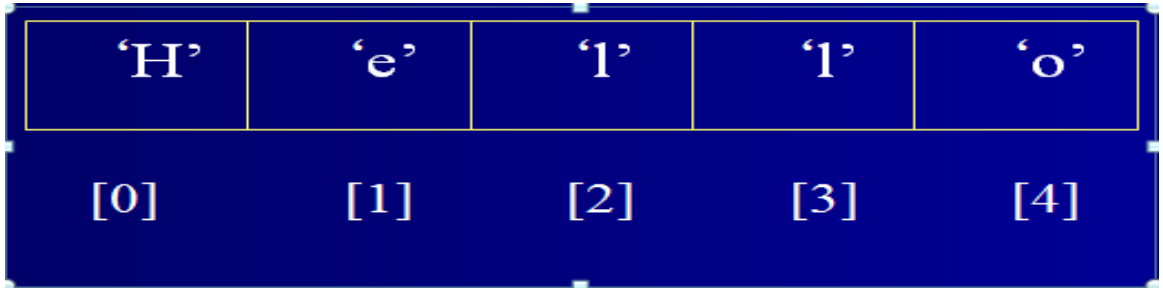
第十章：常用系统类

教学内容：

1. String 和 StringBuffer
2. 标准输入输出
3. Math 类
4. Date 类
5. Random 类

一、 String 和 StringBuffer

- 字符串指的是字符的序列，有两种类型的字符串：一种是创建以后不需要改变的，称为字符串常量，在 Java 中，String 类用于存储和处理字符串常量；另外一种字符串是创建以后，需要对其进行改变的，称为字符串变量，在 Java 中，StringBuffer 类用于存储和操作字符串变量。
- 注意：字符串不是 char[]数组
- 字符串内部各个字符的排列位置是连续的，位置编号由开始到结束是从 0 到 n。



字符串常量和 String 类：

- 在 Java 中，没有内置的字符串类型，字符串常量是作为 String 类的对象存在的。

1．创建 String 类对象

String 类的对象表示的是字符串常量，一个字符串常量创建以后就不能够被修改了。所以在创建 String 类对象时，通常需要向构造函数传递参数来指定创建的字符串的内容。以下是常用的 String 类构造函数：

（1）public String（ ）

该构造函数用于创建一个空的字符串常量。

（2）public String（String value）

该构造函数用于根据一个已经存在的字符串常量来创建一个新的字符串常量，该字符串的内容和已经存在的字符串常量一致。

（3）public String（char[] value）

该构造函数用于根据一个已经存在的字符数组来创建一个新的字符串常量。

（4）public String（StringBuffer buffer）

该构造函数用于根据一个已经存在的 StringBuffer 对象来创建一个新的字符串常量。

创建 String 对象示例：

String s1 = “Egg”;//建议使用该方式

或者

String s2 = new String("Egg");

区别：编译器遇到像“Egg”这样的“字符串常量”时，它会自动创建一个含有“Egg”字符串的对象，然后把该对象的引用传给 s1. 而处理 s2 时，编译器一遇到构造函数的参数“Egg”时，会首先创建一个含有“Egg”字符串的对象（匿名

的）。然后，根据构造函数“new String(” Egg “)”的要求，再创建出另一个含有“Egg”字符串的对象（也是匿名的），然后把该对象的引用传给 s2。这样一来，就多创建了一个匿名的 String 对象。

2．String 类对象的常用操作及方法

在 Java 中，String 类包含有 50 多个方法来实现字符串的各种操作，以下介绍一些我们需要经常使用的方法。

（1）字符串的连接

public String concat(String str)

该方法的参数为一个 String 类对象，作用是将参数中的字符串 str 连接到原来字符串的后面。

"Hello".concat(" World!") => "Hello World!"

