第9章 常用核心类及API

学习目标

- [ ] 了解数学相关API
- [ ] 了解日期时间API
- [ ] 了解系统类API
- [ ] 掌握数组工具类的使用
- [ ] 熟练掌握String类的API
- [ ] 熟练掌握StringBuilder和StringBuffer类的API
- [ ] 能够处理字符串相关的算法处理
- [ ] 认识枚举类型
- [ ] 会使用枚举类型
- [ ] 认识包装类
- [ ] 认识包装类
- [ ] 会使用包装类进行处理字符串
- [ ] 会分析包装类的相关面试题

第九章 常用核心类及API

9.1 字符串String

Java中的String是一个引用数据类型，表示字符串，String也是一个类，位于java.lang包下。Java程序中所有的字符串字面量（如`"abc"`）都可以被看作是此类的实例。因为字符串太常用了，所以Java提供了这种简单的字符串字面量的表示方式。

9.1.1 字符串的特点

1. Java字符串的一个重要特点就是字符串**不可变**。

   这种不可变性是通过内部的`private final char[]`字段，以及没有任何修改`char[]`的方法实现的。修改一个字符串变量值，相当于新生成一个字符串对象。

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

2. 上面这个字符数组`private final char value[];`也是字符串对象的内部存储形式。

   > JDK1.9之前有一个char[] value数组，JDK1.9之后byte[]数组

   所以`"abc"`等效于`char[] data={ 'a' , 'b' , 'c' }`。

例如：

```
String str = "abc";
```

相当于：

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
// String底层是靠字符数组实现的。
```

3. 字符串字面量也是一个String类的实例，存储在字符串常量池中，相同的字符串字面量表示的对象在内存中只有一份。

```
String s1 = "abc";
String s2 = "abc";
System.out.println(s1 == s2);
// 内存中只有一个"abc"对象被创建，同时被s1和s2共享。
```

4. 字符串String类型本身是final声明的，意味着我们不能继承String，也就意味着我们不能去重写他的方法。

9.1.2 构造字符串对象

1. 使用构造方法

- `public String()`：初始化新创建的 String对象，以使其表示空字符序列。
- `String(String original)`： 初始化一个新创建的 `String` 对象，使其表示一个与参数相同的字符序列；换句话说，新创建的字符串是该参数字符串的副本。
- `public String(char[] value)`：通过当前参数中的字符数组来构造新的String。
- `public String(char[] value,int offset, int count)`：通过字符数组的一部分来构造新的String。
- `public String(byte[] bytes)`：通过使用平台的默认字符集解码当前参数中的字节数组来构造新的String。
- `public String(byte[] bytes,String charsetName)`：通过使用指定的字符集解码当前参数中的字节数组来构造新的String。

代码示例：

```
//字符串常量对象，推荐
String str = "hello";

// 无参构造，不推荐
String str1 = new String();

//创建"hello"字符串常量的副本，不推荐
String str2 = new String("hello");

//通过字符数组构造
char chars[] = {'a', 'b', 'c','d','e'};
String str3 = new String(chars);
String str4 = new String(chars,0,3);

// 通过字节数组构造
byte bytes[] = {97, 98, 99 };
String str5 = new String(bytes);
String str6 = new String(bytes,"GBK");
```

2. **使用"+"**

任意数据类型与"字符串"进行拼接，结果都是字符串

```
    public static void main(String[] args) {
        int num = 123456;
        String s = num + "";
        System.out.println(s);

        Student stu = new Student();
        String s2 = stu + "";//自动调用对象的toString()，然后与""进行拼接
        System.out.println(s2);
    }
```

9.1.3 字符串的常用方法

1、系列1

(1) boolean isEmpty()：字符串是否为空

(2) int length()：返回字符串的长度

(3) String concat(xx)：拼接，等价于+

(4) boolean equals(Object obj)：比较字符串是否相等，区分大小写

(5) boolean equalsIgnoreCase(Object obj)：比较字符串是否相等，不区分大小写

(6) int compareTo(String other)：比较字符串大小，区分大小写，按照Unicode编码值比较大小

(7) int compareToIgnoreCase(String other)：比较字符串大小，不区分大小写

(8) String toLowerCase()：将字符串中大写字母转为小写

(9) String toUpperCase()：将字符串中小写字母转为大写

(10) String trim()：去掉字符串前后空白符

```
    @Test
    public void test01(){
        //将用户输入的单词全部转为小写，如果用户没有输入单词，重新输入
        Scanner input = new Scanner(System.in);
        String word;
        while(true){
            System.out.print("请输入单词：");
            word = input.nextLine();
            if(word.trim().length()!=0){
                word = word.toLowerCase();
                break;
            }
        }
        System.out.println(word);
    }

    @Test
    public void test02(){
        //随机生成验证码，验证码由0-9，A-Z,a-z的字符组成
        char[] array = new char[26*2+10];
        for (int i = 0; i < 10; i++) {
```

```
            array[i] = (char)('0' + i);
        }
        for (int i = 10,j=0; i < 10+26; i++,j++) {
            array[i] = (char)('A' + j);
        }
        for (int i = 10+26,j=0; i < array.length; i++,j++) {
            array[i] = (char)('a' + j);
        }
        String code = "";
        Random rand = new Random();
        for (int i = 0; i < 4; i++) {
            code += array[rand.nextInt(array.length)];
        }
        System.out.println("验证码：" + code);
        //将用户输入的单词全部转为小写，如果用户没有输入单词，重新输入
        Scanner input = new Scanner(System.in);
        System.out.print("请输入验证码：");
        String inputCode = input.nextLine();

        if(!code.equalsIgnoreCase(inputCode)){
            System.out.println("验证码输入不正确");
        }
    }
```

2、系列2：查找

（11）boolean contains(xx)：是否包含xx

（12）int indexOf(xx)：从前往后找当前字符串中xx，即如果有返回第一次出现的下标，要是没有返回-1

（13）int lastIndexOf(xx)：从后往前找当前字符串中xx，即如果有返回最后一次出现的下标，要是没有返回-1

```
    @Test
    public void test01(){
        String str = "尚硅谷是一家靠谱的培训机构，尚硅谷可以说是IT培训的小清华，JavaEE是尚硅谷的当家学科，尚硅谷的大数据培训是行业独角兽。尚硅谷的前端和UI专业一样独领风骚。";
        System.out.println("是否包含清华：" + str.contains("清华"));
        System.out.println("培训出现的第一次下标：" + str.indexOf("培训"));
        System.out.println("培训出现的最后一次下标：" + str.lastIndexOf("培训"));
    }
```

3、系列3：字符串截取

（14）String substring(int beginIndex)：返回一个新的字符串，它是此字符串的从beginIndex开始截取到最后的一个子字符串。

（15）String substring(int beginIndex, int endIndex)：返回一个新字符串，它是此字符串从beginIndex开始截取到endIndex(不包含)的一个子字符串。

```
    @Test
    public void test01(){
        String str = "helloworldjavaatguigu";
        String sub1 = str.substring(5);
        String sub2 = str.substring(5,10);
        System.out.println(sub1);
```

```
        System.out.println(sub2);
    }

    @Test
    public void test02(){
        String fileName = "快速学习Java的秘诀.dat";
        //截取文件名
        System.out.println("文件名: " +
fileName.substring(0,fileName.lastIndexOf(".")));
        //截取后缀名
        System.out.println("后缀名: " +
fileName.substring(fileName.lastIndexOf(".")));
    }
```

4、系列4：和字符相关

（16）char charAt(index)：返回[index]位置的字符

（17）char[] toCharArray()： 将此字符串转换为一个新的字符数组返回

（18）String(char[] value)：返回指定数组中表示该字符序列的 String。

（19）String(char[] value, int offset, int count)：返回指定数组中表示该字符序列的 String。

（20）static String copyValueOf(char[] data)： 返回指定数组中表示该字符序列的 String

（21）static String copyValueOf(char[] data, int offset, int count)：返回指定数组中表示该字符序列的 String

（22）static String valueOf(char[] data, int offset, int count)： 返回指定数组中表示该字符序列的 String

（23）static String valueOf(char[] data) ：返回指定数组中表示该字符序列的 String

```
    @Test
    public void test01(){
        //将字符串中的字符按照大小顺序排列
        String str = "helloworldjavaatguigu";
        char[] array = str.toCharArray();
        Arrays.sort(array);
        str = new String(array);
        System.out.println(str);
    }

    @Test
    public void test02(){
        //将首字母转为大写
        String str = "jack";
        str = Character.toUpperCase(str.charAt(0))+str.substring(1);
        System.out.println(str);
    }
```

5、系列5：编码与解码

（24）byte[] getBytes()：编码，把字符串变为字节数组，按照平台默认的字符编码进行编码

byte[] getBytes(字符编码方式)：按照指定的编码方式进行编码

（25）new String(byte[]) 或 new String(byte[], int, int)：解码，按照平台默认的字符编码进行解码

new String(byte[]，字符编码方式) 或 new String(byte[], int, int，字符编码方式)：解码，按照指定的编码方式进行解码

```
    /*
     * GBK，UTF-8，ISO8859-1所有的字符编码都向下兼容ASCII码
     */
    public static void main(String[] args) throws Exception {
        String str = "中国";
        System.out.println(str.getBytes("ISO8859-1").length);// 2
        // ISO8859-1把所有的字符都当做一个byte处理，处理不了多个字节
        System.out.println(str.getBytes("GBK").length);// 4 每一个中文都是对应2个字节
        System.out.println(str.getBytes("UTF-8").length);// 6 常规的中文都是3个字节

