# 一、Object 类：

1、 Object 类是所有类的终极父类，任何一个类都继承了 Object 类

2、 Object 类的常用方法：

* toString()：

返回该对象的字符串表示（完整类名+@+对象的哈希码）

* toString()方法有何作用(有何好处)？

重写Object父类的toString()方法后，就回输出符合我们所需求的格式数据。

* equals(Object obj)：

用于比较两个对象的内存地址，判断是否为同一个对象

* hashCode()：

返回对象的哈希码值（可以理解为对象的内存地址）

3、java中的规范：一般我们重写了一个类的equals方法，我们都会重写它的hashCode方法

# 二、字符串

## 1、字符串入门

### 1、创建字符串的两种方式

（1）String str1 = “abc”;

这种方式创建的字符串对象，首先jvm会去字符串常量池查看有没有“abc”的对象：

如果有，那么不会再创建对象“abc”,会将该对象的内存地址指向栈内存中的str1;

如果没有，会在字符串常量池中创建一个“abc”对象，然后将该对象的内存地址指向栈内存中的str1

（2）String str3 = new String(“abc”);

这种方式创建的字符串对象，

首先，jvm会去字符串常量池查找有没有“abc”对象，如果有，就不创建；如果没有，就创建一个“abc”对象。

然后，会去堆内存中创建一个String对象，将数据“abc”拷贝一份到String对象中

最后，会在栈内存中创建一个str3变量，将new String() 的内存地址指向str3变量。

### 2、使用equals比较两个字符串

String重写equals的源码：

|  |
| --- |
| public boolean equals(Object anObject) {  if (this == anObject) {  return true;  }  if (anObject instanceof String) {  String anotherString = (String)anObject;  int n = value.length;  if (n == anotherString.value.length) {  char v1[] = value;  char v2[] = anotherString.value;  int i = 0;  while (n-- != 0) {  if (v1[i] != v2[i])  return false;  i++;  }  return true;  }  }  return false; } |

* 2.1、String的equals底层实现：

String重写了Object类的equals方法，将两个字符串转换成两个char类型的数组，比较的是两个字符串的内容是否一致，一致返回true，不一致返回false。

* 2.2、String的equals方法使用技巧

当使用equals比较两个字符串时，将常量字符串放到第一个位置，用字符串常量去比较字符串变量，这样可以防止空指针异常（NullPointerException）的出现。

|  |
| --- |
| public static void main(String[] args) {  *test*(null, ""); }  public static void test(String str1, String str2) {  *// 当字符串为空时，值为false* System.*out*.println("中国".equals(str2));  *// 当空字符串与null比较时，值为false* System.*out*.println(str2.equals(str1));  *// 当null与字符串比较时，出现空指针异常* System.*out*.println(str1.equals("中国")); } |

