的性能损耗.
 */
public class StringDemo2 {
    public static void main(String[] args) {
        String str = "a";
        for(int i=0;i<10000000;i++){
            str = str + str;
        }
        System.out.println("执行完毕!");
    }
}
```

StringBuilder是专门用来修改String的一个API,内部维护一个可变的char数组,修改都是在这个数组上进行的,
内部会自动扩容.修改速度和性能开销优异.并且提供了修改字符串的常见操作对应的方法:增删改插

```
package string;

/**
 * java.lang.StringBuilder
 * 专门用来修改String的一个API,内部维护一个可变的char数组,修改都是在这个数组上进行的,
 * 内部会自动扩容.修改速度和性能开销优异.并且提供了修改字符串的常见操作对应的方法:增删改插
 */
public class StringBuilderDemo1 {
    public static void main(String[] args) {
        String str = "好好学习java";

        //内部默认表示一个空字符串
//        StringBuilder builder = new StringBuilder();

        //复制给定字符串到StringBuilder内部
//        StringBuilder builder = new StringBuilder(str);//不是线程安全的
        StringBuffer builder = new StringBuffer(str);//是线程安全的
        /*
            好好学习java
            好好学习java,为了找个好工作!
            append:追加内容
         */
        builder.append(",为了找个好工作!");
        System.out.println(builder);//输出StringBuilder的内容
```

```
        //通过调用toString方法将StringBuilder内容以字符串形式返回.
        str = builder.toString();
        System.out.println(str);

        /*
            好好学习java,为了找个好工作!
            好好学习java,就是为了改变世界!
            replace:替换部分内容
         */
        builder.replace(9,16,"就是为了改变世界");
        System.out.println(builder);

        /*
            好好学习java,就是为了改变世界!
            ,就是为了改变世界!
            delete:删除部分内容
         */
        builder.delete(0,8);
        System.out.println(builder);

        /*
            ,就是为了改变世界!
            活着,就是为了改变世界!
            insert:插入操作
         */
        builder.insert(0,"活着");
        System.out.println(builder);

        //翻转字符串
        builder.reverse();
        System.out.println(builder);
    }
}
```

StringBuffer 和StringBuilder

– StringBuffer是线程安全的，同步处理的，性能稍慢

–StringBuilder是非线程安全的，并发处理的，性能稍快

```
package string;

/**
 * StringBuilder修改字符串的性能
 */
public class StringBuilderDemo2 {
    public static void main(String[] args) {
        StringBuilder builder = new StringBuilder("a");
        for(int i=0;i<10000000;i++){
            builder.append("a");
        }
        System.out.println("执行完毕!");
    }
}
```

# 正则表达式

正则表达式是用来描述一个字符串的内容格式,使用它通常用来匹配一个字符串的内容是否符合格式要求.

## 基本语法

[]:表示一个字符,该字符可以是[]中指定的内容

例如:

**预定义字符**

.:"."表示任意一个字符,没有范围限制

\d:表示任意一个数字,等同于[0-9]

\w:表示任意一个单词字符,等同于[a-zA-Z0-9_]

\s:表示任意一个空白字符.

\D:表示不是数字

\W:不是单词字符

\S:不是空白字符

**量词:**

- ?:表示前面的内容出现0-1次

  例如:

  [abc]? 可以匹配:a 或 b 或 c 或什么也不写
- +:表示前面的内容出现1次以上

  [abc]+ 可以匹配:aaaaaaaaaa...或abcabcbabcbabcbabcbabbabab....

  但是不能匹配:什么都不写或abcfdfsbbaqbb34bbwer...
- *:表示前面的内容出现任意次(0-多次)

  匹配内容与+一致,只是可以一次都不写.
- {n}:表示前面的内容出现n次

  例如:

  [abc]{3} 可以匹配:aaa 或 bbb 或 aab

  不能匹配:aaaa或aad
- {n,m}:表示前面的内容出现最少n次最多m次

  [abc]{3,5} 可以匹配:aaa 或  abcab 或者 abcc

  不能匹配:aaaaaa 或 aabbd
- {n,}:表示前面的内容出现n次以上(含n次)

  [abc]{3,} 可以匹配:aaa 或 aaaaa.... 或 abcbabbcbabcbabcba....

  不能匹配:aa 或 abbdaw...
- ()用于分组,是将括号内的内容看做是一个整体

例如:

(abc){3} 表示abc整体出现3次. 可以匹配abcabcabc.

不能匹配aaa 或abcabc

(abc|def){3}表示abc或def整体出现3次.

可以匹配: abcabcabc 或 defdefdef 或 abcdefabc

## String支持正则表达式的相关方法

### matches方法

boolean matches(String regex)

使用给定的正则表达式验证当前字符串是否满足格式要求,满足则返回true.否则返回false