        /*
         * 不乱码：（1）保证编码与解码的字符集名称一样（2）不缺字节
         */
        System.out.println(new String(str.getBytes("ISO8859-1"), "ISO8859-
1"));// 乱码
        System.out.println(new String(str.getBytes("GBK"), "GBK"));// 中国
        System.out.println(new String(str.getBytes("UTF-8"), "UTF-8"));// 中国
    }
```

字符编码发展

ASCII码

计算机一开始发明的时候是用来解决数字计算的问题，后来人们发现，计算机还可以做更多的事，例如文本处理。但由于计算机只识"数"，因此人们必须告诉计算机哪个数字来代表哪个特定字符，例如65代表字母'A'，66代表字母'B'，以此类推。但是计算机之间字符-数字的对应关系必须得一致，否则就会造成同一段数字在不同计算机上显示出来的字符不一样。因此美国国家标准协会ANSI制定了一个标准，规定了常用字符的集合以及每个字符对应的编号，这就是ASCII字符集（Character Set），也称ASCII码。

那时候的字符编解码系统非常简单，就是简单的查表过程。其中：

- 0～31及127(共33个)是控制字符或通信专用字符（其余为可显示字符），如控制符：LF（换行）、CR（回车）、FF（换页）、DEL（删除）、BS（退格）
- 32～126(共95个)是字符(32是空格），其中48～57为0到9十个阿拉伯数字。
- 65～90为26个大写英文字母，97～122号为26个小写英文字母，其余为一些标点符号、运算符号等。

OEM字符集的衍生

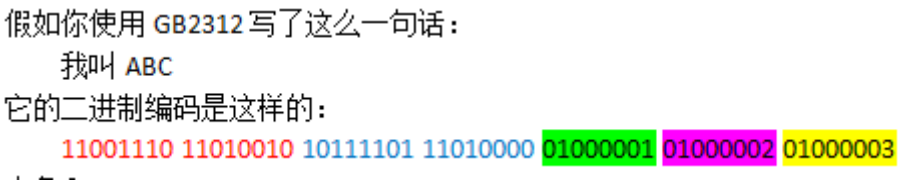
当计算机开始发展起来的时候，人们逐渐发现，ASCII字符集里那可怜的128个字符已经不能再满足他们的需求了。人们就在想，一个字节能够表示的数字（编号）有256个，而ASCII字符只用到了0x00~0x7F，也就是占用了前128个，后面128个数字不用白不用，因此很多人打起了后面这128个数字的主意。可是问题在于，很多人同时有这样的想法，但是大家对于0x80-0xFF这后面的128个数字分别对应什么样的字符，却有各自的想法。这就导致了当时销往世界各地的机器上出现了大量各式各样的OEM字符集。不同的OEM字符集导致人们无法跨机器交流各种文档。例如职员甲发了一封简历résumés给职员乙，结果职员乙看到的却是r?sum?s，因为é字符在职员甲机器上的OEM字符集中对应的字节是0x82，而在职员乙的机器上，由于使用的OEM字符集不同，对0x82字节解码后得到的字符却是?。

多字节字符集（MBCS）和中文字符集

上面我们提到的字符集都是基于单字节编码，也就是说，一个字节翻译成一个字符。这对于拉丁语系国家来说可能没有什么问题，因为他们通过扩展第8个比特，就可以得到256个字符了，足够用了。但是对于亚洲国家来说，256个字符是远远不够用的。因此这些国家的人为了用上电脑，又要保持和ASCII字符集的兼容，就发明了多字节编码方式，相应的字符集就称为多字节字符集（Muilti-Bytes Charecter

Set）。例如中国使用的就是双字节字符集编码。

例如目前最常用的中文字符集GB2312，涵盖了所有简体字符以及一部分其他字符；GBK（K代表扩展的意思）则在GB2312的基础上加入了对繁体字符等其他非简体字符。这两个字符集的字符都是使用1-2个字节来表示。Windows系统采用936代码页来实现对GBK字符集的编解码。在解析字节流的时候，如果遇到字节的最高位是0的话，那么就使用936代码页中的第1张码表进行解码，这就和单字节字符集的编解码方式一致了。如果遇到字节的最高位是1的话，那么就表示需要两个字节值才能对应一个字符。



ANSI标准、国家标准、ISO标准

不同ASCII衍生字符集的出现，让文档交流变得非常困难，因此各种组织都陆续进行了标准化流程。例如美国ANSI组织制定了ANSI标准字符编码（注意，我们现在通常说到ANSI编码，通常指的是平台的默认编码，例如英文操作系统中是ISO-8859-1，中文系统是GBK），ISO组织制定的各种ISO标准字符编码，还有各国也会制定一些国家标准字符集，例如中国的GBK，GB2312和GB18030。

操作系统在发布的时候，通常会往机器里预装这些标准的字符集还有平台专用的字符集，这样只要你的文档是使用标准字符集编写的，通用性就比较高了。例如你用GB2312字符集编写的文档，在中国大陆内的任何机器上都能正确显示。同时，我们也可以在一台机器上阅读多个国家不同语言的文档了，前提是本机必须安装该文档使用的字符集。

Unicode的出现

虽然通过使用不同字符集，我们可以在一台机器上查阅不同语言的文档，但是我们仍然**无法解决一个问题：如果一份文档中含有不同国家的不同语言的字符，那么无法在一份文档中显示所有字符。**为了解决这个问题，我们需要一个全人类达成共识的巨大的字符集，这就是Unicode字符集。

Unicode字符集涵盖了目前人类使用的所有字符，并为每个字符进行统一编号，分配唯一的字符码（Code Point）。Unicode字符集将所有字符按照使用上的频繁度划分为17个层面（Plane），每个层面上有216=65536个字符码空间。其中第0个层面BMP，基本涵盖了当今世界用到的所有字符。其他的层面要么是用来表示一些远古时期的文字，要么是留作扩展。我们平常用到的Unicode字符，一般都是位于BMP层面上的。目前Unicode字符集中尚有大量字符空间未使用。

Unicode同样也不完美，这里就有三个的问题，一个是，我们已经知道，英文字母只用一个字节表示就够了，第二个问题是如何才能区别Unicode和ASCII？计算机怎么知道两个字节表示一个符号，而不是分别表示两个符号呢？第三个，如果和GBK等双字节编码方式一样，用最高位是1或0表示两个字节和一个字节，就少了很多值无法用于表示字符，不够表示所有字符。Unicode在很长一段时间内无法推广，直到互联网的出现，为解决Unicode如何在网络上传输的问题，于是面向传输的众多 UTF（UCS Transfer Format）标准出现了，顾名思义，UTF-8就是每次8个位传输数据，而UTF-16就是每次16个位。UTF-8就是在互联网上使用最广的一种Unicode的实现方式，这是为传输而设计的编码，并使编码无国界，这样就可以显示全世界上所有文化的字符了。

UTF-8最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号。从unicode到uft-8并不是直接的对应，而是要过一些算法和规则来转换（**即Uncidoe字符集≠UTF-8编码方式**）。

并不是直接的对应，而是要过一些算法和规则来转换（**即Uncidoe字符集≠UTF-8编码方式**）。

Unicode符号范围 | UTF-8编码方式

(十六进制) | （二进制）

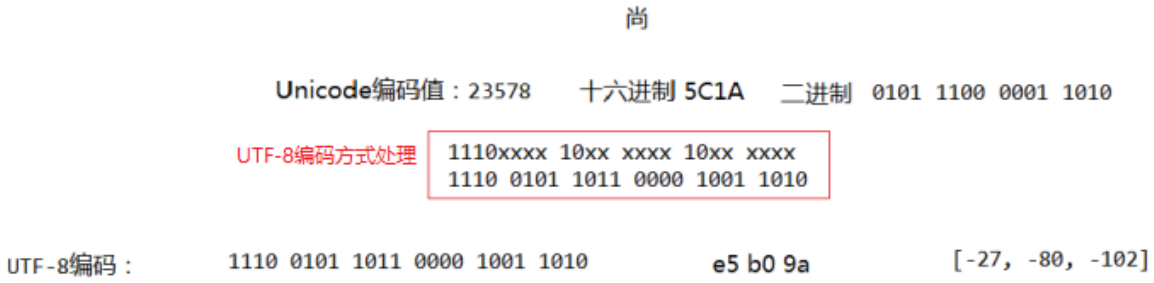
———————————————————————

0000 0000-0000 007F | 0xxxxxxx（兼容原来的ASCII）

0000 0080-0000 07FF | 110xxxxx 10xxxxxx

0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx

0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx



因此，Unicode只是定义了一个庞大的、全球通用的字符集，并为每个字符规定了唯一确定的编号，具体存储成什么样的字节流，取决于字符编码方案。推荐的Unicode编码是UTF-16和UTF-8。

早期字符编码、字符集和代码页等概念都是表达同一个意思。例如GB2312字符集、GB2312编码，936代码页，实际上说的是同个东西。

但是对于Unicode则不同，Unicode字符集只是定义了字符的集合和唯一编号，Unicode编码，则是对UTF-8、UCS-2/UTF-16等具体编码方案的统称而已，并不是具体的编码方案。所以当需要用到字符编码的时候，你可以写gb2312，codepage936，utf-8，utf-16，但请不要写Unicode。

6、系列6：开头与结尾

（26）boolean startsWith(xx)：是否以xx开头

（27）boolean endsWith(xx)：是否以xx结尾

```
    @Test
    public void test2(){
        String name = "张三";
        System.out.println(name.startsWith("张"));
    }

    @Test
    public void test(){
        String file = "Hello.txt";
        if(file.endsWith(".java")){
            System.out.println("Java源文件");
        }else if(file.endsWith(".class")){
            System.out.println("Java字节码文件");
        }else{
            System.out.println("其他文件");
        }
    }
```

7、系列7：正则匹配（不讲）

（28）boolean matches(正则表达式)：判断当前字符串是否匹配某个正则表达式

常用正则表达式：(暂)

字符类

`[abc]`：`a`、`b` 或 `c`（简单类）

`[^abc]`：任何字符，除了 `a`、`b` 或 `c`（否定）

`[a-zA-Z]`：`a` 到 `z` 或 `A` 到 `Z`，两头的字母包括在内（范围）

预定义字符类

`.`：任何字符（与行结束符可能匹配也可能不匹配）

`\d`：数字：`[0-9]`

`\D`：非数字：`[^0-9]`

`\s`：空白字符：`[ \t\n\x0B\f\r]`

`\S`：非空白字符：`[^\s]`

`\w`：单词字符：`[a-zA-Z_0-9]`

`\W`：非单词字符：`[^\w]`

POSIX 字符类（仅 US-ASCII）

`\p{Lower}` 小写字母字符：[a-z]

`\p{Upper}` 大写字母字符：[A-Z]

`\p{ASCII}` 所有 ASCII：[\x00-\x7F]

`\p{Alpha}` 字母字符：[\p{Lower}\p{Upper}]

`\p{Digit}` 十进制数字：[0-9]

`\p{Alnum}` 字母数字字符：[\p{Alpha}\p{Digit}]

`\p{Punct}` 标点符号：!"#$%&'()*+,-./:;<=>?@[]^_`{|}~

`\p{Blank}` 空格或制表符：[\t]

边界匹配器

`^`：行的开头

`$`：行的结尾

Greedy 数量词

X`?`：*X*，一次或一次也没有

X`*`：*X*，零次或多次

X`+`：*X*，一次或多次

X`{`*n*`}`：*X*，恰好 *n* 次

X`{`*n*`,}`：*X*，至少 *n* 次

X`{`*n*`,`*m*`}`：*X*，至少 *n* 次，但是不超过 *m* 次

Logical 运算符

XY：*X* 后跟 *Y*

X`|`*Y*：*X* 或 *Y*

`(`*X*`)`：X，作为捕获组

特殊构造（非捕获）

(?:X) X，作为非捕获组

(?=X) X，通过零宽度的正 lookahead

(?!X) X，通过零宽度的负 lookahead

(?<=X) X，通过零宽度的正 lookbehind

(?<!X) X，通过零宽度的负 lookbehind

(?>X) X，作为独立的非捕获组

```
    @Test
    public void test1(){
        //简单判断是否全部是数字，这个数字可以是1~n位
        String str = "12a345";

        //正则不是Java的语法，它是独立与Java的规则
        //在正则中\是表示转义，
        //同时在Java中\也是转义
        boolean flag = str.matches("\\d+");
        。System.out.println(flag);
    }

    @Test
    public void test2(){
        String str = "123456789";

        //判断它是否全部由数字组成，并且第1位不能是0，长度为9位
        //第一位不能是0，那么数字[1-9]
        //接下来8位的数字，那么[0-9]{8}+
        boolean flag = str.matches("[1-9][0-9]{8}+");
        System.out.println(flag);
    }