## 2、字符串的方法

### 1、构造方法摘要

|  |
| --- |
| 1. String 的构造方法摘要：  1、String()  初始化一个新创建的 String 对象，使其表示一个空字符序列。  2、String(byte[] bytes)  通过使用平台的默认字符集解码指定的 byte 数组，构造一个新的 String。  3、String(byte[] bytes, Charset charset)  通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。  4、String(byte[] bytes, int offset, int length)  通过使用平台的默认字符集解码指定的 byte 子数组，构造一个新的 String。  5、String(byte[] bytes, int offset, int length, Charset charset)  通过使用指定的 charset 解码指定的 byte 子数组，构造一个新的 String。  6、String(byte[] bytes, int offset, int length, String charsetName)  通过使用指定的字符集解码指定的 byte 子数组，构造一个新的 String。  7、String(byte[] bytes, String charsetName)  通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。  8、String(char[] value)  分配一个新的 String，使其表示字符数组参数中当前包含的字符序列。  9、String(char[] value, int offset, int count)  分配一个新的 String，它包含取自字符数组参数一个子数组的字符。  10、String(int[] codePoints, int offset, int count)  分配一个新的 String，它包含 Unicode 代码点数组参数一个子数组的字符。  11、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。 |
| 1、String(); 无参构造方法 String s = new String(); System.out.println("String的无参构造方法：" + s);  2、String(byte[] bytes,Charset charse）：  使用指定的的字符集，将byte数组按ASCII码表进行解码，构成一个新的字符串 \*/ byte[] bytes = {97, 98, 99, 100, 101, 102, 103, 104}; String s1 = new String(bytes, "utf-8"); System.out.println(s1);//abcdefgh  3、String(byte[] bytes, int offset, int length,String charsetName):  使用指定的字符集按ASCII表解码指定的byte数组，从下标为 offset的字符开始，取length个字符构成一个新的数组。  String s2 = new String(bytes, 2, 3, "utf-8");  System.out.println(s2);//cde 4、String(char[] value):  将char数组复制，并组成一个新的 String 字符串，这并不影响原char数组。  char[] chars = {'a', 'b', 'c', 'd', 'e'};  String s3 = new String(chars); System.out.println(s3);//abcde System.out.println(chars[0]);//a  5、String(char[] value, int offset, int count)：  将char数组赋值,从数组索引值为offset的字符开始，取count个字符组成一个新的字符串，这并不影响原char数组  String s4 = new String(chars, 2, 2);  System.out.println(s4);//cd /\*6、String(int[] codePoints, int offset, int count)  复制一份数组，按ASCII表将int数组进行解码，然后从数组索引值为offset的元素开始，取count个元素组成新的字符串。\*/  int[] ints = {48, 49, 50, 51, 52, 53, 54, 55};  String s5 = new String(ints, 2, 3);  System.out.println(s5); /\*7、String(String original)  创建一个字符串的副本，基本不需要使用\*/  String s6 = new String(s2);  String s7 = new String("aaa"); |

### 2、获取字符串

|  |
| --- |
| int length() 获取字符串的长度  char charAt(int index) 获取特定位置的字符 (角标越界)  int indexOf(String str) 查找子串第一次出现的索引值,如果子串没有出现 在字符串中，那么则返回-1表示。  int lastIndexOf(String str) 查找子串最后一次出现的索引值 , 如果子串没有出现 在字符串中，那么则返回-1表示 |
| String string = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz"; // 1、int length()：获取字符串的长度  int length = string.length();  System.out.println("字符串string的长度是：" + length);  // 2、char charAt(int index)：获取特定位置的字符（注意角标越界异常）  char c = string.charAt(5);  System.out.println("字符串str中角标为5的字符是：" + c);  // 3、int indexOf(String str)：查找子字符串第一次出现的索引值，如果没有，则返回-1  int hijk = string.indexOf("hijk");  System.out.println("字符串string中第一次出现子字符串hijk的位置是：" + hijk);  // 4、int lastIndexOf(String str)：查找子字符串最后一次出现的索引值，如果没有，返回-1  int hijk1 = string.lastIndexOf("hijk");  System.out.println("字符串string中最后一次出现子字符串hijk的位置是：" + hijk1); |

