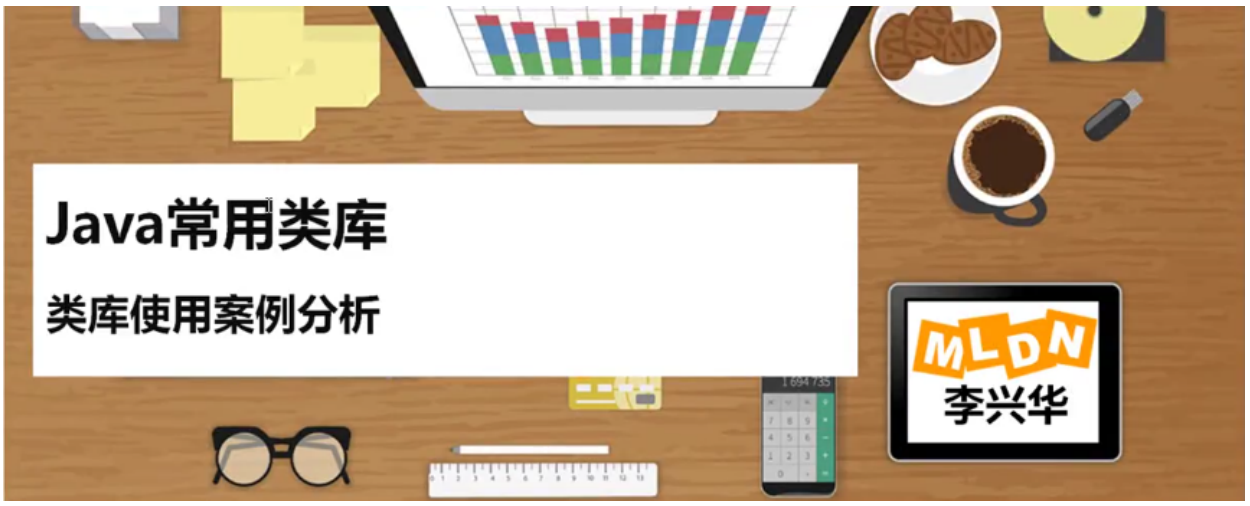

博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

■类库案例分析一

定义一个StringBuffer类对象，然后通过append()方法向对象中添加26个小写字母，要求每次只添加一次，共添加26次，然后按照逆序的方式输出，并且可以删除前5个字符。

本操作主要是训练StringBuffer类中的处理方法，因为StringBuffer的主要特点是内容允许修改。

```
public class JavaAPIDemo {
	public static void main(String[] args) {
		StringBuffer buf = new StringBuffer() ;
		for (int x = 'a' ; x <= 'z' ; x ++) {	// 直接循环设置
			buf.append((char) x) ; // 保存字符
		}
		buf.reverse().delete(0, 5) ; // 反转处理
		System.out.println(buf);
	}
}
```

因为StringBuffer的内容是允许修改的，而String内容不允许修改，现在的程序是一个单线程的开发，所以不需要去考虑所谓的并发访问问题。

■类库案例分析二

利用Random类产生5个1～30之间（包括1和20）的随机整数。

Random产生随机数的操作之中包含有数字0，所以此时不应该存在有数字0的问题。

```
import java.util.Arrays;
import java.util.Random;
public class JavaAPIDemo {
	public static void main(String[] args) {
```

```
        int result [] = NumberFactory.create(5) ;
        System.out.println(Arrays.toString(result));
    }
}
class NumberFactory {
    private static Random random = new Random() ;
    /**
     * 通过随机数来生成一个数组的内容，该内容不包括有0
     * @param len 要开辟的数组大小
     * @return 包含有随机数的内容
     */
    public static int [] create(int len) {
        int data [] = new int [len] ;  // 开辟新的数组
        int foot = 0 ;
        while (foot < data.length) {
            int num = random.nextInt(30) ;
            if(num != 0) {
                data[foot ++] = num ; // 保存数据
            }
        }
        return data ;
    }
}
```

