

65从零开始学Java65之其他几个常用类

前言

配套开源项目资料

一. System类

1. 简介

1.1 常用静态变量

1.2 常用静态方法

2. 静态变量的使用

2.1 out

2.2 in

2.3 err

3. 静态方法的使用

3.1 currentTimeMillis()方法

3.2 exit()方法

3.3 gc()方法

3.4 getProperty()方法

3.5 arraycopy()

二. Random随机类

1. 简介

2. 常用API方法

3. 基本用法

4. 配套视频

三. SecureRandom类

1. 简介

2. 基本使用

四. 结语

作者：孙玉昌，昵称【**一一哥**】，另外【**壹壹哥**】也是我哦

千锋教育高级教研员、CSDN博客专家、万粉博主、阿里云专家博主、掘金优质作者

前言

我们在解决实际问题时，除了经常需要对数字、日期、时间进行操作之外，有时候还需要对系统进行设置，另外还需要生成一些随机数字。那么我们又该如何实现这些需求呢？接下来壹哥会带着大家来学习一下Java中的另几个常用类，包括System、Random、SecureRandom等。

-----前戏已做完，精彩即开始-----

全文大约【**4000**】字，不说废话，只讲可以让你学到技术、明白原理的纯干货！本文带有丰富的案例及配图视频，让你更好地理解 and 运用文中的技术概念，并可以给你带来具有足够启迪的思考.....

配套开源项目资料

Github:

[GitHub – SunLtd/LearnJava](#)

Gitee:

[一一哥/从零开始学Java](#)

一. System类

1. 简介

System类位于java.lang包中，**代表着当前Java程序的运行平台，系统级的很多属性和控制方法都放在该类中**。由于该类的构造方法是private的，所以我们无法直接通过new的方式来创建该类的对象。System类提供了一些静态变量和静态方法，允许我们直接通过System类来调用这些类变量和类方法。在System类中，虽然有挺多的静态变量和方法，但对我们来说，只需记住一些常用的即可。

1.1 常用静态变量

System类中常用的静态变量有如下几个：

- `PrintStream out`：标准输出流；
- `InputStream in`：标准输入流；
- `PrintStream err`：标准错误输出流；

1.2 常用静态方法

System类中常用的静态方法有如下几个：

- `currentTimeMillis()`：返回当前的计算机时间；
- `exit()`：终止当前正在运行的 Java 虚拟机；
- `gc()`：请求系统进行垃圾回收，完成内存中的垃圾清除；
- `getProperty()`：获得系统中属性名为 `key` 的属性对应的值；
- `arraycopy()`：进行数组复制，即从指定源数组中复制一个数组。

接下来壹哥就把以上这些静态变量和静态方法的基本使用，给大家简要介绍一下。

2. 静态变量的使用

首先我们来看看System类中几个常用静态变量该如何使用。

2.1 out

`out`静态变量属于`PrintStream`类型，是System类中的标准输出流，用于接收要输出的数据。`out`中的数据内容通常会输出到显示器，或用户指定的某个输出目标。其实对我们来说，`out`并不陌生，可以说在我们之前的案例中经常使用，尤其是`out`中的`print`方法，最近我们一直在使用。但我们要搞清楚，`print`属于`PrintStream`流的方法，并不是 `System`类的方法。

```
1 //输出字符串不换行
2 System.out.print("Hello World");
3
4 //输出字符串并换行
5 System.out.println("Hello World");
```

2.2 in

in静态变量属于InputStream类型，是System类中的标准输入流，用于接收输入的数据。in通常是对应着键盘的输入，或是用户指定的另一个输入源。我们在之前的案例中，也简单使用过in常量，但它没有out用的那么频繁。

```
1  import java.io.IOException;
2  import java.util.Scanner;
3
4  public class Demo01 {
5
6      public static void main(String[] args) {
7          //in的用法
8          //用法1: 配合Scanner,作为它的参数
9          System.out.println("请输入内容:");
10         Scanner scanner=new Scanner(System.in);
11         String content = scanner.next();
12         System.out.println("content="+content);
13
14         //用法2: 挨个读取输入的每个字符
15         System.out.println("请输入内容,按回车键结束输入:");
16         int c;
17         try {
18             //读取输入的每个字符
19             c = System.in.read();
20             // 判断输入的字符是不是回车键
21             while(c != '\r') {
22                 //输出字符
23                 System.out.print((char) c);
24                 c = System.in.read();
25             }
26         } catch(IOException e) {
27             //捕获异常，壹哥以后会给大家专门讲解异常类
28             System.out.println(e.toString());
29         }
30     }
31
32 }
```