### 3、判断方法

|  |
| --- |
| 1、boolean endsWith(String str) 是否以指定字符串结束  2、boolean isEmpty()是否长度为0 如：“” null V1.6  3、boolean contains(CharSequences) 是否包含指定序列 应用：搜索  4、boolean equals(Object anObject) 两个字符串的内容是否相等  5、boolean equalsIgnoreCase(String anotherString) 忽略大小写后两个字符串的内容是否相等 |
| String string1 = "abcdefghijklmnopqrstuvwxyz";  String string2 = " ";  String string3 = null;  String string4 = "abCdeFgHijklmnOpqrSTuvwxyz";  // 1、boolean endsWith(String str) 是否以指定字符结束  System.out.println("字符串是否以 xyz 结尾？：" + string1.endsWith("xyz"));//true  System.out.println("字符串是否以 xyz 结尾？：" + string2.endsWith("xyz"));//false  // 2、boolean isEmpty()是否长度为0 如：“” null V1.6  System.out.println("字符串长度是否为0？：" + string1.isEmpty());//false  System.out.println("字符串长度是否为0？：" + string2.isEmpty());//false  // string3==null,该处发生空指针异常NullPointerException  // System.out.println("字符串长度是否为0？：" + string3.isEmpty());  // 3、boolean contains(CharSequences) 是否包含指定序列 应用：搜索  System.out.println("字符串是否包含指定的字符序列？：" + string1.contains("defg"));//true  System.out.println("字符串是否包含指定的字符序列？：" + string2.contains(" "));//true  // 4、boolean equals(Object anObject) 两个字符串的内容是否相等  System.out.println("比较两个字符串的内容是否相同？：" + string1.equals(string4));//false   // 5、boolean equalsIgnoreCase(String anotherString) 忽略大小写是否相等  System.out.println("忽略大小写后比较两个字符串的内容是否相同？：" + string1.equalsIgnoreCase(string4));//true |

### 4、转换方法

|  |
| --- |
| 1、char[] toCharArray() 将字符串转换为字符数组  2、byte[] getBytes() 将字符串转换成字节数组 |
| String str = "Dome3.java"; char[] chars = str.toCharArray(); System.out.println("字符串转换成字符(char[])数组:" + Arrays.toString(chars));  byte[] bytes = str.getBytes(); System.out.println("字符串转换成字节(byte[])数组" + Arrays.toString(bytes)); |

## 3、字符串注意事项

1、字符串是常量，一旦被创建，就不能被修改

2、字符串一旦值被修改，那么就会马上创建一个新的对象，旧的常量值将会被GC不定时的给回收。

## 4、String源码解读

1、 String对象的值实际上是一个final类型的字节数组（char[]）

|  |
| --- |
| private final char value[]; |

2、 String类的带参构造方法：重写了String的hashCode方法，使得内容相同的字符串有着相同的hashCode

|  |
| --- |
| public String(String original) {  this.value = original.value;  this.hash = original.hash; }  // 重写了String的hashCode()方法  public int hashCode() {  int h = hash;  if (h == 0 && value.length > 0) {  char val[] = value;   for (int i = 0; i < value.length; i++) {  h = 31 \* h + val[i];  }  hash = h;  }  return h; } |

3、String重写了Object的equals() 方法

|  |
| --- |
| public boolean equals(Object anObject) {  if (this == anObject) {  return true;  }  if (anObject instanceof String) {  String anotherString = (String)anObject;  int n = value.length;  if (n == anotherString.value.length) {  char v1[] = value;  char v2[] = anotherString.value;  int i = 0;  while (n-- != 0) {  if (v1[i] != v2[i])  return false;  i++;  }  return true;  }  }  return false; } |

## 5、StringBuffer

1、String字符串不适合频繁的改动，因为一旦改动就会创建新的对象，如果需要频繁的改变字符串的内容，建议使用字符串缓冲类StringBuffer

1. StringBuffer实际上就是一个存储字符的容器，这是一个字符串缓冲类

## 6、StringBuffer源代码解读

1、StringBuffer的空参构造方法：StringBuffer创建对象是默认的长度是16 。

|  |
| --- |
| // StringBuffer的构造器  public StringBuffer() {  super(16); }  // StringBuffer 的父类AbstractString Builder类的构造器  AbstractStringBuilder(int capacity) {  value = new char[capacity]; } |

1. StringBuffer的带参构造函数：StringBuffer创建对象是默认的长度是string.length() + 16 。

|  |
| --- |
| // StringBuffer的带参构造器  public StringBuffer(String str) {  super(str.length() + 16);  append(str); }  //AbstractStringBuilder  AbstractStringBuilder(int capacity) {  value = new char[capacity]; } |