    @Test
    public void test03(){
        //密码要求：必须有大写字母，小写字母，数字组成，6位
        System.out.println("Cly892".matches("^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])
[A-Za-z0-9]{6}$"));//true
        System.out.println("1A2c45".matches("^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])
[A-Za-z0-9]{6}$"));//true
        System.out.println("Clyyyy".matches("^(?=.*[A-Z])(?=.*[0-9])[A-Za-z0-9]
{6}$"));//false
    }
```

1.验证用户名和密码，要求第一个字必须为字母，一共6~16位字母数字下划线组成：（^[a-zA-Z]\w{5,15}$)

2.验证电话号码：xxx/xxxx-xxxxxxx/xxxxxxxx：（^(\d{3,4}-)\d{7,8}$)

3.验证手机号码：(^(13[0-9]|14[5|7]|15[0|1|2|3|5|6|7|8|9]|18[0|1|2|3|5|6|7|8|9])\d{8}$)

4.验证身份证号：（^\d{15}$)|(^\d{18}$)|(^\d{17}(\d|X|x)$)

5.验证Email地址：(^\w+([-+.]\w+)@\w+([-.]\w+).\w+([-.]\w+)*$)

6.只能输入由数字和26个英文字母组成的字符串：(^[A-Za-z0-9]+$)

7.整数或者小数：(^[0-9]+(.[0-9]+){0,1}$)

8.中文字符的正则表达式：([\u4e00-\u9fa5])

9.金额校验(非零开头的最多带两位小数的数字)：(^([1-9][0-9]*)+(.[0-9]{1,2})?$)

10.IPV4地址：(((\d{1,2})|(1\d{1,2})|(2[0-4]\d)|(25[0-5]))\.){3}((\d{1,2})|(1\d{1,2})|(2[0-4]\d)|(25[0-5]))

8、系列8：替换

（29）String replace(xx,xx)：不支持正则

（30）String replaceFirst(正则，value)：替换第一个匹配部分

（31）String repalceAll(正则， value)：替换所有匹配部分

```
    @Test
    public void test4(){
        String str = "hello244world.java;887";
        //把其中的非字母去掉
        str = str.replaceAll("[^a-zA-Z]", "");
        System.out.println(str);
    }
```

9、系列9：拆分

（32）String[] split(正则)：按照某种规则进行拆分

```
    @Test
    public void test4(){
        String str = "张三.23|李四.24|王五.25";
        //|在正则中是有特殊意义，我这里要把它当做普通的|
        String[] all = str.split("\\|");

        //转成一个一个学生对象
        Student[] students = new Student[all.length];
        for (int i = 0; i < students.length; i++) {
            //.在正则中是特殊意义，我这里想要表示普通的.
            String[] strings = all[i].split("\\.");//张三，  23
            String name = strings[0];
            int age = Integer.parseInt(strings[1]);
            students[i] = new Student(name,age);
        }

        for (int i = 0; i < students.length; i++) {
            System.out.println(students[i]);
        }

    }

    @Test
    public void test3(){
        String str = "1Hello2World3java4atguigu5";
        str = str.replaceAll("^\\d|\\d$", "");
        String[] all = str.split("\\d");
        for (int i = 0; i < all.length; i++) {
            System.out.println(all[i]);
```

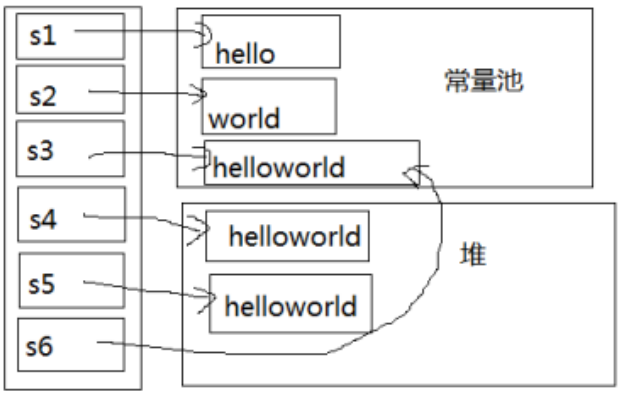
```
        }
    }

    @Test
    public void test2(){
        String str = "1Hello2World3java4atguigu";
        str = str.replaceFirst("\\d", "");
        System.out.println(str);
        String[] all = str.split("\\d");
        for (int i = 0; i < all.length; i++) {
            System.out.println(all[i]);
        }
    }


    @Test
    public void test1(){
        String str = "Hello World java atguigu";
        String[] all = str.split(" ");
        for (int i = 0; i < all.length; i++) {
            System.out.println(all[i]);
        }
    }
```

9.1.4 字符串对象的内存分析

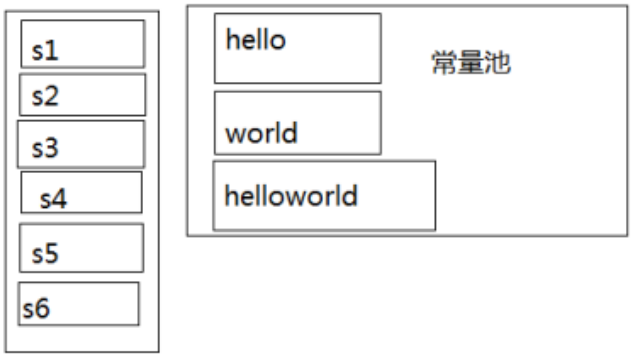
```
    @Test
    public void test1(){
        String s1 = "hello";
        String s2 = "hello";
        String s3 = new String("hello");

    }
    @Test
    public void test02(){
        String s1 = "hello";
        String s2 = "world";
        String s3 = "helloworld";

        String s4 = s1 + "world";//s4字符串内容也helloworld，s1是变量，"world"常量，变量 + 常量的结果在堆中
        String s5 = s1 + s2;//s5字符串内容也helloworld，s1和s2都是变量，变量 + 变量的结果在堆中
        String s6 = "hello" + "world";//常量+常量，编译期经过优化，跟s3完全相同的情况。

        System.out.println(s3 == s4);//false
        System.out.println(s3 == s5);//false
        System.out.println(s3 == s6);//true
    }

    @Test
    public void test03(){
        final String s1 = "hello";
        final String s2 = "world";
        String s3 = "helloworld";
```

```
        String s4 = s1 + "world";//s4字符串内容也helloworld，s1是常量，"world"常量，常量+ 常量 结果在常量池中
        String s5 = s1 + s2;//s5字符串内容也helloworld，s1和s2都是常量，常量+ 常量 结果在常量池中
        String s6 = "hello" + "world";//常量+ 常量 结果在常量池中，因为编译期间就可以确定结果

        System.out.println(s3 == s4);//true
        System.out.println(s3 == s5);//true
        System.out.println(s3 == s6);//true
    }

    @Test
    public void test04(){
        String s1 = "hello";
        String s2 = "world";
        String s3 = "helloworld";

        String s4 = (s1 + "world").intern();//如果常量池已经有“helloworld”，直接返回，否则把拼接结果的引用放到常量池中
        String s5 = (s1 + s2).intern();

        System.out.println(s3 == s4);//true
        System.out.println(s3 == s5);//true
    }
```

```
String s1 = "hello";
String s2 = "world";
String s3 = "hello" + "world";
String s4 = s1 + "world";
String s5 = s1 + s2;
String s6 = (s1 + s2).intern();
System.out.println(s3==s4);//false
System.out.println(s3==s5);//false
System.out.println(s4==s5);//false
System.out.println(s3==s6);//true
```




```
final String s1 = "hello";
final String s2 = "world";
String s3 = "hello" + "world";
String s4 = s1 + "world";
String s5 = s1 + s2;
String s6 = (s1 + s2).intern();
System.out.println(s3==s4); //true
System.out.println(s3==s5);//true
System.out.println(s4==s5); //true
System.out.println(s3==s6); //true
```



结论：

- 常量+常量：结果是常量池
- 常量与变量 或 变量与变量：结果是堆
- intern方法有jvm版本的区别，这里不再深入分析，jdk8中执行原理是如果字符串常量池有内容相同的字符串则直接返回，否则把堆中创建的字符串引用放入字符串常量池，返回此引用，总之所有版本都是通过字符串常量池返回的内容。

9.1.5 字符串相关面试题

1、字符串特点相关

1. **面试题：字符串的length和数组的length有什么不同?**

   字符串的length()，数组的length属性

2. **字符串对象不可变**

```
class TEXT{
    public int num;
    public String str;

    public TEXT(int num, String str){
        this.num = num;
        this.str = str;
    }
}
public class Class4 {
    //tIn是传对象的地址，修改形参的属性，会影响实参
    //intIn是传数据，基本数据类型的形参修改和实参无关
    //Integer和String对象不可变
    public static void f1(TEXT tIn, int intIn, Integer integerIn, String
strIn){
        tIn.num =200;
        tIn.str = "bcd";//形参和实参指向的是同一个TEXT的对象，修改了属性，就相当于修改
实参对象的属性
        intIn = 200;//基本数据类型的形参是实参的“副本”，无论怎么修改和实参都没关系
        integerIn = 200;//Integer对象和String对象一样都是不可变，一旦修改都是新对
象，和实参无关
        strIn = "bcd";
    }
    public static void main(String[] args) {
        TEXT tIn = new TEXT(100, "abc");//tIn.num = 100, tIn.str="abc"
        int intIn = 100;
        Integer integerIn = 100;
        String strIn = "abc";

        f1(tIn,intIn,integerIn,strIn);

        System.out.println(tIn.num + tIn.str + intIn + integerIn + strIn);
        //200 + bcd + 100 + 100 + abc
    }
}
```

12. 下列代码执行的结果是（ ）

```
public class Example {
    public static void stringReplace (String text) {
        text = text.replace ( 'j' , 'i' );
    }

    public static void bufferReplace (StringBuffer text) {
        text.append ( "C" );
        text=new StringBuffer( "Hello" );
        text.append( "World!" );
    }

    public static void main (String args[]) {
        String textString = new String ( "java" );
        StringBuffer textBuffer = new StringBuffer ( "java" );

        stringReplace (textString);
        bufferReplace (textBuffer);