■类库案例分析三

输入一个Email地址，然后使用正则表达式验证该Email地址是否正确。

对于此时的输入可以通过命令参数实现数据的输入，如果要想进行验证，最好的做法是设置一个单独的验证处理类。

```
public class JavaAPIDemo {
    public static void main(String[] args) {
        if (args.length != 1) {       // 输入有一个参数
            System.out.println("程序执行错误，没有输入初始化参数，正确格式为：java
JavaAPIDemo EMAIL地址");
            System.exit(1); // 系统退出
        }
        String email = args[0] ; // 获取初始化参数
        if (Validator.isEmail(email)) {
            System.out.println("输入的email地址正确！");
        } else {
            System.out.println("输入的email地址错误！");
        }
    }
}
class Validator {      // 定义一个专门的验证程序类
    private Validator() {}
    public static boolean isEmail(String email) {
```

```
        if (email == null || "".equals(email)) {     // 数据为空
            return false ;
        }
        String regex = "\\w+@\\w+\\.\\w+" ;
        return email.matches(regex) ;
    }
}
```

如果以后要有更多的验证，只需要在Validator类之中扩展方法即可。

■类库案例分析四

编写程序，用0～1之间的随机数来模拟扔硬币试验，统计扔1000次后出现正、反面的次数并输出。

```
import java.util.Random;
public class JavaAPIDemo {
    public static void main(String[] args) {
        Coin coin = new Coin() ;
        coin.throwCoin(1000);
        System.out.println("正面出现次数：" + coin.getFront() + "、背面出现次数：" +
coin.getBack());
    }
}

class Coin {     // 模拟硬币的扔的操作
    private int front ; // 保存正面次数
    private int back ; // 保存背面次数
    private Random random = new Random() ;
    /**
     * 扔硬币的处理
     * @param num 扔硬币的执行次数
     */
    public void throwCoin(int num) {
        for (int x = 0 ; x < num ; x ++) {
            int temp = random.nextInt(2) ;
            if (temp == 0) {
                this.front ++ ;
            } else {
                this.back ++ ;
            }
        }
    }
    public int getFront() {
        return this.front ;
    }
    public int getBack() {
        return this.back ;
    }
}
```

■类库案例分析五

编写正则表达式，判断给定的是否是一个合法的IP地址。

IP地址的组成就是数字，对于数字的组成有一个基础的要求，第一位的内容只能是无、1、2，第二位的内容可以0~9，第三位的内容可以0~9。

```
public class JavaAPIDemo {
	public static void main(String[] args) {
		String str = "192.168.1.299" ;
		System.out.println(Validator.validateIP(str));
	}
}
class Validator {
	public static boolean validateIP(String ip) {
		if (ip == null || "".equals(ip)) {
			return false ;
		}
		String regex = "([12]?[0-9]?[0-9]\\.){3}([12]?[0-9]?[0-9])" ;
		if(ip.matches(regex)) {	// 验证通过，还需要对IP地址进行拆分处理
			String result [] = ip.split("\\.") ; // 拆分数据
			for (int x = 0 ; x < result.length ; x ++) {
				int temp = Integer.parseInt(result[x]) ;
				if (temp > 255) {
					return false ;
				}
			}
		} else {
			return false ;
		}
		return true ;
	}
}
```

■类库案例分析六

给定下面的HTML代码：

```
<font face="Arial,Serif" size="+2" color="red">
	要求对内容进行拆分，拆分之后的结果是：
		face Arial,Serif
		size +2
		color red
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class JavaAPIDemo {
```

```
    public static void main(String[] args) {
        String str = "<font face=\"Arial,Serif\" size=\"+2\" color=\"red\">" ;
        String regex = "\\w+=\"[a-zA-Z0-9,\\+]+\"" ;
        Matcher matcher = Pattern.compile(regex).matcher(str) ;
        while(matcher.find()) {
            String temp = matcher.group(0) ;
            String result [] = temp.split("=") ;
            System.out.println(result[0] + "\t" + result[1].replaceAll("\"", ""));
        }
    }
}
```