（2）求字符串的长度

public int length()

返回字串的长度，这里的长度指的是字串中 Unicode 字符的数目。

"Hello".length() => 5

"".length() => 0

（3）求字符串中某一位置的字符

public char charAt(int index)

该方法在一个特定的位置索引一个字符串，以得到字符串中指定位置的字符。值得注意的是，在字符串中第一个字符的索引是 0，第二个字符的索引是 1，依次类推，最后一个字符的索引是 length()-1。

" Hello".charAt(0) => ` `

" Hello".charAt(1) => 'H'

（4）字符串的比较

比较字符串可以利用 String 类提供的下列方法：

1）public int compareTo（String anotherString）

该方法比较两个字符串，和 Character 类提供的 compareTo 方法相似，Character 类提供的 compareTo 方法比较的是两个字符类数据，而这里比较的是字符串数据。其比较过程实际上是两个字符串中相同位置上的字符按 Unicode 中排列顺序逐个比较的结果。如果在整个比较过程中，没有发现任何不同的地方，则表明两个字符串是完全相等的， compareTo 方法返回 0；如果在比较过程中，发现了不同的地方，则比较过程会停下来，这时一定是两个字符串在某个位置上不相同，如果当前字符串在这个位置上的字符大于参数中的这个位置上的字符，compareTo 方法返回一个大于 0 的整数，否则返回一个小于 0 的整数。

2）public boolean equals（Object anObject）

该方法比较两个字符串，和 Character 类提供的 equals 方法相似，因为它们都是重载 Object 类的方法。该方法比较当前字符串和参数字符串，在两个字符串相等的时候返回 true，否则返回 false。

"Hello".equals("Hello") => true

"Hello".equals("hello") => false

3）public boolean equalsIgnoreCase（String anotherString）

该方法和 equals 方法相似，不同的地方在于，equalsIgnoreCase 方法将忽略字母大小写的区别。

"Hello".equalsIgnoreCase("hello") => true

"Hello".equalsIgnoreCase("hELLo") => true

（5）从字符串中提取子串

利用 String 类提供的 substring 方法可以从一个大的字符串中提取一个子串，该方法有两种常用的形式：

1）public String substring（int beginIndex）

该方法从 beginIndex 位置起，从当前字符串中取出剩余的字符作为一个新的字符串返回。

"Hello".substring(2) => "llo"

"Hello".substring(1) => "ello"

2）public String substring（int beginIndex, int endIndex）

该方法从当前字符串中取出一个子串，该子串从 beginIndex 位置起至 endIndex-1 为结束。子串返的长度为 endIndex-beginIndex。

"Hello".substring(1,3) => "el"

"Hello".substring(2,4) => "ll"

（6）判断字符串的前缀和后缀

判断字符串的前缀是否为指定的字符串利用 String 类提供的下列方法：

1）public boolean startsWith（String prefix）

该方法用于判断当前字符串的前缀是否和参数中指定的字符串 prefix 一致，如果是，返回 true，否则返回 false。

"Hello".startsWith("He") => true

"Hello".startsWith("el") => false

2）public boolean startsWith（String prefix, int toffset）

该方法用于判断当前字符串从 toffset 位置开始的子串的前缀是否和参数中指定的字符串 prefix 一致，如果是，返回 true，否则返回 false。

"Hello".startsWith("He",0) => true

"Hello".startsWith("He",1) => false

判断字符串的后缀是否为指定的字符串利用 String 类提供的方法：

public boolean endsWith（String suffix）

该方法用于判断当前字符串的后缀是否和参数中指定的字符串 suffix 一致，如果是，返回 true，否则返回 false。

"Hello".endsWith("lo") => true

（7）字符串中单个字符的查找

字符串中单个字符的查找可以利用 String 类提供的下列方法：

1）public int indexOf（String ch）

该方法用于查找当前字符串中某一个特定字符 ch 出现的位置。该方法从头向后查找，如果在字符串中找到字符 ch，则返回字符 ch 在字符串中第一次出现的位置；如果在整个字符串中没有找到字符 ch，则返回-1。