        System.out.println (textString + textBuffer);
    }
}
```

A. iavaHelloWorld
B. javajavaC
C. javaHelloWorld
D. iavajavaC

3. **字符串对象个数**

15. String str = new String("abc");这行代码创建了几个对象（ ）
A、1 B、2 C、3 D、4

2. 这条语句 String s="a"+"b"+"c"+"d"一共创建了多少个对象（ ）

A：1个

B：4个

C：6个

D：7个

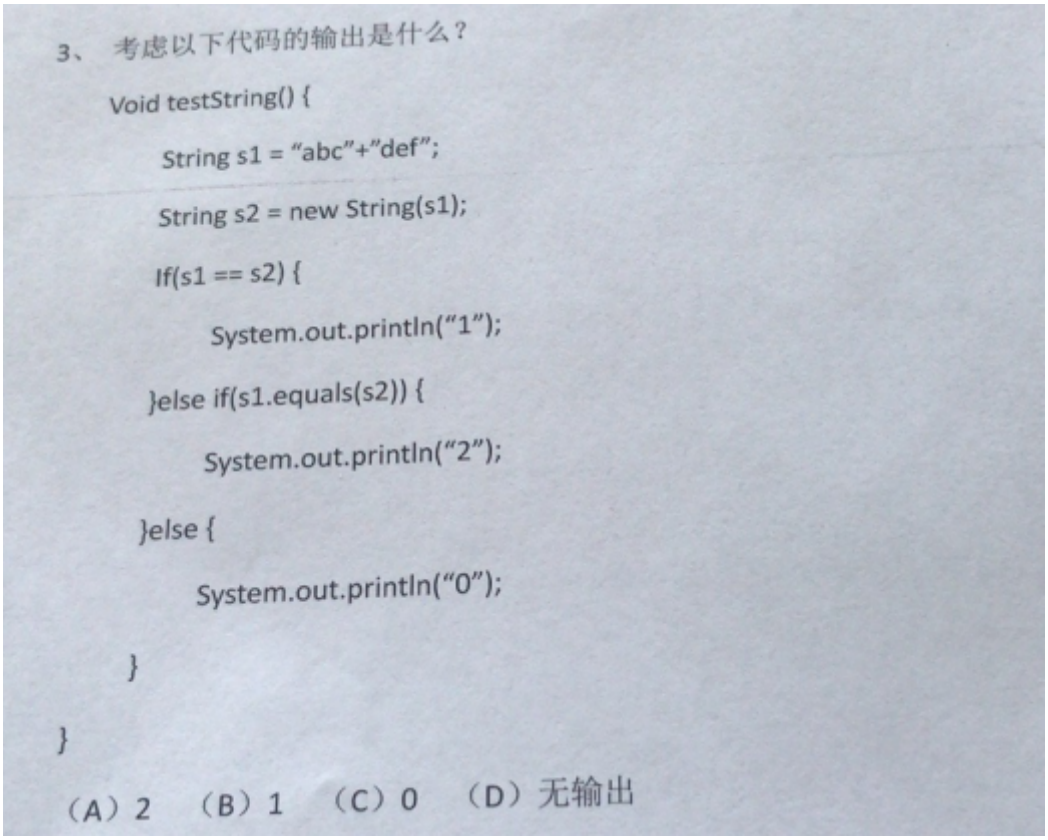
4. **字符串对象比较**

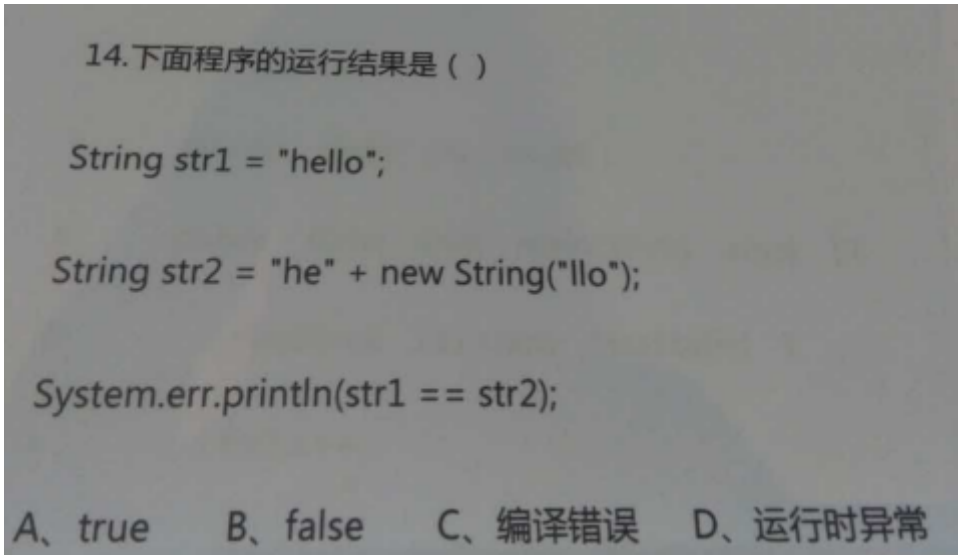
16. 阅读下面代码：

```
String str1 = "abc";
String str2 = new String("abc");
boolean b1 = str1 == str2;
boolean b2 = str1.eqauls(str2);
```

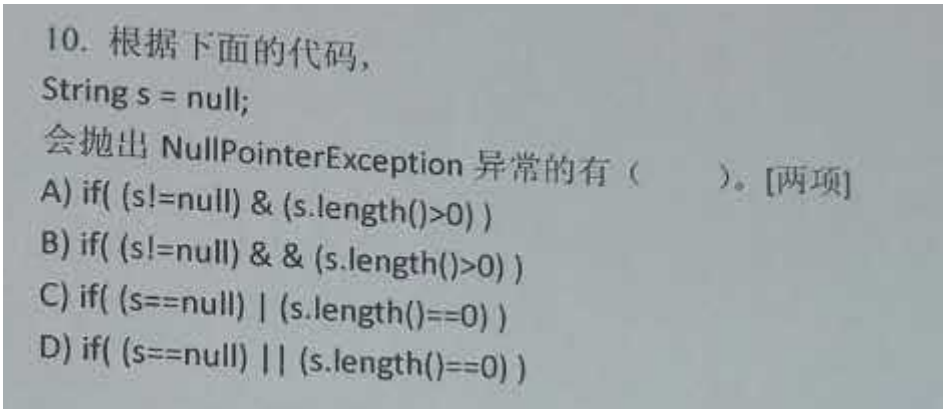
判断正确的是（ ）

A、b1 的值为 true，b2 的值为 true
B、b1 的值为 false，b2 的值为 true
C、b1 的值为 true，b2 的值为 false
D、b1 的值为 false，b2 的值为 false


3、 考虑以下代码的输出是什么？
Void testString() {
String s1 = “abc”+”def”;
String s2 = new String(s1);
If(s1 == s2) {
System.out.println(“1”);
}else if(s1.equals(s2)) {
System.out.println(“2”);
}else {
System.out.println(“0”);
}
}
（A）2 （B）1 （C）0 （D）无输出


14.下面程序的运行结果是（ ）
String str1 = "hello";
String str2 = "he" + new String("llo");
System.err.println(str1 == str2);
A、true B、false C、编译错误 D、运行时异常

5. **空字符串**

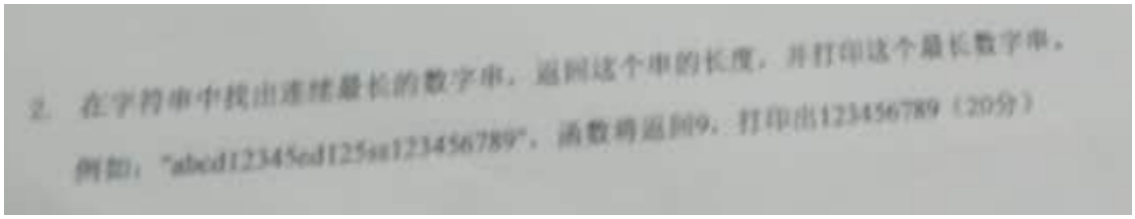

10. 根据下面的代码，
String s = null;
会抛出 NullPointerException 异常的有（ ）。[两项]
A) if((s!=null) & (s.length()>0))
B) if((s!=null) & & (s.length()>0))
C) if((s==null) | (s.length()==0))
D) if((s==null) || (s.length()==0))

2、字符串算法相关

1. 编程题

在字符串中找出连续最长数字串，返回这个串的长度，并打印这个最长数字串。

例如：abcd12345cd125se123456789，返回9，打印出123456789



```
public class TestExer1 {
    public static void main(String[] args) {
        String str = "abcd12345cd125se123456789";

        //去掉最前和最后的字母
        str =   str.replaceAll("^[a-zA-Z]+", "");

        //[a-zA-Z]：表示字母范围
        //+：一次或多次
        String[] strings = str.split("[a-zA-Z]+");

        String max = "";
        for (String string : strings) {
            if(string.length() > max.length()) {
                max = string;
            }
        }
        System.out.println("最长的数字串：" + max + "，它的长度为：" +
max.length());
    }
}
```

2. 编程题

不能使用trim()，实现去除字符串两端的空格。

```
    public static void main(String[] args) {
        String str ="    he   llo   ";
        System.out.println(myTrim(str));
        System.out.println(myTrim2(str));
        System.out.println(myTrim3(str));
    }

    public static String myTrim3(String str){
        //利用正则表达式
        //^表示开头    \s表示  空白符    *表示0次或多次      |表示或者     $表示结尾
        return str.replaceAll("(^\\s*)|(\\s*$)", "");
    }

    public static String myTrim2(String str){
        while(str.startsWith(" ")){
```

```
            str = str.replaceFirst(" ", "");
        }
        while(str.endsWith(" ")){
            str = str.substring(0, str.length()-1);
        }
        return str;
    }

    public static String myTrim(String str){
        char[] array = str.toCharArray();
        int start =0;
        for(int i=0;i<array.length;i++){
            if(array[i] == ' '){
                start++;
            }else{
                break;
            }
        }
        int end = array.length-1;
        for(int i=end;i>=0;i--){
            if(array[i] == ' '){
                end--;
            }else{
                break;
            }
        }

        String result = str.substring(start,end+1);

        return result;
    }
```

3. 编程题

将字符串中指定部分进行反转。比如将“abcdefgho”反转为”abfedcgho”

```
    public static void main(String[] args) {
        String str ="abcdefgho";
        System.out.println(str);
        System.out.println(reverse(str,2,5));

    }

    //从第start个字符，到第end个字符
    public static String reverse(String str,int start,int end){
        char[] array = str.toCharArray();
        for(int i = start,j=end;i< j;i++,j--){
            char temp =array[i];
            array[i]=array[j];
            array[j]=temp;
        }
        String s = new String(array);
        return s;
    }

    //从第start个字符，到第end个字符
    public static String reverse(String str,int start,int end){
```

```
        String left = str.substring(0,start);
        String middle = str.substring(start,end+1);
        String left = str.substring(end+1);
        return left+new StringBuilder(middle).reverse()+right;
    }
```

4. 编程题

获取一个字符串在另一个字符串中出现的次数。

比如：获取"ab"在 “abababkkcadkabkebfkabkskab”中出现的次数

```
    public static void main(String[] args) {
        String str1="ab";
        String str2="abababkkcadkabkebfkabkskab";
        System.out.println(count(str1,str2));
    }

    public static int count(String str1,String str2){
        int count =0;
        do{
            int index = str2.indexOf(str1);
            if(index !=-1){
                count++;
                str2 = str2.substring(index + str1.length());
            }else{
                break;
            }

        }while(true);
        return count;
    }
```

5. 编程题

获取两个字符串中最大相同子串。

比如：str1 = "abcwerthelloyuiodef“;str2 = "cvhellobnm"

提示：将短的那个串进行长度依次递减的子串与较长的串比较。

```
    public static void main(String[] args) {
        String str=findMaxSubString("abcwerthelloyuiodef","cvhellobnm");
        System.out.println(str);
    }

    //提示：将短的那个串进行长度依次递减的子串与较长的串比较。
    public static String findMaxSubString(String str1,String str2){
        String result="";

        String mixStr = str1.length()<str2.length()?str1:str2;
        String maxStr = str1.length()>str2.length()?str1:str2;

        //外循环控制从左到右的下标，内循环控制从右到左的下标
        for(int i=0;i<mixStr.length();i++){
            for(int j=mixStr.length();j>=i;j--){
```