```
package string;

public class MatchesDemo {
    public static void main(String[] args) {
        /*
            邮箱的正则表达式
            用户名@域名
            fancq@tedu.cn
            [a-zA-Z0-9_]+@[a-zA-Z0-9]+(\.[a-zA-Z]+)+
         */
        String mail = "fancq@tedu.cn";
        String regex = "[a-zA-Z0-9_]+@[a-zA-Z0-9]+(\\.[a-zA-Z]+)+";
        boolean match = mail.matches(regex);
        if(match){
            System.out.println("是邮箱");
        }else{
            System.out.println("不是邮箱");
        }
    }
}
```

### split方法

String[] split(String regex)

将当前字符串按照满足正则表达式的部分进行拆分,将拆分后的每部分以数组形式返回.

```
package string;

import java.util.Arrays;

public class SplitDemo {
    public static void main(String[] args) {
        String str = "abc123def456ghi";
        //按照数字部分拆分,获取其中每部分字母
        String[] arr = str.split("[0-9]+");
        System.out.println(arr.length);
        System.out.println(Arrays.toString(arr));
```

```
        str = "123,456,789,023";
        //拆分出所有的数字部分
        arr = str.split(",");
        System.out.println(Arrays.toString(arr));
        //如果连续遇到拆分项,则会拆分出一个空字符串.但是在字符串末尾连续遇到则忽略.
        str = ",,,123,,,456,789,023,,,,";
        //拆分出所有的数字部分
        arr = str.split(",");
        System.out.println(Arrays.toString(arr));

        str = "123.456.789.023";
        //拆分出所有的数字部分
        arr = str.split("\\.");//.在正则表达式中表示任意字符,要注意转意!
        System.out.println(Arrays.toString(arr));
    }
}
```

**replaceAll方法**

String replaceAll(String regex,String str)

将当前字符串中满足正则表达式的部分替换为给定内容

```
package string;

public class ReplaceAllDemo {
    public static void main(String[] args) {
        String str = "abc123def456ghi";
        //将当前字符串中的数字部分替换为#NUMBER#
        str = str.replaceAll("[0-9]+","#NUMBER#");
        System.out.println(str);
    }
}
```

# Object类

Object是所有类的顶级超类,其中有两个经常被子类重写的方法:

toString()与equals().

## 编写Point类进行测试

```
package object;

import java.util.Objects;

/**
 * 使用当前类测试超类Object中经常被子类重写的方法:equals与toString
 *
 * Point类设计目的是表示直角坐标系中的一个点
 */
public class Point {
```

```
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Point point = (Point) o;
        return x == point.x && y == point.y;
    }

    @Override
    public int hashCode() {
        return Objects.hash(x, y);
    }

    @Override
    public String toString() {
        return "Point{" +
                "x=" + x +
                ", y=" + y +
                '}';
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

}
```

编写测试类

```
package object;