2、当默认的长度不够时,增长多少倍?（2倍+2 个长度）

|  |
| --- |
| // StringBuffer 的append方法：重写了父类的append方法  @Override public synchronized StringBuffer append(String str) {  toStringCache = null; // 将原来的值置为空，返回最新改变后的StringBuffer值  super.append(str);  return this; }  // StringBuffer的父类 AbstractStringBuilder类的源代码  AbstractStringBuilder append(AbstractStringBuilder asb) {  if (asb == null)  return appendNull();  int len = asb.length();  ensureCapacityInternal(count + len);//确保内部容量  asb.getChars(0, len, value, count);  count += len;  return this; }  // 确保内部容量充足（判断StringBuffer容器是否能够容纳修改后的字符串长度）  private void ensureCapacityInternal(int minimumCapacity) {  *// overflow-conscious code* if (minimumCapacity - value.length > 0) // 如果新字符串长度 – 原来数组的长度 > 0 ;  expandCapacity(minimumCapacity); // 扩增容量 }  // 扩增容量的方法  void expandCapacity(int minimumCapacity) {  int newCapacity = value.length \* 2 + 2; // 扩增的容量是原来长度的 两倍+2个长度  if (newCapacity - minimumCapacity < 0)  newCapacity = minimumCapacity;  if (newCapacity < 0) {  if (minimumCapacity < 0) *// overflow* throw new OutOfMemoryError();  newCapacity = Integer.*MAX\_VALUE*;  }  value = Arrays.*copyOf*(value, newCapacity); } |

### 1、面试笔试题：使用StringBuffer无参构造器创建一个对象时，默认的初始容量值是多少？如果长度不够使用了，自动增长多少倍？

答：

StringBuffer的底层实现是 依赖了一个char类型的数组来存储字符数据的，默认的字符数组长度是16 。

当字符串发生改变，容量不够时，StringBuffer对象会调用父类的方法，使对象的值变为原数组长度的（2倍+2）个长度。

## 7、StringBuffer与StringBuilder的区别

1、相同点

* 两个都是字符串缓冲类
* 两个类的方法都是一致的

2、不同点

* + StringBuilder是线程不安全的，操作效率高；StringBuffer是线程安全的，操作效率低
  + StringBuffer是jdk1.0出现的；StringBuilder是jdk1.5出现的

# 三、System系统类

## 1、System系统类，主要获取系统的属性数据

## 2、System类的常用方法

（1）arraycopy(Object src, int srcPos, Object dest, int destPos, int length) 一般

src - 源数组。

srcPos - 源数组中的起始位置。

dest - 目标数组。

destPos - 目标数据中的起始位置。

length - 要复制的数组元素的数量。

（2）currentTimeMillis() 获取当前系统系统。 重点

（3）exit(int status) 退出jvm 如果参数是0表示正常退出jvm，非0表示异常退出jvm。 一般

（4）gc() 建议jvm赶快启动垃圾回收期回收垃圾。

（5）getenv(String name) 根据环境变量的名字获取环境变量。

（6）getProperty(key)