```
                String str=mixStr.substring(i, j);
                if(maxStr.contains(str)){
                    //找出最大相同子串
                    if(result.length()<str.length()){
                        result = str;
                    }
                }
            }
        }
        return result;
    }
```

6. 编程题

编写代码完成如下功能

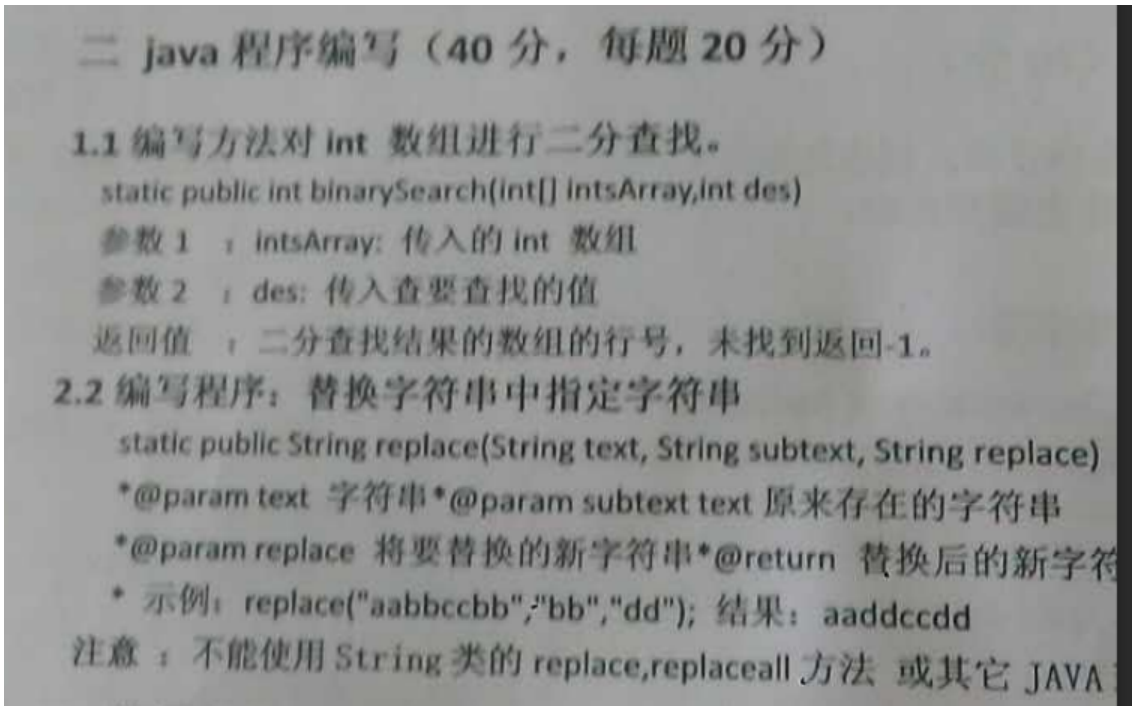
public static String replace(String text, String target, String replace){

....

}

示例：replace(“aabbccbb”, “bb”, “dd”); 结果：aaddccdd

注意：不能使用String及StringBuffer等类的replace等现成的替换API方法。

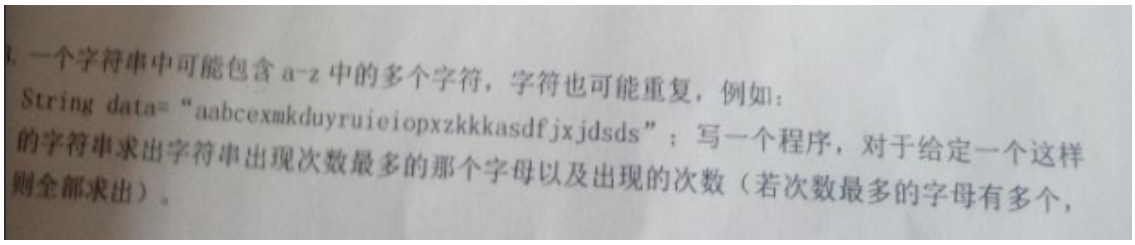



```
    public static void main(String[] args) {
        System.out.println(replace("aabbcbcbb","bb","dd"));
    }
    public static String replace(String text, String target, String replace)
{
        while(true) {
            int index = text.indexOf(target);
            if(index!=-1) {
                text = text.substring(0,index) + replace +
text.substring(index+target.length());
            }else {
                break;
            }
        }
```

```
        return text;
    }
```

7. 编程题

1个字符串中可能包含a-z中的多个字符，字符也可能重复，例如：String data = “aabcexmkduyruieiopxzkkkasdfjxjdsds”;写一个程序，对于给定一个这样的字符串求出字符串出现次数最多的那个字母以及出现的次数（若次数最多的字母有多个，则全部求出）




```
    public static void main(String[] args) {
        String str = "aabbyolhljlhlxxmnbwyteuhfhjloiqqbhrg";

        //统计每个字母的次数
        int[] counts = new int[26];
        char[] letters = str.toCharArray();
        for (int i = 0; i < letters.length; i++) {
            counts[letters[i]-97]++;
        }

        //找出最多次数值
        int max = counts[0];
        for (int i = 1; i < counts.length; i++) {
            if(max < counts[i]) {
                max = counts[i];
            }
        }
        //找出所有最多次数字母
        for (int i = 0; i < counts.length; i++) {
            if(counts[i] == max) {
                System.out.println((char)(i+97));
            }
        }
    }
```

> 如果学习完集合之后，该题还可以使用Map集合写出不同的答案

8. 编程题

假设日期段用两个6位长度的正整数表示，例如：(201401，201406)用来表示2014年1月到2014年6月，求两个日期段的重叠月份数。例如：输入：时间段1：201401和201406，时间段2：201403和201409，输出：4

解释：重叠月份：3,4,5,6月共4个月

情形1：两个时间段都是同一年内的，实现代码如下：

```
    public static void main(String[] args) {
        String date1Start = "201401";
        String date1End = "201406";
```

```
        String date2Start = "201403";
        String date2End = "201409";

        int date1StartMonth = Integer.parseInt(date1Start.substring(4));
        int date1EndMonth = Integer.parseInt(date1End.substring(4));

        int date2StartMonth = Integer.parseInt(date2Start.substring(4));
        int date2EndMonth = Integer.parseInt(date2End.substring(4));

        int start = date1StartMonth >= date2StartMonth ? date1StartMonth :
date2StartMonth;
        int end = date1EndMonth <= date2EndMonth ? date1EndMonth :
date2EndMonth;
        System.out.println("重叠月份数："+(end-start+1));

        System.out.println("重叠的月份有：");
        for (int i = start; i <= end; i++) {
            System.out.println(i);
        }
    }
```

情形2：两个时间段可能不在同一年内的，实现代码如下：

```
    public static void main(String[] args) {
        String date1Start = "201401";
        String date1End = "201506";

        String date2Start = "201403";
        String date2End = "201505";

        String date1 = handleDate(date1Start,date1End);
        String date2 = handleDate(date2Start,date2End);
        System.out.println(date1);
        System.out.println(date2);

        String sameDate = findMaxSubString(date1,date2);

        System.out.println("重叠的月份数：" + sameDate.length()/6);
        if (!"".equals(sameDate)) {
            System.out.println("重叠的月份有：");
            while (sameDate.length() > 0) {
                String sameMonth = sameDate.substring(0, 6);
                System.out.println(sameMonth);
                sameDate = sameDate.substring(6);
            }

        }
    }

    public static String findMaxSubString(String str1,String str2){
        String result="";

        String mixStr = str1.length()<str2.length()?str1:str2;
        String maxStr = str1.length()>str2.length()?str1:str2;

        //外循环控制从左到右的下标，内循环控制从右到左的下标
        for(int i=0;i<mixStr.length();i++){
```

```
            for(int j=mixStr.length();j>=i;j--){
                String str=mixStr.substring(i, j);
                if(maxStr.contains(str)){
                    //找出最大相同子串
                    if(result.length()<str.length()){
                        result = str;
                    }
                }
            }
        }
        return result;
    }

    public static String handleDate(String dateStart, String dateEnd) {
        int dateStartYear = Integer.parseInt(dateStart.substring(0,4));
        int dateEndYear = Integer.parseInt(dateEnd.substring(0,4));
        int dateStartMonth = Integer.parseInt(dateStart.substring(4));
        int dateEndMonth = Integer.parseInt(dateEnd.substring(4));

        String date = "";
        if(dateStartYear == dateEndYear) {//一年之内
            for (int i = dateStartMonth; i <=dateEndMonth; i++) {
                if(i<10) {
                    date += dateStartYear+"0"+i;
                }else {
                    date += dateStartYear+""+i;
                }
            }
        }else {//跨年
            for (int i = dateStartMonth; i <=12; i++) {//date1StartYear起始年
                if(i<10) {
                    date += dateStartYear+"0"+i;
                }else {
                    date += dateStartYear+""+i;
                }
            }
            for (int i = dateStartYear+1; i < dateEndYear; i++) {//中间间隔年
                for (int j = 1; j <= 12; j++) {
                    if(j<10) {
                        date += i+"0"+j;
                    }else {
                        date += i+""+j;
                    }
                }
            }
            for (int i = 1; i <= dateEndMonth; i++) {//date1EndYear结束年
                if(i<10) {
                    date += dateEndYear+"0"+i;
                }else {
                    date += dateEndYear+""+i;
                }
            }
        }
        return date;
    }
}
```

9.2 StringBuilder&StringBuffer

9.2.1 与String区别

因为String对象是不可变对象，虽然可以共享常量对象，但是对于频繁字符串的修改和拼接操作，效率极低。因此，JDK又在java.lang包提供了可变字符序列StringBuilder和StringBuffer类型。

StringBuffer：老的，线程安全的（因为它的方法有synchronized修饰），效率低

StringBuilder：线程不安全的，效率高

9.2.2 常用API

常用的API，StringBuilder、StringBuffer的API是完全一致的

（1）StringBuffer append(xx)：拼接，追加

（2）StringBuffer insert(int index, xx)：在[index]位置插入xx

（3）StringBuffer delete(int start, int end)：删除[start,end)之间字符

StringBuffer deleteCharAt(int index)：删除[index]位置字符

（4）void setCharAt(int index, xx)：替换[index]位置字符

（5）StringBuffer reverse()：反转

（6）void setLength(int newLength)：2设置当前字符序列长度为newLength

（7）StringBuffer replace(int start, int end, String str)：替换[start,end)范围的字符序列为str

（8）int indexOf(String str)：在当前字符序列中查询str的第一次出现下标

int indexOf(String str, int fromIndex)：在当前字符序列[fromIndex,最后]中查询str的第一次出现下标

int lastIndexOf(String str)：在当前字符序列中查询str的最后一次出现下标

int lastIndexOf(String str, int fromIndex)：在当前字符序列[fromIndex,最后]中查询str的最后一次出现下标

（9）String substring(int start)：截取当前字符序列[start,最后]

（10）String substring(int start, int end)：截取当前字符序列[start,end)

（11）String toString()：返回此序列中数据的字符串表示形式

```
@Test
public void test6(){
    StringBuilder s = new StringBuilder("helloworld");
    s.setLength(30);
    System.out.println(s);
}
@Test
public void test5(){
    StringBuilder s = new StringBuilder("helloworld");
    s.setCharAt(2, 'a');
    System.out.println(s);
}
```

```
    @Test
    public void test4(){
        StringBuilder s = new StringBuilder("helloworld");
        s.reverse();
        System.out.println(s);
    }

    @Test
    public void test3(){
        StringBuilder s = new StringBuilder("helloworld");
        s.delete(1, 3);
        s.deleteCharAt(4);
        System.out.println(s);
    }


    @Test
    public void test2(){
        StringBuilder s = new StringBuilder("helloworld");
        s.insert(5, "java");
        s.insert(5, "chailinyan");
        System.out.println(s);
    }

    @Test
    public void test1(){
        StringBuilder s = new StringBuilder();
        s.append("hello").append(true).append('a').append(12).append("atguigu");
        System.out.println(s);
        System.out.println(s.length());
    }
```

9.2.3 效率测试