■类库案例分析七

编写程序，实现国际化应用，从命令行输入国家的代号，例如，1表示中国，2表示美国，然后根据输入代号的不同调用不同的资源文件显示信息。

本程序的实现肯定要通过Locale类的对象来指定区域，随后利用ResourceBundle类加载资源文件，而对于数据的输入可以继续使用初始化参数形式来完成。

1、定义中文的资源文件：cn.mldn.message.Messages_zh_CN.properties

```
info=感谢小强同学请客吃冰激凌!
```

2、定义英文的资源文件：cn.mldn.message.Messages_en_US.properties

```
info=Thanks Qiang , your ice very nice !
```

3、定义程序类进行加载控制；

```
import java.util.Locale;
import java.util.ResourceBundle;
public class JavaAPIDemo {
    public static void main(String[] args) {
        if (args.length != 1) {       // 没有得到输入参数
            System.out.println("程序执行错误，没有设置区域编码，正确格式：java
JavaAPIDemo 选择项");
            System.exit(1);
        }
        int choose = Integer.parseInt(args[0]) ;
        System.out.println(new MessageUtil().getMessage(choose));
    }
}
class MessageUtil {
    public static final int CHINA = 1;
    public static final int USA = 2;
    private static final String KEY = "info";
    private static final String BASENAME = "cn.mldn.message.Messages";
    public String getMessage(int num) {
        Locale loc = this.getLocale(num) ;
```

```
            if (loc == null) {
                return "Nothing" ;
            } else {
                return ResourceBundle.getBundle(BASENAME, loc).getString(KEY) ;
            }
        }
        private Locale getLocale(int num) {
            switch (num) {
            case CHINA:
                return new Locale("zh", "CN");
            case USA:
                return new Locale("en", "US");
            default:
                return null;
            }
        }
}
```

■类库案例分析八

按照“姓名：年龄：成绩|姓名：年龄：成绩”的格式定义字符串“张三:21:98|李四:22:89|王五20:70”，要求将每组值分别保存在Student对象之中，并对这些对象进行排序，排序的原则为：按照成绩由高到低排序，如果成绩相等，则按照年龄由低到高排序。

本程序最典型的做法是直接利用比较器完成处理，如果不适用比较器也可以完成，相当于自己采用冒泡的方法进行排列，使用了比较器就可以利用Arrays类做处理。

```
import java.util.Arrays;
public class JavaAPIDemo {
        public static void main(String[] args) {
            String input = "张三:21:98|李四:22:89|王五:20:70" ;
            String result[] = input.split("\\|") ;
            Student students [] = new Student [result.length] ;
            for (int x = 0 ; x < result.length ; x ++) {
                String temp [] = result[x].split(":") ;
                students[x] = new
Student(temp[0],Integer.parseInt(temp[1]),Double.parseDouble(temp[2])) ;
            }
            Arrays.sort(students);
            System.out.println(Arrays.toString(students));
        }
}
class Student implements Comparable<Student>{
        private String name ;
        private int age ;
        private double score ;
        public Student(String name, int age, double score) {
            super();
```

```
            this.name = name;
            this.age = age;
            this.score = score;
        }
        @Override
        public int compareTo(Student stu) {
            if (this.score < stu.score) {
                return 1 ;
            } else if (this.score > stu.score) {
                return -1 ;
            } else {
                return this.age - stu.age ;
            }
        }
        @Override
        public String toString() {
            return "【学生信息】姓名：" + this.name + "、年龄：" + this.age + "、成绩：" +
this.score + "\n" ;
        }
}
```

结构化字符串处理：“内容|内容|”，如果有复杂的情况内容里面可能再有其它标记。