（7）finalize() 如果一个对象被垃圾回收器回收的时候，会先调用对象的finalize()方法。

|  |
| --- |
| **public static void** main(String[] args) {  */\* （1）arraycopy(Object src, int srcPos, Object dest, int destPos, int length) 复制数组（一般）  src - 源数组。  srcPos - 源数组中的起始位置。  dest - 目标数组。  destPos - 目标数据中的起始位置。  length - 要复制的数组元素的数量。   （2）currentTimeMillis() 获取当前系统系统。 （重点常用）  （3）exit(int status) 退出jvm 如果参数是0表示正常退出jvm，非0表示异常退出jvm。 （一般）   （4）gc() 建议jvm赶快启动垃圾回收期回收垃圾。  （5）getenv(String name) 根据环境变量的名字获取环境变量。  （6）getProperty(key):获取系统的属性  （7）finalize() 如果一个对象被垃圾回收器回收的时候，会先调用对象的finalize()方法。\*/      /\*（1）arraycopy(Object src, int srcPos, Object dest, int destPos, int length) 复制数组（一般）  src - 源数组。  srcPos - 源数组中的起始位置。  dest - 目标数组。  destPos - 目标数据中的起始位置。  length - 要复制的数组元素的数量。\*/* String[] str = {**"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**, **"g"**, **"h"**, **"i"**, **"j"**, **"k"**, **"l"**};  String[] destArr = **new** String[15];  System.*arraycopy*(str, 4, destArr, 4, 5);  System.***out***.println(Arrays.*toString*(destArr));   *//（2）currentTimeMillis() 获取当前系统系统。 （重点常用）* **long** l = System.*currentTimeMillis*();*//1572338694863（1700年1月1日据现在多少毫秒）   //（3）exit(int status) 退出jvm 如果参数是0表示正常退出jvm，非0表示异常退出jvm。 （一般）  //System.exit(0);* System.***out***.println(**"aaaaaaaaaaaaaaa"**);   *//（4）gc() 建议jvm赶快启动垃圾回收期回收垃圾。* **for** (**int** i = 0; i < 4; i++) {  **new** Student(**"狗娃"** + i + **"号"**);  System.*gc*();  }   */\*（5）getenv(String name) 根据环境变量的名字获取环境变量。\*/* String java\_home = System.*getenv*(**"JAVA\_HOME"**);  System.***out***.println(java\_home);   */\*（6）getProperty(key)\*/* Properties properties = System.*getProperties*();  properties.list(System.***out***);  } |

# 四、Runtime运行环境类

## 1、应用程序运行环境

## 2、常用方法

*1、getRuntime() 返回当前应用程序的运行环境对象。  
 2、exec(String command) 根据指定的路径执行对应的可执行文件。  
 3、freeMemory() 返回 Java 虚拟机中的空闲内存量。。 以字节为单位  
 4、maxMemory() 返回 Java 虚拟机试图使用的最大内存量。  
 5、totalMemory() 返回 Java 虚拟机中的内存总量*

|  |
| --- |
| */\** *1、getRuntime() 返回当前应用程序的运行环境对象。  2、exec(String command) 根据指定的路径执行对应的可执行文件。  3、freeMemory() 返回 Java 虚拟机中的空闲内存量。。 以字节为单位  4、maxMemory() 返回 Java 虚拟机试图使用的最大内存量。  5、totalMemory() 返回 Java 虚拟机中的内存总量\*/* **public static void** main(String[] args) **throws** IOException , InterruptedIOException {  *// 1、getRuntime() 返回当前应用程序的运行环境对象。* Runtime runtime = Runtime.*getRuntime*(); *// System.out.println(runtime.toString()); // Process exec = runtime.exec("E:\\aa\\A.java");* System.***out***.println(**"虚拟机中的空闲内存量："**+runtime.freeMemory());  System.***out***.println(**"虚拟机中试图使用的最大容量内存："**+runtime.maxMemory());  System.***out***.println(**"虚拟机中的内存总量："**+runtime.totalMemory());   } |

# 五、Date、Calendar日期类

## 1、Date类

* 使用DateFormat类来格式化日期，但是 jdk1.1之后，被**Calendar**类取代。

（1）构造方法：

* Date()：分配 Date 对象并初始化此对象，以表示分配它的时间（精确到毫秒）
* Date(long date)：分配 Date 对象并初始化此对象，以表示自从标准基准时间（称为“历元（epoch）”，即 1970 年 1 月 1 日 00:00:00 GMT）以来的指定毫秒数。