```
/*
 * Runtime：JVM运行时环境
 * Runtime是一个单例的实现
 */
public class TestTime {
    public static void main(String[] args) {
//      testStringBuilder();
        testStringBuffer();
//      testString();
    }
    public static void testString(){
        long start = System.currentTimeMillis();
        String s = new String("0");
        for(int i=1;i<=10000;i++){
            s += i;
        }
        long end = System.currentTimeMillis();
        System.out.println("String拼接+用时："+(end-start));//444

    }
    public static void testStringBuilder(){
        long start = System.currentTimeMillis();
```

```
        StringBuilder s = new StringBuilder("0");
        for(int i=1;i<=10000;i++){
            s.append(i);
        }
        long end = System.currentTimeMillis();
        System.out.println("StringBuilder拼接+用时："+(end-start));//4

    }
    public static void testStringBuffer(){
        long start = System.currentTimeMillis();
        StringBuffer s = new StringBuffer("0");
        for(int i=1;i<=10000;i++){
            s.append(i);
        }
        long end = System.currentTimeMillis();
        System.out.println("StringBuffer拼接+用时："+(end-start));//7

    }
}
```

9.3 和数学相关的类

9.3.1 Math类

`java.lang.Math` 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。类似这样的工具类，其所有方法均为静态方法，并且不会创建对象，调用起来非常简单。

- public static final double PI：返回圆周率

```
double pi = Math.PI;
```

- `public static double abs(double a)`：返回 double 值的绝对值。

```
double d1 = Math.abs(-5); //d1的值为5
double d2 = Math.abs(5); //d2的值为5
```

- `public static double ceil(double a)`：返回大于等于参数的最小的整数。

```
double d1 = Math.ceil(3.3); //d1的值为 4.0
double d2 = Math.ceil(-3.3); //d2的值为 -3.0
double d3 = Math.ceil(5.1); //d3的值为 6.0
```

- `public static double floor(double a)`：返回小于等于参数最大的整数。

```
double d1 = Math.floor(3.3); //d1的值为3.0
double d2 = Math.floor(-3.3); //d2的值为-4.0
double d3 = Math.floor(5.1); //d3的值为 5.0
```

- `public static long round(double a)`：返回最接近参数的 long。(相当于四舍五入方法)

```
long d1 = Math.round(5.5); //d1的值为6.0
long d2 = Math.round(5.4); //d2的值为5.0
```

- public static double pow(double a,double b)：返回a的b幂次方法
- public static double sqrt(double a)：返回a的平方根
- public static double random()：返回[0,1)的随机值
- public static double max(double x, double y)：返回x,y中的最大值
- public static double min(double x, double y)：返回x,y中的最小值

```
double result = Math.pow(2,31);
double sqrt = Math.sqrt(256);
double rand = Math.random();
```

练习

请使用`Math`相关的API，计算在`-10.8`到`5.9`之间，绝对值大于`6`或者小于`2.1`的整数有多少个?

```
public class MathTest {
  public static void main(String[] args) {
    // 定义最小值
    double min = -10.8;
    // 定义最大值
    double max = 5.9;
    // 定义变量计数
    int count = 0;
    // 范围内循环
    for (double i = Math.ceil(min); i <= max; i++) {
      // 获取绝对值并判断
      if (Math.abs(i) > 6 || Math.abs(i) < 2.1) {
        // 计数
        count++;
      }
    }
    System.out.println("个数为: " + count + " 个");
  }
}
```

9.3.2 java.util.Random

用于产生随机数

- public Random():创建一个新的随机数生成器。此构造方法将随机数生成器的种子设置为某个值，该值与此构造方法的所有其他调用所用的值完全不同。(没有真正的随机数，需要种子产生随机数，同一个种子产生的伪随机数序列相同)
- public Random(long seed):使用单个 long 种子创建一个新的随机数生成器。该种子是伪随机数生成器的内部状态的初始值，该生成器可通过方法 next(int) 维护。
- boolean nextBoolean():返回下一个伪随机数，它是取自此随机数生成器序列的均匀分布的 boolean 值。
- void nextBytes(byte[] bytes):生成随机字节并将其置于用户提供的 byte 数组中。
- double nextDouble():返回下一个伪随机数，它是取自此随机数生成器序列的、在 0.0 和 1.0 之间均匀分布的 double 值。
- float nextFloat():返回下一个伪随机数，它是取自此随机数生成器序列的、在 0.0 和 1.0 之间均匀分布的 float 值。

- double nextGaussian():返回下一个伪随机数，它是取自此随机数生成器序列的、呈高斯（“正态”）分布的 double 值，其平均值是 0.0，标准差是 1.0。
- int nextInt():返回下一个伪随机数，它是此随机数生成器的序列中均匀分布的 int 值。
- int nextInt(int n):返回一个伪随机数，它是取自此随机数生成器序列的、在 0（包括）和指定值（不包括）之间均匀分布的 int 值。
- long nextLong():返回下一个伪随机数，它是取自此随机数生成器序列的均匀分布的 long 值。

```
    @Test
    public void test03(){
        Random r = new Random();
        System.out.println("随机整数：" + r.nextInt());
        System.out.println("随机小数：" + r.nextDouble());
        System.out.println("随机布尔值：" + r.nextBoolean());
    }
```

9.3.3 BigInteger

不可变的任意精度的整数。

- BigInteger(String val)
- BigInteger add(BigInteger val)
- BigInteger subtract(BigInteger val)
- BigInteger multiply(BigInteger val)
- BigInteger divide(BigInteger val)
- BigInteger remainder(BigInteger val)
- int intValue():将此 BigInteger 转换为 int。
- long longValue():将此 BigInteger 转换为 long。
- float floatValue():将此 BigInteger 转换为 float。
- ....

```
    @Test
    public void test01(){
//      long bigNum = 123456789123456789123456789L;

        BigInteger b1 = new BigInteger("123456789123456789123456789");
        BigInteger b2 = new BigInteger("789234567891234567891234567891");

//      System.out.println("和：" + (b1+b2));//错误的，无法直接使用+进行求和

        System.out.println("和：" + b1.add(b2));
        System.out.println("减：" + b1.subtract(b2));
        System.out.println("乘：" + b1.multiply(b2));
        System.out.println("除：" + b2.divide(b1));
        System.out.println("余：" + b2.remainder(b1));
    }
```

9.3.4 BigDecimal

不可变的、任意精度的有符号十进制数。

- BigDecimal(String val)
- BigDecimal add(BigDecimal val)
- BigDecimal subtract(BigDecimal val)

- BigDecimal multiply(BigDecimal val)
- BigDecimal divide(BigDecimal val)
- BigDecimal divide(BigDecimal divisor, int roundingMode)
- BigDecimal divide(BigDecimal divisor, int scale, RoundingMode roundingMode)
- BigDecimal remainder(BigDecimal val)
- double doubleValue():将此 BigDecimal 转换为 double。
- ....

```
    @Test
    public void test02(){
        /*double big = 12.123456789123456789123456789;
        System.out.println("big = " + big);*/

        BigDecimal b1 = new BigDecimal("123.456789123456789123456789123456678");
        BigDecimal b2 = new
BigDecimal("7.89234567891234567891234567899988988888");

//      System.out.println("和：" + (b1+b2));//错误的，无法直接使用+进行求和

        System.out.println("和：" + b1.add(b2));
        System.out.println("减：" + b1.subtract(b2));
        System.out.println("乘：" + b1.multiply(b2));
        System.out.println("除：" +
b1.divide(b2,20,RoundingMode.UP));//divide(BigDecimal divisor, int scale, int
roundingMode)
        System.out.println("除：" +
b1.divide(b2,20,RoundingMode.DOWN));//divide(BigDecimal divisor, int scale, int
roundingMode)
        System.out.println("余：" + b1.remainder(b2));
    }
    //保留两位小数的方式：
    @Test
    public void test02(){
        double f = 111231.5585;
        BigDecimal bg = new BigDecimal(f);
        double f1 = bg.setScale(2, BigDecimal.ROUND_HALF_UP).doubleValue();
        System.out.println(f1);
    }
```

9.4 日期时间API

9.4.1 java.util.Date

new Date()：当前系统时间

long getTime()：返回该日期时间对象距离1970-1-1 0.0.0 0毫秒之间的毫秒值

new Date(long 毫秒)：把该毫秒值换算成日期时间对象

```
    @Test
    public void test5(){
        long time = Long.MAX_VALUE;
        Date d = new Date(time);
        System.out.println(d);
    }
```

```
    @Test
    public void test4(){
        long time = 1559807047979L;
        Date d = new Date(time);
        System.out.println(d);
    }
    @Test
    public void test3(){
        Date d = new Date();
        long time = d.getTime();
        System.out.println(time);//1559807047979
    }

    @Test
    public void test2(){
        long time = System.currentTimeMillis();
        System.out.println(time);//1559806982971
        //当前系统时间距离1970-1-1 0:0:0 0毫秒的时间差，毫秒为单位
    }

    @Test
    public void test1(){
        Date d = new Date();
        System.out.println(d);
    }
```

9.4.2 java.util.Calendar

`Calendar` 类是一个抽象类，它为特定瞬间与一组诸如 `YEAR`、`MONTH`、`DAY_OF_MONTH`、`HOUR` 等 日历字段 之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。瞬间可用毫秒值来表示，它是距历元（即格林威治标准时间 1970 年 1 月 1 日的 00:00:00.000，格里高利历）的偏移量。与其他语言环境敏感类一样，`Calendar` 提供了一个类方法 `getInstance`，以获得此类型的一个通用的对象。

（1）getInstance()：得到Calendar的对象

（2）get(常量)

```
    @Test
    public void test6(){
        Calendar c = Calendar.getInstance();
        System.out.println(c);

        int year = c.get(Calendar.YEAR);
        System.out.println(year);

        int month = c.get(Calendar.MONTH)+1;
        System.out.println(month);

        //...
    }

    @Test
    public void test7(){
        TimeZone t = TimeZone.getTimeZone("America/Los_Angeles");
```

```
        //getInstance(TimeZone zone)
        Calendar c = Calendar.getInstance(t);
        System.out.println(c);
    }
```

9.4.3 java.text.SimpleDateFormat

SimpleDateFormat用于日期时间的格式化。

字母	日期或时间元素	表示	示例
G	Era 标志符	Text	AD
y	年	Year	1996; 96
M	年中的月份	Month	July; Jul; 07
w	年中的周数	Number	27
W	月份中的周数	Number	2
D	年中的天数	Number	189
d	月份中的天数	Number	10
F	月份中的星期	Number	2
E	星期中的天数	Text	Tuesday; Tue
a	Am/pm 标记	Text	PM
H	一天中的小时数（0-23）	Number	0
k	一天中的小时数（1-24）	Number	24
K	am/pm 中的小时数（0-11）	Number	0
h	am/pm 中的小时数（1-12）	Number	12
m	小时中的分钟数	Number	30
s	分钟中的秒数	Number	55
S	毫秒数	Number	978
z	时区	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	时区	RFC 822 time zone	-0800

```
    @Test
    public void test10() throws ParseException{
        String str = "2019年06月06日 16时03分14秒 545毫秒  星期四 +0800";
        SimpleDateFormat sf = new SimpleDateFormat("yyyy年MM月dd日 HH时mm分ss秒 SSS
毫秒  E Z");
        Date d = sf.parse(str);
        System.out.println(d);
    }

    @Test
    public void test9(){
        Date d = new Date();