Hello".indexOf('h') => -1

"Hello".indexOf('H') => 0

"Hello".indexOf('l') => 2

2）public int indexOf（String ch, int fromIndex）

该方法和第一种方法类似，不同的地方在于，该方法从 fromIndex 位置向后查找，返回的仍然是字符 ch 在字符串第一次出现的位置。

"Hello".indexOf("He",0) => 0

"Hello".indexOf("He",1) => -1

3）public int lastIndexOf（String ch）

该方法和第一种方法类似，不同的地方在于，该方法从字符串的末尾位置向前查找，返回的仍然是字符 ch 在字符串第一次出现的位置。

"Hello".lastIndexOf('H') => 0

"Hello".lastIndexOf('h') => -1

4）public int lastIndexOf（String ch, int fromIndex）

该方法和第二种方法类似，不同的地方在于，该方法从 fromIndex 位置向前 查找，返回的仍然是字符 ch 在字符串第一次出现的位置。

"Hello".lastIndexOf('H',1) => 0

"Hello".lastIndexOf('o',1) => -1

（8）字符串中子串的查找

字符串中子串的查找与字符串中单个字符的查找十分相似，可以利用 String 类提供的下列方法：

1）public int indexOf（String str）

"Hello".indexOf("He") => 0

“Hello”.indexOf(“he”) => -1//未找到

2）public int indexOf（String str，int fromIndex）

"Hello".indexOf("He",0) => 0

"Hello".indexOf("He",1) => -1

3）public int lastIndexOf（String str）

"Hello".lastIndexOf("He") => 0

"Hello".lastIndexOf("lo") => 3

4）public int lastIndexOf（String str，int fromIndex）

"Hello".lastIndexOf("He",0) => 0

"Hello".lastIndexOf("lo",2) => -1

（9）字符串中字符大小写的转换

字符串中字符大小写的转换，可以利用下列方法：

1）public String toLowerCase（ ）

该方法将字符串中所有字符转换成小写，并返回转换后的新串。

"Hello".toLowerCase() => "hello"

2）public String toUpperCase（ ）

该方法将字符串中所有字符转换成大写，并返回转换后的新串。

"Hello".toUpperCase() => "HELLO"

（10）字符串中多余空格的去除

public String trim()

该方法只是去掉开头和结尾的空格，并返回得到的新字符串。值得注意的是，在原来字符串中间的空格并不去掉。

〝 Hello　".trim() => "Hello"

（11）字符串中字符的替换

1）public String replace（char oldChar,char newChar）

该方法用字符 newChar 替换当前字符串中所有的字符 oldChar，并返回一个新的字符串。

"Hello".replace('l', 'L') => "HeLLo"

2）public String replaceFirst（String regex, String replacement）

该方法用字符串 replacement 的内容替换当前字符串中遇到的第一个和字 符串 regex 相一致的子串，并将产生的新字符串返回。

3）public String replaceAll（String regex, String replacement）

该方法用字符串 replacement 的内容替换当前字符串中遇到的所有 和字符串 regex 相一致的子串，并将产生的新字符串返回。

String 类各方法的小结：

- 因为 String 对象的内容是不可以被更改的，所以以上的方法都不会改变 String 对象的内容。当返回值类型为 String 时，实际上是会产生一个新的 String 对象。

3 . StringBuffer 类

创建 StringBuffer 类对象

StringBuffer 类对象表示的是字符串变量，每一个 StringBuffer 类对象都是可以扩充和修改的字符串变量。以下是常用的 StringBuffer 类构造函数：

（1）public StringBuffer（ ）

（2）public StringBuffer（int length）

（3）public StringBuffer（String str）

StringBuffer 类对象的常用方法

（1）StringBuffer 类对象的扩充

StringBuffer 类提供两组方法用来扩充 StringBuffer 对象所包含的字符，分别是：

1）public StringBuffer append（Object obj）

append 方法用于扩充 StringBuffer 对象所包含的字符，该方法将指定的参数对象转化为字符串后，将其附加在原来的 StringBuffer 对象之后，并返回新的 StringBuffer 对象。附加的参数对象可以是各种数据类型的，如 int、char、String、double 等。

new StringBuffer("Hello:").append(2) => "Hello:2"

new StringBuffer("Hello:").append(new char[] {'a','b'}) => "Hello:ab"