（2）普通方法：

* boolean after(Date when)：测试此日期是否在指定日期之后。
* boolean before(Date when)：测试此日期是否在指定日期之前。
* int compareTo(Date anotherDate)：比较两个日期的顺序
* boolean equals(Object obj)：比较两个日期的相等性。
* long getTime()：返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。
* void setTime(long time)：设置此 Date 对象，以表示 1970 年 1 月 1 日 00:00:00 GMT 以后 time 毫秒的时间点。

|  |
| --- |
| Date date1 = **new** Date(); Date date2 = **new** Date(); Date date3 = **new** Date(); Date date4 = **new** Date();  date1.setTime(122343244); date2.setTime(124566666); date4.setTime(124566666); date2.setTime(124543566);  **long** time1 = date1.getTime(); System.***out***.println(time1);  *// 测试此日期是否在指定日期之后* System.***out***.println(**"测试date1是否在date2之后:"** + date1.after(date2)); System.***out***.println(**"测试date2是否在date3之前:"** + date1.after(date2)); System.***out***.println(**"date1与date2的的先后顺序："** + date1.compareTo(date3)); System.***out***.println(**"date4与date2的相等性："** + date2.equals(date4)); |

## 2、SimpleDateFormat类

* 作用1：可以把日期转换成指定格式的日期 format()
* 作用2：可以把一个字符转换成指定的日期 parse()

|  |
| --- |
| *//创建日期格式化对象,并指定日期格式* SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**); Date date = **new** Date();*//获取当前系统时间* String format = simpleDateFormat.format(date);*// 将Date系统时间转换成指定格式的日期格式* System.***out***.println(format);*//2019-10-30 17:16:10*   * String birthday = **"1992-09-10 17:30:59"**;*//与SimpleDateFormat初始化日期格式相同的日期字符串* Date parse = simpleDateFormat.parse(birthday);*//birthday 的格式必须与SimpleDateFormat格式完全一致* System.***out***.println(parse);*//Thu Sep 10 17:30:59 CST 1992* |

|  |
| --- |
| **1、format()**  Date date = **new** Date();*//获取当前系统时间* System.***out***.println(**"当前系统时间是："** + date);*//Wed Oct 30 17:37:59 CST 2019 //初始化日期格式* SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**); *//将系统时间转换成指定格式的字符串* String format = simpleDateFormat.format(date); System.***out***.println(**"转换后的系统时间是："** + format);*//2019-10-30 17:37:59*  **2、parse()** *//初始化系统时间* SimpleDateFormat sdf = **new** SimpleDateFormat(**"yyyy年MM月dd日 HH时mm分ss秒"**); *//与指定格式相同格式的字符串* String birthday = **"1992年09月10日 17时30分59秒"**; *//将字符串转换成系统格式的时间* Date parse = sdf.parse(birthday); System.***out***.println(**"转换后的系统时间是："** + parse);*//Thu Sep 10 17:30:59 CST 1992* |

## 3、Calendar日期类

1、Calendar.getInstance(): 获取Calendar对象

2、calendar.get(Calendar.参数值): 获取当前系统时间（年月日时分秒）

* + 参数值：
    - Calendar.YEAR：年
    - Calendar.MONTH：月（0-11 代表中国 1-12月份，所以正确月份应该 +1 再表示）
    - Calendar.DATE：一月中的某一天
    - HOUR\_OF\_DAY：一天中的某个小时（HOUR 十二进制 HOUR\_OF\_DAY二十四进制）
    - MINUTE：一小时中的某分
    - SECOND：一分中的某一秒
    - MILLISECOND：毫秒
    - DAY\_OF\_WEEK：星期几（外国的星期日是一周中的第一天，中国需 -1表示）