        SimpleDateFormat sf = new SimpleDateFormat("yyyy年MM月dd日 HH时mm分ss秒 SSS
毫秒  E Z");
        //把Date日期转成字符串，按照指定的格式转
        String str = sf.format(d);
        System.out.println(str);
    }
```

9.4.4 JDK8后日期类

Java1.0中包含了一个Date类，但是它的大多数方法已经在Java 1.1引入Calendar类之后被弃用了。而Calendar并不比Date好多少。它们面临的问题是：

- 可变性：象日期和时间这样的类对象应该是不可变的。Calendar类中可以使用三种方法更改日历字段：set()、add() 和 roll()。
- 偏移性：Date中的年份是从1900开始的，而月份都是从0开始的。
- 格式化：格式化只对Date有用，Calendar则不行。
- 此外，它们也不是线程安全的等。

可以说，对日期和时间的操作一直是Java程序员最痛苦的地方之一。第三次引入的API是成功的，并且java 8中引入的java.time API 已经纠正了过去的缺陷，将来很长一段时间内它都会为我们服务。

Java 8 吸收了 Joda-Time （第三方开发）的精华，以一个新的开始为 Java 创建优秀的 API。

- java.time – 包含值对象的基础包
- java.time.chrono – 提供对不同的日历系统的访问。
- java.time.format – 格式化和解析时间和日期
- java.time.temporal – 包括底层框架和扩展特性
- java.time.zone – 包含时区支持的类

Java 8 吸收了 Joda-Time 的精华，以一个新的开始为 Java 创建优秀的 API。新的 java.time 中包含了所有关于时钟（Clock），本地日期（LocalDate）、本地时间（LocalTime）、本地日期时间（LocalDateTime）、时区（ZonedDateTime）和持续时间（Duration）的类。

1、LocalDate、LocalTime、LocalDateTime

本地日期时间类

方法	描述
now() / now(ZoneId zone)	静态方法，根据当前时间创建对象/指定时区的对象
of()	静态方法，根据指定日期/时间创建对象
getDayOfMonth()/getDayOfYear()	获得月份天数(1-31) /获得年份天数(1-366)
getDayOfWeek()	获得星期几(返回一个 DayOfWeek 枚举值)
getMonth()	获得月份, 返回一个 Month 枚举值
getMonthValue() / getYear()	获得月份(1-12) /获得年份
getHours()/getMinute()/getSecond()	获得当前对象对应的小时、分钟、秒
withDayOfMonth()/withDayOfYear()/withMonth()/withYear()	将月份天数、年份天数、月份、年份修改为指定的值并返回新的对象
with(TemporalAdjuster t)	将当前日期时间设置为校对器指定的日期时间
plusDays(), plusWeeks(), plusMonths(), plusYears(),plusHours()	向当前对象添加几天、几周、几个月、几年、几小时
minusMonths() / minusWeeks()/minusDays()/minusYears()/minusHours()	从当前对象减去几月、几周、几天、几年、几小时
plus(TemporalAmount t)/minus(TemporalAmount t)	添加或减少一个 Duration 或 Period
isBefore()/isAfter()	比较两个 LocalDate
isLeapYear()	判断是否是闰年（在LocalDate类中声明）
format(DateTimeFormatter t)	格式化本地日期、时间，返回一个字符串
parse(Charsequence text)	将指定格式的字符串解析为日期、时间

```
    @Test
    public void test7(){
        LocalDate now = LocalDate.now();
        LocalDate before = now.minusDays(100);
        System.out.println(before);//2019-02-26
    }

    @Test
```

```
    public void test06(){
        LocalDate lai = LocalDate.of(2019, 5, 13);
        LocalDate go = lai.plusDays(160);
        System.out.println(go);//2019-10-20
    }

    @Test
    public void test05(){
        LocalDate lai = LocalDate.of(2019, 5, 13);
        System.out.println(lai.getDayOfYear());
    }


    @Test
    public void test04(){
        LocalDate lai = LocalDate.of(2019, 5, 13);
        System.out.println(lai);
    }

    @Test
    public void test03(){
        LocalDateTime now = LocalDateTime.now();
        System.out.println(now);
    }

    @Test
    public void test02(){
        LocalTime now = LocalTime.now();
        System.out.println(now);
    }

    @Test
    public void test01(){
        LocalDate now = LocalDate.now();
        System.out.println(now);
    }
```

2、指定时区日期时间：ZonedDateTime

常见时区ID：

```
Asia/Shanghai
UTC
America/New_York
```

```
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class TestZonedDateTime {
    public static void main(String[] args) {
        ZonedDateTime t = ZonedDateTime.now();
        System.out.println(t);

        ZonedDateTime t1 = ZonedDateTime.now(ZoneId.of("America/New_York"));
        System.out.println(t1);
    }
}
```

3、持续日期/时间：Period和Duration

Period:用于计算两个“日期”间隔

```
public static void main(String[] args) {
        LocalDate t1 = LocalDate.now();
        LocalDate t2 = LocalDate.of(2018, 12, 31);
        Period between = Period.between(t1, t2);
        System.out.println(between);

        System.out.println("相差的年数："+between.getYears());//1年
        System.out.println("相差的月数："+between.getMonths());//又7个月
        System.out.println("相差的天数："+between.getDays());//零25天
        System.out.println("相差的总数："+between.toTotalMonths());//总共19个月
    }
```

Duration:用于计算两个“时间”间隔

```
    public static void main(String[] args) {
        LocalDateTime t1 = LocalDateTime.now();
        LocalDateTime t2 = LocalDateTime.of(2017, 8, 29, 0, 0, 0, 0);
        Duration between = Duration.between(t1, t2);
        System.out.println(between);

        System.out.println("相差的总天数："+between.toDays());
        System.out.println("相差的总小时数："+between.toHours());
        System.out.println("相差的总分钟数："+between.toMinutes());
        System.out.println("相差的总秒数："+between.getSeconds());
        System.out.println("相差的总毫秒数："+between.toMillis());
        System.out.println("相差的总纳秒数："+between.toNanos());
        System.out.println("不够一秒的纳秒数："+between.getNano());
    }
```

4、DateTimeFormatter：日期时间格式化

该类提供了三种格式化方法：

预定义的标准格式。如：`DateTimeFormatter.ISO_DATE_TIME;` `ISO_DATE`

本地化相关的格式。如：`DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM)`

自定义的格式。如：`DateTimeFormatter.ofPattern(“yyyy-MM-dd hh:mm:ss”)`

```
    @Test
    public void test(){
        LocalDateTime now = LocalDateTime.now();
        //预定义的标准格式
        DateTimeFormatter df = DateTimeFormatter.ISO_DATE_TIME;//2019-06-
06T16:38:23.756
        //格式化操作
        String str = df.format(now);
        System.out.println(str);
    }

    @Test
    public void test1(){
        LocalDateTime now = LocalDateTime.now();
        //本地化相关的格式
//      DateTimeFormatter df =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG);//2019年6月6日 下午04时40分
03秒
        DateTimeFormatter df =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT);//19-6-6 下午4:40
        //格式化操作
        String str = df.format(now);
        System.out.println(str);
    }


    @Test
    public void test2(){
        LocalDateTime now = LocalDateTime.now();
        //自定义的格式
        DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy年MM月dd日 HH时mm
分ss秒  SSS毫秒  E 是这一年的D天");
         //格式化操作
        String str = df.format(now);
        System.out.println(str);
    }

    //把字符串解析为日期对象
    public void test3(){
        //自定义的格式
        DateTimeFormatter pattern = DateTimeFormatter.ofPattern("yyyy.MM.dd");
        //解析操作
        LocalDate parse = LocalDate.parse("2020.12.12", pattern);
        System.out.println(parse);
    }
```

9.5 系统相关类

9.5.1 java.lang.System类

系统类中很多好用的方法，其中几个如下：

- static long currentTimeMillis()： 返回当前系统时间距离1970-1-1 0:0:0的毫秒值
- static void exit(int status)： 退出当前系统
- static void gc()： 运行垃圾回收器。

- static String getProperty(String key)：获取某个系统属性
- ...

```
public class Test{
    public static void main(String[] args){
        long time = System.currentTimeMillis();
        System.out.println("现在的系统时间距离1970年1月1日凌晨：" + time + "毫秒");

        System.exit(0);

        System.out.println("over");//不会执行
    }
}
```

9.5.2 java.lang.Runtime类

每个 Java 应用程序都有一个 `Runtime` 类实例，使应用程序能够与其运行的环境相连接。可以通过 `getRuntime` 方法获取当前运行时。 应用程序不能创建自己的 Runtime 类实例。

public static Runtime getRuntime()： 返回与当前 Java 应用程序相关的运行时对象。

public long totalMemory()：返回 Java 虚拟机中的内存总量。此方法返回的值可能随时间的推移而变化，这取决于主机环境。

public long freeMemory()：回 Java 虚拟机中的空闲内存量。调用 gc 方法可能导致 freeMemory 返回值的增加。

public long maxMemory()： 返回 Java 虚拟机试图使用的最大内存量。

Process exec(String command):在单独的进程中执行指定的字符串命令。

9.6 数组工具类

java.util.Arrays数组工具类，提供了很多静态方法来对数组进行操作，而且如下每一个方法都有各种重载形式，以下只列出int[]类型的，其他类型的数组类推：

- static int binarySearch(int[] a, int key)： 要求数组有序，在数组中查找key是否存在，如果存在返回第一次找到的下标，不存在返回负数
- static int[] copyOf(int[] original, int newLength) ： 根据original原数组复制一个长度为newLength的新数组，并返回新数组
- static int[] copyOfRange(int[] original, int from, int to)： 复制original原数组的[from,to)构成新数组，并返回新数组
- static boolean equals(int[] a, int[] a2)： 比较两个数组的长度、元素是否完全相同
- static void fill(int[] a, int val)： 用val填充整个a数组
- static void fill(int[] a, int fromIndex, int toIndex, int val)：将a数组[fromIndex,toIndex)部分填充为val
- static void sort(int[] a)： 将a数组按照从小到大进行排序
- static void sort(int[] a, int fromIndex, int toIndex)： 将a数组的[fromIndex, toIndex)部分按照升序排列
- static String toString(int[] a)： 把a数组的元素，拼接为一个字符串，形式为：[元素1，元素2，元素3。。。]

示例代码：

```java
import java.util.Arrays;
import java.util.Random;

public class Test{
    public static void main(String[] args){
        int[] arr = new int[5];
        // 打印数组,输出地址值
        System.out.println(arr); // [I@2ac1fdc4
        // 数组内容转为字符串
        System.out.println("arr数组初始状态："+ Arrays.toString(arr));

        Arrays.fill(arr, 3);
        System.out.println("arr数组现在状态："+ Arrays.toString(arr));

        Random rand = new Random();
        for (int i = 0; i < arr.length; i++) {
            arr[i] = rand.nextInt(100);//赋值为100以内的随机整数
        }
        System.out.println("arr数组现在状态："+ Arrays.toString(arr));

        int[] arr2 = Arrays.copyOf(arr, 10);
        System.out.println("新数组：" + Arrays.toString(arr2));

        System.out.println("两个数组的比较结果：" + Arrays.equals(arr, arr2));