上面的这个案例，System.in.read()语句可以读入一个字符，read()方法是InputStream类拥有的方法。变量c必须用 int 类型，而不能用char类型，否则可能会丢失精度而导致编译失败。另外上面的程序，如果输入的是汉字将不能正常输出。如果我们想正常输出汉字，需要把 System.in声明

为 `InputStreamReader` 类型的实例。比如 `InputStreamReader in=new InputStreamReader(System.in,"GB2312")`，此时就可以读入完整的Unicode码，才能显示正常的汉字。

2.3 err

`err`静态变量属于`PrintStream`类型，是`System`类中的标准错误输出流，用于接收要输出的数据。`err`中的数据内容通常会输出到显示器，或用户指定的某个输出目标。其用法与`System.out`一样，只是不需要我们提供参数就可以输出错误信息，也可以用来输出用户指定的其他信息，包括一些变量的值。

```
1 //err的用法
2 System.err.println();
3
4 //输出指定的内容
5 System.err.println("错误信息");
```

以上这几个静态变量都很简单，大家记住其用法即可。

3. 静态方法的使用

接下来壹哥再跟大家说说`System`类中的几个常用静态方法的用法。

3.1 currentTimeMillis()方法

`currentTimeMillis()`方法用于返回当前计算机的时间戳，时间格式是当前计算机的时间与GMT时间(格林尼治时间)，自1970年 1月 1日 0时 0分 0秒以来所经历的毫秒数，我们一般用它来测试程序的执行时间。通过调用`currentTimeMillis()`方法，我们可以获得一个长整型的数字，该数字是以差值表达的当前时间。其实`currentTimeMillis()`方法我们在之前的文章中已经详细讲解过，这里壹哥就不再细说了。

```
1 long time = System.currentTimeMillis();
```

3.2 exit()方法

exit()方法用于终止当前正在运行的Java虚拟机，也就是可以用于退出程序。该方法需要一个整型的status参数，0表示正常退出，非零表示异常退出。我们使用该方法，可以在图形界面编程中实现程序的退出功能。该方法的用法如下：

Java 复制代码

```
1 public class Demo01 {
2     public static void main(String[] args) {
3         //exit的用法
4         try {
5             //睡眠5秒
6             Thread.sleep(5000);
7             //5秒后正常退出程序
8             System.exit(0);
9         } catch (InterruptedException e) {
10             e.printStackTrace();
11         }
12     }
13 }
```

3.3 gc()方法

gc()方法用于请求对系统主动进行垃圾回收，完成内存中的垃圾清除。但系统是否会立刻回收这些垃圾，却取决于系统中垃圾回收算法的具体实现，以及系统执行时的具体情况。一般我们在开发时不会主动调用该方法，有时候调用了也未必有效果。

Java 复制代码

```
1 //主动进行垃圾回收
2 System.gc();
```

3.4 getProperty()方法

getProperty()方法可以根据指定的key，获得系统中对应的某些属性值，系统中常见的属性名及其属性如下表所示：

属性名	属性说明
java.version	Java 运行时环境版本
java.home	Java 安装目录

os.name	操作系统的名称
os.version	操作系统的版本
user.name	用户的账户名称
user.home	用户的主目录
user.dir	用户的当前工作目录

Java | 复制代码

```

1 public class Demo03 {
2
3     public static void main(String[] args) {
4         //getProperty的用法
5
6         //获取java版本
7         String version = System.getProperty("java.version");
8         System.out.println("Java版本:"+version);
9
10        //获取java安装目录
11        String home = System.getProperty("java.home");
12        System.out.println("Java目录:"+home);
13
14        //系统名称
15        String name = System.getProperty("os.name");
16        System.out.println("操作系统名称:"+name);
17
18        //用户名称
19        String user = System.getProperty("user.name");
20        System.out.println("当前用户名称:"+user);
21    }
22
23 }