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

```
        Point p = new Point(1,2);
        /*
            System.out.println(Object o)
            将给定对象toString方法返回返回的字符串输出到控制台

            toString方法是Object提供的方法，该方法默认返回的字符串为:
            类名@地址
         */
        System.out.println(p);
//          System.out.println(p.toString());
        /*
            任何类型与字符串链接结果都是字符串
            这意味着链接的类型会被转为字符串再进行链接。其他类型是如何
            转换为String?就是依靠toString方法
         */
        String line = "当前对象是:" + p;
        System.out.println(line);


        Point p2 = new Point(1,2);
        System.out.println("p2:"+p2);
        //==对于引用类型的意义是比较是否为同一个对象
        System.out.println(p==p2);//false  两个不同的对象
        //equals则是比较两个对象"像不像"(对象内容，特征是否一致)
        System.out.println(p.equals(p2));//true 两个对象的内容相同

    }
}
```

## 包装类

java定义了8个包装类,目的是为了解决基本类型不能直接参与面向对象开发的问题,使得基本类型可以通过包装类的实例以对象的形式存在.

- 其中数字类型的包装类都继承自java.lang.Number,而char和boolean的包装类直接继承自Object
- Number是一个抽象类,定义了一些方法,目的是让包装类可以将其表示的基本类型转换为其他数字类型.

```
package integer;

public class IntegerDemo1 {
    public static void main(String[] args) {
        //基本类型转换为包装类
        int i = 123;
        //java推荐我们使用包装类的静态方法valueOf将基本类型转换为包装类,而不是直接new
        Integer i1 = Integer.valueOf(i);//Integer会重用-128-127之间的整数对象
        Integer i2 = Integer.valueOf(i);
        System.out.println(i1==i2);//true
        System.out.println(i1.equals(i2));//true

        double dou = 123.123;
        Double dou1 = Double.valueOf(dou);//Double则是直接new
        Double dou2 = Double.valueOf(dou);
```

```
        System.out.println(dou1==dou2);//false
        System.out.println(dou1.equals(dou2));//true

        //包装类转换为基本类型
        int in = i1.intValue();//获取包装类对象中表示的基本类型值
        double doub = i1.doubleValue();
        System.out.println(in);//123
        System.out.println(doub);//123.0

        in = dou1.intValue();//大类型转小类型可能存在丢精度!
        doub = dou1.doubleValue();
        System.out.println(in);//123
        System.out.println(doub);//123.123
    }
}
```

**包装类常用功能**

```
package integer;

public class IntegerDemo2 {
    public static void main(String[] args) {
        //1可以通过包装类获取其表示的基本类型的取值范围
        //获取int的最大值和最小值?
        int imax = Integer.MAX_VALUE;
        System.out.println(imax);
        int imin = Integer.MIN_VALUE;
        System.out.println(imin);

        long lmax = Long.MAX_VALUE;
        System.out.println(lmax);
        long lmin = Long.MIN_VALUE;
        System.out.println(lmin);

        /*
            2字符串转换为基本类型的前提是该字符串正确描述了基本类型可以保存的值,否则
            会抛出异常:NumberFormatException
         */
        String str = "123";
//        String str = "123.123";//这个字符串不能解析为int值!
        int d = Integer.parseInt(str);
        System.out.println(d);//123
        double dou = Double.parseDouble(str);
        System.out.println(dou);//123.123
    }
}
```

**自动拆装箱特性**

JDK5之后推出了一个新的特性:自动拆装箱

该特性是编译器认可的.当编译器编译源代码时发现有基本类型和引用类型相互赋值使用时会自动补充代码来完成他们的转换工作,这个过程称为自动拆装箱.

```
package integer;
```

```
public class IntegerDemo3 {
    public static void main(String[] args) {
        /*
            触发自动拆箱特性,编译器会补充代码将包装类转换为基本类型,下面的代码会变为:
            int i = new Integer(123).intValue();
         */
        int i = new Integer(123);
        /*
            触发编译器自动装箱特性,代码会被编译器改为:
            Integer in = Integer.valueOf(123);
         */
        Integer in = 123;
    }
}
```