|  |
| --- |
| *//1、Calendar.getInstance(): 获取Calendar对象* Calendar calendar = Calendar.*getInstance*(); *// 2、calendar.get(..): 获取当前系统时间（年月日时分秒）* System.***out***.println(calendar.get(Calendar.***YEAR***) + **"年"** + (calendar.get(Calendar.***MONTH***) + 1) + **"月"** *//月份需+1才正确* + calendar.get(Calendar.***DATE***) + **"日 "** + calendar.get(Calendar.***HOUR\_OF\_DAY***) + **"时"** *//HOUR 十二进制/HOUR\_OF\_DAY二十四进制* + calendar.get(Calendar.***MINUTE***) + **"分"***//MINUTE 分* + calendar.get(Calendar.***SECOND***) + **"秒"***//SECOND 秒* + calendar.get(Calendar.***MILLISECOND***) + **"毫秒 "***//毫秒、* + **"星期"** + (calendar.get(Calendar.***DAY\_OF\_WEEK***) - 1) *//星期几（外国的星期日是一周中的第一天，中国需 -1表示）* );*//2019年10月30日 17时16分10秒509毫秒 星期3* |

# 六、Math数学类

1、abs()：绝对值  
2、ceil()：向上取整  
3、floor()：向下取整  
4、round()：四舍五入  
5、max();返回比较大的数  
6、min()：返回比较小的数  
5、random():返回0-1之间的随机数（包括0不包括1,取小数点后16位，最高可达17位）

|  |
| --- |
| System.***out***.println(**"返回绝对值："** + Math.*abs*(-12));*//12* System.***out***.println(**"向上取整："** + Math.*ceil*(-3.2));*//-3.0* System.***out***.println(**"向下取整："** + Math.*floor*(-12.3));*//-13.0* System.***out***.println(**"四舍五入："** + Math.*round*(-12.8));*//-13* System.***out***.println(**"返回 12 与 14 中比较大的数："** + Math.*max*(12, 14));*//14* System.***out***.println(**"返回-12 与 -14 中比较小的数："** + Math.*min*(-12, -14));*//-14* System.***out***.println(**"返回random随机小数："** + Math.*random*());*//0.7655888957536922* System.***out***.println(**"返回random0-10之间的随机整数："** + Math.*random*() \* 10);*//4.53474292007628* |

# 七、Random随机数类

1、用来产生一个随机数对象，与Math.random() 方法相同

2、验证码：用Random类来产生一个随机的验证码

|  |
| --- |
| **public static void** main(String[] args) {  String s = *VerificationCode*();  System.***out***.println(**"验证码内容是："** + s); } *// 产生四位数的二维码* **public static** String VerificationCode() {  *// 字符数组，二维码从数组中取出* **char**[] chars = {**'中'**, **'华'**, **'人'**, **'民'**, **'共'**, **'和'**, **'国'**,**'7'**, **'0'**, **'周'**, **'年'**, **'生'**  , **'日'**, **'快'**, **'乐'**, **'1'**,**'2'**, **'3'**, **'4'**, **'5'**, **'6'**, **'8'**, **'9'**};  *//随机数对象* Random random = **new** Random();  *// 创建一个数组，记录char数组被取出元素的下标* **int**[] destArr = **new int**[4];  *//字符串缓冲类* StringBuilder stringBuilder = **new** StringBuilder();  *// 计数器，表示取出了几个数组* **int** count = 0;  **while** (count < 4) {  *// 标记 默认不重复* **boolean** flag = **false**;  *//(char数组的下标) 随机数不能大于数组长度* **int** index = random.nextInt(chars.**length**);   *//遍历destArr数组* **for** (**int** i = 0; i < destArr.**length**; i++) {  *// 如果char数组的下标与destArr数组的元素相等，说明取出过该元素* **if** (destArr[i] == index) {  flag = **true**;  **break**;  }  }  *// flag==false 说明没有取出过该元素* **if** (flag == **false**) {  *// 将字符放入stringBuilder字符串缓冲区中* stringBuilder.append(chars[index]);  *// 将char数组的下标放入destArr数组中* destArr[count] = index;  *// 取出元素数* count++;  }  }  **return** stringBuilder.toString(); } |