2）public StringBuffer insert（ int 插入位置，参数对象类型 参数对象名）

该方法将指定的参数对象转化为字符串后，将其插入在原来的 StringBuffer 对象中指定的位置，并返回新的 StringBuffer 对象。

new StringBuffer("Hello").insert(1,'c') => "Hcello"

（2）StringBuffer 类对象的长度与容量

一个 StringBuffer 类对象的长度指的是它包含的字符个数；容量指的是被分配的字符空间的数量。

1）public int length（ ）

该方法返回当前 StringBuffer 类对象包含的字符个数。

new StringBuffer("Hello").length() => 5

2）public int capacity（ ）

方法返回当前 StringBuffer 类对象分配的字符空间的数量。

new StringBuffer().capacity() => 16 // 可容納 16 个字符

（3）StringBuffer 类对象的修改

public void setCharAt(int index, char ch)

该方法将当前 StringBuffer 对象中的 index 位置的字符替换为指定的字符 ch。

new StringBuffer("Hello").setCharAt(1,'E') => "HEllo"

（4）字符串的赋值和加法

字符串是在程序中要经常使用的数据类型，在 Java 编译系统中引入了字符串的赋值和加法操作

StringBuffer 方法小结：

因为 StringBuffer 对象的内容可以被更改，所以以上的方法可能会改变 StringBuffer 对象的内容。当返回值类型为 String 时，实际上是会传回原来的 StringBuffer 对象。

4 . String 与 StringBuffer 的区别

- String 与 StringBuffer。它们之间所有的差别是在于
  - String 对象的内容是不可以被更改；而 StringBuffer 的对象内容则是可以被更改的。
  - String 类型的字串其长度是固定的；而 StringBuffer 类型的字串其长度是变动、並非固定的。
- 如果只需要读取字串其中的内容時，String 类型则已足夠

5 . String 和 StringBuffer 的转换

- 产生一个内容与 String 对象 str 相同的 StringBufffer 对象：

StringBuffer sb = new StringBuffer(str);

- 产生一个内容与 StringBuffer 对象 sb 相同的 String 对象：

String sb = sb.toSting();

6 . 字符串的合并

- Java 编译器对 String 的“ + ”运算的实际处理过程：

  对如下代码:

```
String str = "baidu";
```

```
    System.out.println("www."+str+".com");
```

会隐式的将它们编译成：

```
    String str = "baidu";
    System.out.println(new StringBuffer(new
StringBuffer("www.").append(str).toString()).append(".com").toString());
```

测试+连接和 append()的性能：

```
public class Test {

    private static String str;
    private static StringBuffer sb;

    public static long testString() {
        str = "A";
        long start = System.currentTimeMillis();
        for (int i = 0; i < 50000; i++) {
            str += "A";
        }
        long end = System.currentTimeMillis();
        return end - start;
    }

    public static long testStringBuffer() {
        sb = new StringBuffer("A");
        long start = System.currentTimeMillis();
        for (int i = 0; i < 50000; i++) {
            sb.append("A");
```

```
        }
        long end = System.currentTimeMillis();
        return end - start;
    }

    public static void main(String[] args) {
        System.out.println("append消耗的时间：" + testStringBuffer());
        System.out.println("+连接消耗的时间：" + testString());
        System.out.println(str.equals(sb.toString()));//true
    }

}
```

testStringBuffer()方法消耗的时间比testString()消耗的时间短的多，说明append()方法性能跟好。

7．String 的比较

==比较：比较的并非两个字符串的内容，而是所指向的地址是否时同一个地址；

equals()和 equalsIgnoreCase()方法：先比较地址，再比较的是字符串的内容；

compareTo()：比较的是字符的 Unicode 码逐一比较；

注意：以上方法均是 String 的方法，而不是 StringBuffer 的方法，如果 StringBuffer 需要使用必须先使用 toString()，而 equals()是重写的 Object 类的方法.