```

3.5 arraycopy()

arraycopy()方法用于数组复制，可以从指定的源数组中复制出一个数组，复制会从指定的位置开始，到目标数组的指定位置结束。arraycopy()方法一般有5个参数，其中，src表示源数组，srcPos表示从源数组中复制的起始位置，dest表示目标数组，destPos表示要复制到的目标数组的起始位置，length表示复制的个数。

```
1 public class Demo04 {
2
3     public static void main(String[] args) {
4         //arraycopy的用法
5
6         //源数组
7         char[] srcArray = {'A','B','C','D'};
8         //目标数组
9         char[] destArray = {'1','2','3','4','5'};
10
11        //进行数组复制
12        System.arraycopy(srcArray,1,destArray,1,2);
13
14        System.out.println("遍历源数组:");
15        for(int i = 0;i < srcArray.length;i++) {
16            System.out.println("源数组中的每个元素:"+srcArray[i]);
17        }
18
19        System.out.println("遍历目标数组:");
20        for(int j = 0;j < destArray.length;j++) {
21            System.out.println("新数组中的每个元素:"+destArray[j]);
22        }
23    }
24
25 }
```

二. Random随机类

1. 简介

我们在开发时，除了操作一些固定的数字之外，有时候还要操作一些不确定的随机数。Java中给我们提供了两种生成指定范围内随机数的方法：

- 使用Random类：伪随机数类，用来创建伪随机数。所谓伪随机数，就是指我们只要给定一个初始的种子，产生的随机数序列是完全一样的；
- 调用Math类的random()方法：Math.random()内部其实是在调用Random类，它也是伪随机数，但我们无法指定种子。

Random类为我们提供了比较丰富的随机数生成方法，比如nextInt()、nextLong()、nextFloat()、nextDouble()等方法。这些方法可以产生boolean、int、long、float、byte数组以及double类型的随机数，这是它比random()方法更好的地方，random()方法只能产生0~1之间的double类型随机数。

而且Random类提供的所有方法，生成的随机数字都是均匀分布的，也就是说区间内部的数字生成的概率是均等的。Random类位于java.util包中，该类有如下两个常用的构造方法：

- Random()：默认利用当前系统的时间戳作为种子数，使用该种子数构造出Random对象。
- Random(long seed)：使用单个的long类型参数，创建一个新的随机数生成器。

2. 常用API方法

在Random类中，有如下一些常用的API方法供我们操作随机数：

方法	说明
boolean nextBoolean()	生成一个随机的 boolean 值，生成 true 和 false 的值概率相等
double nextDouble()	生成一个随机的 double 值，数值介于 [0,1.0)，含 0 而不包含 1.0
int nextInt()	生成一个随机的 int 值，该值介于 int 的区间，也就是 -231~231-1。如果需要生成指定区间的 int 值，则需要进行一定的数学变换
int nextInt(int n)	生成一个随机的 int 值，该值介于 [0,n)，包含 0 而不包含 n。如果想生成指定区间的 int 值，也需要进行一定的数学变换
void setSeed(long seed)	重新设置 Random 对象中的种子数。设置完种子数以后的 Random 对象和相同种子数使用 new 关键字创建出的 Random 对象相同
long nextLong()	返回一个随机长整型数字
boolean nextBoolean()	返回一个随机布尔型值
float nextFloat()	返回一个随机浮点型数字
double nextDouble()	返回一个随机双精度值

3. 基本用法

接下来壹哥通过一个案例，来给大家讲解一下上述方法该如何使用。

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

```
1  import java.util.Random;
2
3  /**
4   * @author 一一哥Sun
5   */
6  public class Demo07 {
7
8      public static void main(String[] args) {
9          // 随机类生成随机数
10         Random r = new Random();
11
12         // 生成[0,1.0]区间的小数
13         double d1 = r.nextDouble();
14         System.out.println("d1="+d1);
15
16         // 生成[0,10.0]区间的小数
17         double d2 = r.nextDouble() * 10;
18         System.out.println("d2="+d2);
19
20         // 生成[0,10]区间的整数
21         int i1 = r.nextInt(10);
22         System.out.println("i1="+i1);
23
24         // 生成[0,25)区间的整数
25         int i2 = r.nextInt(30) - 5;
26         System.out.println("i2="+i2);
27
28         // 生成一个随机长整型值
29         long l1 = r.nextLong();
30         System.out.println("l1="+l1);
31
32         // 生成一个随机布尔型值
33         boolean b1 = r.nextBoolean();
34         System.out.println("b1="+b1);
35
36         // 生成一个随机浮点型值
37         float f1 = r.nextFloat();
38         System.out.println("f1="+f1);
39     }
40
41 }
```

但是我们从上面的案例中，会发现每次生成的随机数可能都是不同的，并没有体现出伪随机数的特性，这是为什么呢？其实这是因为我们创建Random实例时，如果没有给定种子，默认是使用系统

的当前时间戳作为种子。因此每次运行时，种子都不同，所以得到的伪随机数序列就不同。如果我们在创建Random实例时指定一个固定的种子，就会得到完全确定的随机数序列。