        Arrays.sort(arr);
        System.out.println("arr数组现在状态："+ Arrays.toString(arr));
    }
}
```

9.7 包装类

9.7.1 包装类

Java提供了两大类数据类型，基本类型与引用类型，使用基本类型在于效率，但是缺少像引用数据类型一样的丰富API，那么Java提供了针对基本数据类型的保证类，以提供更加便捷的操作功能，包装类就是把基本数据类型包装成对应的引用数据类型。

序号	基本数据类型	包装类（java.lang包）
1	byte	Byte
2	short	Short
3	int	**Integer**
4	long	Long
5	float	Float
6	double	Double
7	char	**Character**
8	boolean	Boolean
9	void	Void

9.7.2 装箱与拆箱

装箱：把基本数据类型转为包装类对象。

> 转为包装类的对象，是为了使用专门为对象设计的API和特性

拆箱：把包装类对象拆为基本数据类型。

> 转为基本数据类型，一般是因为需要运算，Java中的大多数运算符是为基本数据类型设计的。比较、算术等

基本数值---->包装对象

```
Integer i1 = new Integer(4);//使用构造函数函数
Integer i2 = Integer.valueOf(4);//使用包装类中的valueOf方法
```

包装对象---->基本数值

```
Integer i1 = new Integer(4);
int num1 = i1.intValue();
```

JDK1.5之后，可以自动装箱与拆箱。

> 注意：只能与自己对应的类型之间才能实现自动装箱与拆箱。

```
Integer i = 4;//自动装箱。相当于Integer i = Integer.valueOf(4);
i = i + 5;//等号右边：将i对象转成基本数值(自动拆箱) i.intValue() + 5;
//加法运算完成后，再次装箱，把基本数值转成对象。
```

```
Integer i = 1;
Double d = 1;//错误的，1是int类型
```

总结：对象（引用数据类型）能用的运算符有哪些？

（1）instanceof

（2）=：赋值运算符

（3）==和!=：用于比较地址，但是要求左右两边对象的类型一致或者是有父子类继承关系。

（4）对于字符串这一种特殊的对象，支持“+”，表示拼接。

9.7.3 包装类的常用API

1. **基本数据类型和字符串之间的转换**

   （1）把基本数据类型转为字符串

   ```
   int a = 10;
   //String str = a;//错误的
   //方式一：
   String str = a + "";
   //方式二：
   String str = String.valueOf(a);
   //方式三：
   Integer i=10;
   String str=i.toString();
   ```

（2）把字符串转为基本数据类型

String转换成对应的基本类型，除了Character类之外，其他所有包装类都具有parseXxx静态方法可以将字符串参数转换为对应的基本类型，例如：

- `public static int parseInt(String s)`：将字符串参数转换为对应的int基本类型。
- `public static long parseLong(String s)`：将字符串参数转换为对应的long基本类型。
- `public static double parseDouble(String s)`：将字符串参数转换为对应的double基本类型。

或把字符串转为包装类，然后可以自动拆箱为基本数据类型

- `public static Integer valueOf(String s)`：将字符串参数转换为对应的Integer包装类，然后可以自动拆箱为int基本类型
- `public static Long valueOf(String s)`：将字符串参数转换为对应的Long包装类，然后可以自动拆箱为long基本类型
- `public static Double valueOf(String s)`：将字符串参数转换为对应的Double包装类，然后可以自动拆箱为double基本类型

注意:如果字符串参数的内容无法正确转换为对应的基本类型，则会抛出`java.lang.NumberFormatException`异常。

```
int a = Integer.parseInt("整数的字符串");
double d = Double.parseDouble("小数的字符串");
boolean b = Boolean.parseBoolean("true或false");

int a = Integer.valueOf("整数的字符串");
double d = Double.valueOf("小数的字符串");
boolean b = Boolean.valueOf("true或false");
```

2. **数据类型的最大最小值**

```
Integer.MAX_VALUE和Integer.MIN_VALUE
Long.MAX_VALUE和Long.MIN_VALUE
Double.MAX_VALUE和Double.MIN_VALUE
```

3. **字符转大小写**

```
Character.toUpperCase('x');
Character.toLowerCase('X');
```

4. **整数转进制**

```
Integer.toBinaryString(int i)
Integer.toHexString(int i)
Integer.toOctalString(int i)
```

9.7.4 包装类对象的缓存问题

包装类的数据在缓存数值范围内时，直接从内存中取出对象，超过范围会创建新的对象

包装类	缓存对象
Byte	-128~127
Short	-128~127
Integer	-128~127
Long	-128~127
Float	没有
Double	没有
Character	0~127
Boolean	true和false

示例代码：

```
Integer i = 1;
Integer j = 1;
System.out.println(i == j);//true

Integer i = 128;
Integer j = 128;
System.out.println(i == j);//false

Integer i = new Integer(1);//新new的在堆中
Integer j = 1;//这个用的是缓冲的常量对象，在方法区
System.out.println(i == j);//false

Integer i = new Integer(1);//新new的在堆中
Integer j = new Integer(1);//另一个新new的在堆中
System.out.println(i == j);//false
```

```
    @Test
    public void test3(){
        Double d1 = 1.0;
        Double d2 = 1.0;
        System.out.println(d1==d2);//false 比较地址，没有缓存对象，每一个都是新new的
    }
```

> 以上示例代码证明缓存的存在，源码中也有体现，

9.7.5 面试题

1. **类型转换问题**

```
    @Test
    public void test4(){
        Double d1 = 1.0;
        double d2 = 1.0;
        System.out.println(d1==d2);//true 和基本数据类型比较会自动拆箱，比较数据值
    }
```

```
    @Test
    public void test2(){
        Integer i = 1000;
        double j = 1000;
        System.out.println(i==j);//true  会先将i自动拆箱为int，然后根据基本数据类型“自动类型转换”规则，转为double比较
    }

    @Test
    public void test(){
        Integer i = 1000;
        int j = 1000;
        System.out.println(i==j);//true 会自动拆箱，按照基本数据类型进行比较
    }
```

2. **不可变对象**

```
public class TestExam {
    public static void main(String[] args) {
        int i = 1;
        Integer j = new Integer(2);
        Circle c = new Circle();
        change(i,j,c);
        System.out.println("i = " + i);//1
        System.out.println("j = " + j);//2
        System.out.println("c.radius = " + c.radius);//10.0
    }

    /*
     * 方法的参数传递机制：
     * （1）基本数据类型：形参的修改完全不影响实参
     * （2）引用数据类型：通过形参修改对象的属性值，会影响实参的属性值
     * 这类Integer等包装类对象是“不可变”对象，即一旦修改，就是新对象，和实参就无关了
     */
    public static void change(int a ,Integer b,Circle c ){
        a += 10;
//      b += 10;//等价于  b = new Integer(b+10);
        c.radius += 10;
        /*c = new Circle();
        c.radius+=10;*/
    }
}
class Circle{
    double radius;
}
```

9.8 枚举Enum

9.8.1 概述

某些类型的对象是有限的几个，这样的例子举不胜举：

- 星期：Monday(星期一)……Sunday(星期天)
- 性别：Man(男)、Woman(女)
- 月份：January(1月)……December(12月)
- 季节：Spring(春节)……Winter(冬天)

- 支付方式：Cash（现金）、WeChatPay（微信）、Alipay(支付宝)、BankCard(银行卡)、CreditCard(信用卡)
- 员工工作状态：Busy（忙）、Free（闲）、Vocation（休假）
- 订单状态：Nonpayment（未付款）、Paid（已付款）、Fulfilled（已配货）、Delivered（已发货）、Checked（已确认收货）、Return（退货）、Exchange（换货）、Cancel（取消）

枚举类型本质上也是一种类，只不过是这个类的对象是固定的几个，而不能随意让用户创建。

在JDK1.5之前，需要程序员自己通过特殊的方式来定义枚举类型。

在JDK1.5之后，Java支持enum关键字来快速的定义枚举类型。

9.8.2 枚举类的创建

在JDK1.5之前如何声明枚举类呢?

- 构造器加private私有化
- 本类内部创建一组常量对象，并添加public static修饰符，对外暴露这些常量对象

示例代码：

```
public class TestEnum {
    public static void main(String[] args) {
        Season spring = Season.SPRING;
        System.out.println(spring);
    }
}
class Season{
    private String seasonName;

    public static final Season SPRING=new Season("春");
    public static final Season SUMMER=new Season("夏");
    public static final Season AUTUMN=new Season("秋");
    public static final Season WINTER=new Season("冬");

    private Season(String name){
        this.seasonName=name;
    }

    @Override
    public String toString() {
        return seasonName;
    }
}
```

JDK1.5之后的枚举

语法格式：

```
【修饰符】 enum 枚举类名{
    常量对象列表
}

【修饰符】 enum 枚举类名{
    常量对象列表;

    其他成员列表;
}
```

示例代码：

```
public class TestEnum {
    public static void main(String[] args) {
        Season spring = Season.SPRING;
        System.out.println(spring);
    }
}
enum Season{
    SPRING,SUMMER,AUTUMN,WINTER
}
```

示例代码：

```
public class TestEnum {
    public static void main(String[] args) {
        Season spring = Season.SPRING;
        System.out.println(spring);
    }
}
enum Season{
    SPRING("春"),SUMMER("夏"),AUTUMN("秋"),WINTER("冬");
    private final String description;

    private Season(String description){
        this.description = description;
    }

    public String toString(){//需要手动编写，无法使用Generate toString()...
        return description;
    }
}
```

枚举类的要求和特点：

- 枚举类的常量对象列表必须在枚举类的首行，因为是常量，所以建议大写。
- 如果常量对象列表后面没有其他代码，那么“；”可以省略，否则不可以省略“；”。
- 编译器给枚举类默认提供的是private的无参构造，如果枚举类需要的是无参构造，就不需要声明，写常量对象列表时也不用加参数，
- 如果枚举类需要的是有参构造，需要手动定义private的有参构造，调用有参构造的方法就是在常量对象名后面加(实参列表)就可以。
- 枚举类默认继承的是java.lang.Enum类，因此不能再继承其他的类型。
- JDK1.5之后switch，提供支持枚举类型，case后面可以写枚举常量名。
- 枚举类型如有其它属性，建议（**不是必须**）这些属性也声明为final的，因为常量对象在逻辑意义上应该不可变。

9.8.3 枚举类型常用方法

```
1.toString(): 默认返回的是常量名（对象名），可以继续手动重写该方法!
2.name():返回的是常量名（对象名） 【很少使用】
3.ordinal():返回常量的次序号，默认从0开始
4.values():返回该枚举类的所有的常量对象，返回类型是当前枚举的数组类型，是一个静态方法
5.valueOf(String name)：根据枚举常量对象名称获取枚举对象
```

示例代码：

```
public class TestEnum {
    public static void main(String[] args) {
        Season[] values = Season.values();
        for (int i = 0; i < values.length; i++) {
            switch(values[i]){
            case SPRING:
                System.out.println(values[i]+":春暖花开，万物复苏");
                break;
            case SUMMER:
                System.out.println(values[i]+":百花争艳，郁郁葱葱");
                break;
            case AUTUMN:
                System.out.println(values[i]+":菊桂飘香，百树凋零");
                break;
            case WINTER:
                System.out.println(values[i]+":梅花独开，大地一色");
                break;
            }
        }
    }
}
enum Season{
    SPRING,SUMMER,AUTUMN,WINTER
}
```

练习

1、声明月份枚举类Month：

（1）创建：1-12月常量对象

```
JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY,AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DEC
EMBER
```

（2）声明两个属性：value（月份值，例如：JANUARY的value为1），
description（描述，例如：JANUARY的description为1月份是一年的开始）。

（3）声明一个有参构造，创建12个对象

（4）声明一个方法：public static Month getByValue(int value)

（5）手动重写toString()：返回对象信息，例如：1->JANUARY->1月份是一年的开始。

2、在测试类中，从键盘输入1个1-12的月份值，获取对应的月份对象，并打印对象