Java 生成随机数的 N 种方法

Hollis 1月29日

编译：ImportNew/覃佑桦
www.baeldung.com/java-generating-random-numbers

1.引言

本文将探讨用 Java 生成随机数的不同方法。

2.Java API

Java API 提供了几种随机数生成方法。让我们一起来了解一下。

2.1.java.lang.Math

Math 类中的 random 方法返回一个 [0.0, 1.0) 区间的 double 值。下面这段代码能得到一个 min 和 max 之间的随机数：

```
int randomWithMathRandom = (int) ((Math.random() * (max - min)) + min);
```

2.2.java.util.Random

Java 1.7 之前，最流行的随机数生成方法是 nextInt。这个方法提供了带参数和无参数两个版本。不带参数调用时，nextInt 会以近似相等概率返回任意 int 值，因此很可能会得到负数：

```
Random random = new Random();
int randomWithNextInt = random.nextInt();
```

如果调用 netxInt 时带上 bound 参数，将得到预期区间内的随机数：

```
int randomWintNextIntWithinARange = random.nextInt(max - min) + min;
```

上面的代码段能得到一个 [0, bound) 范围内的随机数。因此 bound 参数必须大于0。否则会抛出 java.lang.IllegalArgumentException 异常。

Java 8 引入了新的 ints 方法，返回一个java.util.stream.IntStream，让我们看看如何使用。

不带参数的 ints方法将返回一个无限 int 流：

```
IntStream unlimitedIntStream = random.ints();
```

调用时还可以指定参数来限制流大小：

```
IntStream limitedIntStream = random.ints(streamSize);
```

当然，也可以为生成数值设置最大值和最小值：

```
IntStream limitedIntStreamWithinARange = random.ints(streamSize, min, max);
```

2.3.java.util.concurrent.ThreadLocalRandom

Java 1.7 中ThreadLocalRandom类提供了一种新的更高效的随机数生成方法。与 Random 类相比有三个重要区别：

- 无需显式初始化 ThreadLocalRandom 实例。这样可以避免创建大量无用的实例，浪费垃圾收集器回收时间。
- 不能为 ThreadLocalRandom 设置随机种子（seed），这可能会导致问题。如果需要设置随机种子，应该避免采用这种方式生成随机数。
- Random 类在多线程时表现不佳。

下面让我们看看如何使用：

```
nWithThreadLocalRandomInARange = ThreadLocalRandom.current().nextInt(min, max);
```

Java 8 及更高版本提供了几种新方法。首先，nextInt 方法有两个变体：

```
int randomWithThreadLocalRandom = ThreadLocalRandom.current().nextInt();
int randomWithThreadLocalRandomFromZero = ThreadLocalRandom.current().nextInt(
```

其次，还可以使用 ints 方法：

```
IntStream streamWithThreadLocalRandom = ThreadLocalRandom.current().ints();
```

2.4.java.util.SplittableRandom

Java 8 还带来了一个快速随机数生成器SplittableRandom类。

正如 JavaDoc 中描述的那样，这是一个支持并行计算的随机数生成器。值得注意的是，这个实例非线程安全。使用该类时需要当心。

现在有了 nextInt 和 ints 方法。调用 nextInt 时，可以通过参数指定 top 和 bottom 范围：

```
SplittableRandom splittableRandom = new SplittableRandom();
int randomWithSplittableRandom = splittableRandom.nextInt(min, max);
```

这样可以检查 max 参数是否大于 min。检查失败会收到一个到IllegalArgumentException。**但是，这里不会进行正数或负数检查。因此，参数可以填写负数。**也可以在调用时使用一个参数或者不使用参数。工作方式与之前描述的相同。

这里也提供了ints方法。这意味着可以轻松得到 int流。可以选择流数据个数有限或无限。对于有限流，可以为数字生成范围设置 top 和 bottom：

```
IntStream limitedIntStreamWithinARangeWithSplittableRandom = splittableRandom.
```

2.5.java.security.SecureRandom

如果应用程序对安全敏感，则应考虑使用SecureRandom。这是一个强加密随机数生成器。实例默认构造函数不使用随机种子。因此，我们应该：

- 设置随机种子：随机种子不可预测
- 设置 java.util.secureRandomSeed 系统属性为 true。

该类继承自java.util.Random。现在，我们介绍了上面各种随机数生成方法。例如，如果需要获取任意 int 值，调用 nextInt 时可以不带参数：

```
SecureRandom secureRandom = new SecureRandom();
int randomWithSecureRandom = secureRandom.nextInt();
```

如果需要设置随机数生成范围，则可以在调用时带 bound 参数：

```
nt randomWithSecureRandomWithinARange = secureRandom.nextInt(max - min) + min;
```

切记，如果参数小于零会抛出 IllegalArgumentException。

3.第三方 API

上文的介绍中，Java 提供了许多随机数生成类和方法。然而，也有很多第三方 API 可以生成随机数。

下面对其中的一些进行介绍。

3.1.org.apache.commons.math3.random.RandomDataGenerator

Apache Commons 项目中 Commons 数学库提供了很多生成器。最简单，也可能最有用的是 RandomDataGenerator。它使用 Well19937c 算法生成随机数。我们也可以自行提供随机数算法。

让我们看看如何使用。首先，添加依赖项：

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-math3</artifactId>
    <version>3.6.1</version>
</dependency>
```

最新版本的 commons-math3 可以在 Maven Central 中找到。

然后开始使用：

```
RandomDataGenerator randomDataGenerator = new RandomDataGenerator();
int randomWithRandomDataGenerator = randomDataGenerator.nextInt(min, max);
```

3.2.it.unimi.dsi.util.XoRoShiRo128PlusRandom

这是最快的随机数生成器之一。由米兰大学信息科学系开发。

也可以在 Maven Central 仓库中找到。首先，添加依赖项：

```
<dependency>
    <groupId>it.unimi.dsi</groupId>
    <artifactId>dsiutils</artifactId>
    <version>2.6.0</version>
</dependency>
```

该生成器继承了 java.util.Random 类。但是，如果仔细看一下 JavaDoc，就会发现只有一种调用方法——通过 nextInt 生成随机数。最重要的是，该方法只提供无参和单个参数两种版本。其他任何调用都将直接调用 java.util.Random 方法。

例如，如果想得到某个范围内的随机数，可以这样写：

```
XoRoShiRo128PlusRandom xoroRandom = new XoRoShiRo128PlusRandom();
int randomWithXoRoShiRo128PlusRandom = xoroRandom.nextInt(max - min) + min;
```

4.总结

生成随机数有很多办法，但没有最佳方法。因此，应当根据需求选择最合适的方法。

本文的完整示例可以在 GitHub 上下载。
github.com/eugenp/tutorials/tree/master/java-numbers-3