4. 配套视频

与本节内容配套的视频链接如下：

<https://player.bilibili.com/player.html?bvid=BV1Ja411x7XB&p=136&page=136>

三. SecureRandom类

1. 简介

壹哥在前面给大家说过，Random是一种伪随机数类。这时候就有小伙伴问了，那有没有真随机数类呢？当然是有的！

SecureRandom就是一种真随机数！从原理来看，SecureRandom内部使用了RNG(Random Number Generator，随机数生成)算法，来生成一个不可预测的安全随机数。但在JDK的底层，实际上SecureRandom也有多种不同的具体实现。有的是使用**安全随机种子加上伪随机数算法**来生成安全的随机数，有的是使用真正的**随机数生成器**来生成随机数。实际使用时，我们可以优先获取高强度的安全随机数生成器；如果没有提供，再使用普通等级的安全随机数生成器。但不管哪种情况，我们都无法指定种子。

因为**这个种子是通过CPU的热噪声、读写磁盘的字节、网络流量等各种随机事件产生的“熵”**，所以这个种子理论上是不可能重复的。这也就保证了SecureRandom的安全性，所以最终生成的随机数就是安全的真随机数。

尤其是在密码学中，安全的随机数非常重要。如果我们使用不安全的伪随机数，所有加密体系都将被攻破。因此，为了保证系统的安全，我们尽量使用SecureRandom来产生安全的随机数。

2. 基本使用

SecureRandom给我们提供了nextBoolean()、nextBytes()、nextDouble()、nextFloat()、nextInt()等随机数生成方法，如下图所示：

● **nextBoolean()** : boolean - Random

● **nextBytes(byte[] bytes)** : void - SecureRandom

● **nextBytes(byte[] bytes, SecureRandomParameters params)** : void - SecureRandom

● **nextDouble()** : double - Random

● **nextDouble(double bound)** : double - RandomGenerator

● **nextDouble(double origin, double bound)** : double - RandomGenerator

● **nextExponential()** : double - RandomGenerator

● **nextFloat()** : float - Random

● **nextFloat(float bound)** : float - RandomGenerator

● **nextFloat(float origin, float bound)** : float - RandomGenerator

● **nextGaussian()** : double - Random

● **nextGaussian(double mean, double stddev)** : double - RandomGenerator

● **nextInt()** : int - Random

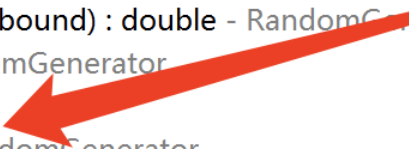
● **nextInt(int bound)** : int - Random

● **nextInt(int origin, int bound)** : int - RandomGenerator

● **nextLong()** : long - Random

● **nextLong(long bound)** : long - RandomGenerator

● **nextLong(long origin, long bound)** : long - RandomGenerator



千锋教育-孙玉昌

接下来我们就通过一个案例，来看看到底该如何生成一个安全的随机数。

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

千锋教育-孙玉昌

```
1  import java.security.NoSuchAlgorithmException;
2  import java.security.SecureRandom;
3  import java.util.Arrays;
4
5  public class Demo05 {
6
7      public static void main(String[] args) {
8          //SecureRandom真随机数的用法
9
10         SecureRandom sr = null;
11         try {
12             //获取高强度安全随机数生成器实例对象
13             sr = SecureRandom.getInstanceStrong();
14         } catch (NoSuchAlgorithmException e) {
15             //处理异常，获取普通的安全随机数生成器
16             sr = new SecureRandom();
17         }
18
19         //生成16个随机数
20         byte[] buffer = new byte[16];
21         //用安全随机数填充buffer
22         sr.nextBytes(buffer);
23         System.out.println("随机数="+Arrays.toString(buffer));
24
25         //生成100以内的随机整数
26         int nextInt = sr.nextInt(100);
27         System.out.println("随机数="+nextInt);
28     }
29
30 }
```

-----正片已结束，来根事后烟-----

四. 结语

至此，壹哥就把与系统类、伪随机数、真随机数等相关的类给大家介绍完了，这样我们就把开发时的一些常见类介绍完毕了。今天的重点内容是：

- System：代表着当前Java程序的运行平台，系统级的很多属性和控制方法都放在该类中；

- Random：生成伪随机数；
- SecureRandom：生成安全的真随机数。

如果你独自学习觉得有很多困难，可以加入壹哥的学习互助群，大家一起交流学习。