二、 标准输入输出

- **System 类的这 2 个静态字段就是系统的三个标准流：**
  - **System.in：表示系统的标准输入流,通常的环境下标准输入流指向键盘输入;**
  - **System.out：表示系统的标准输出流,通常环境下指向屏幕输出;**

三、 Math 类

数学类包含了许多数学函数，如 sin、cos、exp、abs 等。Math 类是一个工具类，它在解决与数学有关的一些问题是有着非常重要的作用。

这个类有两个静态属性：E 和 PI。E 代表数学中的 e 2.7182818，而 PI 代表派 pi 3.1415926。

引用时，用法如：Math.E 和 Math.Pi

Math 类中的方法都是静态方法，用法为 Math.***** (*****为方法名)。用法如：

```
int a=Math.abs(124);

int b=Math.floor(-5.2);

double s=Math.sqrt(7);
```

四、 Date 类

- Date 类实际上只是一个包裹类, 它包含的是一个长整型数据, 表示的是从 GMT(格林尼治标准时间)1970 年, 1 月 1 日 00:00:00 这一刻之前或者是之后经历的毫秒数。

- Date 类常用的两个构造函数：
  - Date()无参数的构造函数创建的对象可以获取本地当前时间。
  - Date(long time)使用一个从 GMT(格林尼治标准时间)1970 年, 1 月 1 日 00:00:00 这一刻之前或者是之后经历的毫秒数创建一个 Date 对象
- Date 示例：
  - Date date = new Date();
  - System.out.println(date.getTime()); //1228061164796
  - System.out.println(date.toString()); //Mon Dec 01 00:06:04 CST 2008
- SimpleDateFormat 类

```
Date date = new Date();
//HH表示24小时
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
System.out.println(sdf.format(date));
```

```
Date date = new Date();
DateFormat shortDateFormat = DateFormat.getDateTimeInstance(DateFormat.SHORT,
          DateFormat.SHORT);
DateFormat mediumDateFormat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
          DateFormat.MEDIUM);
DateFormat longDateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
          DateFormat.LONG);
```

```
        DateFormat fullDateFormat = DateFormat.getDateTimeInstance(DateFormat.FULL,
                DateFormat.FULL);
        System.out.println(shortDateFormat.format(date)); //08-12-1 下午1:28
        System.out.println(mediumDateFormat.format(date));//2008-12-1 13:28:31
        System.out.println(longDateFormat.format(date)); //2008年12月1日 下午01时28分31秒
        System.out.println(fullDateFormat.format(date)); //2008 年 12 月 1 日 星期一 下午 01 时 28 分 31
秒 CST
```

五、 Random 类

java.util.Random 类来产生一个随机数发生器。它有两种形式的构造函数，分别是 Random()和 Random(long seed)。Random()使用当前时间即:

System.currentTimeMillis()作为发生器的种子，Random(long seed)使用指定的 seed 作为发生器的种子。随机数发生器(Random)对象产生以后，通过调用不同的 method：nextInt()、nextLong()、nextFloat()、nextDouble()等获得不同类型随机数 。

猜数字游戏：

```
public class Test {

    public static void main(String[] args) {

        int number = new Random().nextInt(20) + 1;
        int count = 0;
        int i = Integer.parseInt(JOptionPane.showInputDialog("请输入一个1-20的整数:"));
        while (i != number) {
```

```
                if (i > number)
                    i = Integer
                            .parseInt(JOptionPane.showInputDialog("第" + (++count) + "次猜，你猜大
了，请重新输入:"));
                else
                    i = Integer
                            .parseInt(JOptionPane.showInputDialog("第" + (++count) + "次猜，你猜小
了，请重新输入:"));
                System.out.println(i);
            }
            JOptionPane.showMessageDialog(null, "你猜对了");
            System.exit(0);
        }

}
```