Java生成随机不重复推广码邀请码

> 欢迎进入我的博客:blog.scarlettbai.com查看更多文章

最近接到一个需求，要批量生成推广码，首先我们知道推广码的特效有如下两点：

1：不可重复

2：不可以被推测出

关于这两点，我们的思路大体分为如下几类：

1:每次生成一个随机码后**查数据库是否有相同**的，有则重新生成(每次都要访问数据库，导致效率极低，不推荐)

2:依据**数据库的主键作为唯一键**，进行打乱或插入操作，如主键为8000001，取出后生成3位(据需求增减)随机数或字母，插入主键值中，构成如8000E0V0S1D这种串，可保证推广码不重复及不可推测出(这种方式也需要连接数据库取主键，当然可以一次性预生成所需数目的主键，然后生成对应推广码后更新进数据库，效率会高一些)

3:利用**算法**来保证值唯一，如UUID等，本篇重点介绍此种方式(不需连接数据库，算法选择合适则效率很高)

具体实现

上面说了，本文重点讨论利用算法实现不重复性，首先我们会想到最简单的方式：**UUID,**我们来看下具体效果：

```
public static void main(String[] args) {
    List<String> lists = new ArrayList<String>();
    Set<String> sets = new HashSet<String>();

    for (int i = 1 ; i < 100 ; i ++) {
        String encode = UUID.randomUUID().toString().replaceAll("-","").toUpperCase();
        lists.add(encode);
        sets.add(encode);
    }
```

```
ReedomCodeTest
D:\worksoftware\Java\jdk1.6.0_43\bin\java ...
0ABA7A7CAB3D485E9F28A8F060A0E1A3
0EFA32940A67489EA2731FC006F9EA54
106F6A7D1ED14C67B1D077E3A074CFC8
12EA66DE51894C6F85BA535F13AF99D1
157AC09D0B50470CA2E9B24112A899CB
16F85B68A997481AB71D7C4235AA6B1B
1EC20D8BFA3543DEA52A65B5B447318A
```

图中可以看出，**UUID生成的随机串去掉-后长度为32位**，虽然可以保证几乎绝对不会出现重复情况且支持多线程并发也不会重复，但是**长度太长**，不太实用。

既然UUID被淘汰了，那么接下来我们来看下我们最常接触的**SHA**算法：

```
24          long timestamp = System.currentTimeMillis();
25          for (int i = 1 ; i < 100 ; i ++) {
26              String value = Long.toHexString(timestamp+i);
27              value = EncrytUtils.SHA1(value).toUpperCase();
28              lists.add(value);
29              sets.add(value);
30          }
```

```
Run ReedomCodeTest
D:\worksoftware\Java\jdk1.6.0_43\bin\java ...
00989062689A12B30DB06E11F81B2B2492D0C5F5
```

```
074C96319780C364CCD8572983B9BA26615F252A
08C5D727D188BD01795C7C11F69055A56FDF2500
09F0932C615CB8EE03893F1D0EFE1BA6571F8512
0C852572B04DA13A4797CDD71AE1482DA6D36899
0F49D7D39342C69D69C608690DED55971CB2F1F0
11E92BD5947A141F62A51E9B1EF27532F035DFE2
19BCE93222B84EF41A051128E508BEA160B1EE47
1ABD5DFFA2DE79110E4525CBDD65E158FFFA9173
1F999062F9253C34CE8C333B3E6D9C867A500A97
213E6CE9765F96FDC07C2AACE284B970B979E287
265E98FD8E4CF35E6CE00ED0AA6717A414C28C6B
```

从图中可以看出，SHA算法生成的串也明显**太长，淘汰。**

接下来我想到了对称加密算法中的**RC4**，他可以保证密文长度和原字符串长度一样，关于RC4算法本篇不多描述，大家有兴趣可以上google百度一下，算法网上也有一大堆，大家可以随便下一个就好，此处**密文选为自增的数字，秘钥写死**，接下来我们看一下效果：

```
25          for (int i = 1 ; i < 100 ; i ++) {
26              String value = "";
27              value = RC4(String.format("%07d" , i) , "WHITE").toUpperCase();
28              lists.add(value);
29              sets.add(value);
30          }
31
Run  ReedomCodeTest
D:\worksoftware\Java\jdk1.6.0_43\bin\java ...
ÈÜÈ ÍÐ@
ÈÜÈ ÍÐA
ÈÜÈ ÍÐB
ÈÜÈ ÍÐC
ÈÜÈ ÍÐD
ÈÜÈ ÍÐE
```

图中可以看出，生成的码**不在字母及数字范围内**，不可能拿这些码给用户作为推广码，那么最简单的一个方式是，将这个码转换为16进制，效果如下：

```
25          for (int i = 1 ; i < 100 ; i ++) {
26              String value = "";
27              value = RC4.RC4_string_tohex(String.format("%07d" , i) , "WHITE").toUpperCase()
28              lists.add(value);
29              sets.add(value);
30          }
31
Run  ReedomCodeTest
D:\worksoftware\Java\jdk1.6.0_43\bin\java ...
C8DCC814EDD040
C8DCC814EDD041
C8DCC814EDD042
C8DCC814EDD043
C8DCC814EDD044
C8DCC814EDD045
```

大家可以看到，此时**密文都成了我们所熟知的字母和数字，但是长度变为了原字符长度的2倍**，不过依旧算可以接受，但是看加密后的密文全都是连续性的，不满足推广码第二点的可推测，而且由于私钥全都一样容易被破解，此处我们不需求解密，所以可以直接把**私钥换成UUID**，来看下效果：

```
25          for (int i = 1 ; i < 100 ; i ++) {
26              String value = "";
27              value = RC4.RC4_string_tohex(String.format("%07d" , i) , UUID.randomUUID().toString()).toUpperCase()
28              lists.add(value);
```

```
29              sets.add(value);
30          }
31
Run  ReedomCodeTest
D:\worksoftware\Java\jdk1.6.0_43\bin\java ...
00417C5D430190
009AF88FFBFF91
00F11F6FE148CC
014558C4523172
0302E7F6EC900B
031994D80A3CE0
0383F99A491F84
04AA36258B3A66
```

可以看到，此处生成的密文不再可以推测出，满足了我们的需求，那么对于是否会重复，我们再来做个测试，此处选用的**需加密字符串长度为7位**：

```
25          for (int i = 1 ; i < 10000000 ; i ++) {
26              String value = "";
27              value = RC4.RC4_string_tohex(String.format("%07d" , i) , UUID.randomUUID().toString()).toUpperCase();
28              lists.add(value);
29              sets.add(value);
30          }
31
Run  ReedomCodeTest
D:\worksoftware\Java\jdk1.6.0_43\bin\java ...
lists.size=9999999
sets.size=9999999
```

从图中可以看出，**字符串长度为7位时，生成了九百多万个密文，都没有重复的**，基本可以满足绝大部分情况，更多的我也没有进行测试，大家有兴趣可以测试下生成多少位时会出现重复。此种做法的好处是，**原字符串长度越长，生成的不会重复的密文量级则会更大**，大家可以按需调节。

不过这种方式的唯一不足在于，**生成的密文长度是偶数的**，如果大家需要确定的奇数长度密文，可以把**RC4加密结果用别的方式转换**为我们常用的字母和数字，楼主本处是没有特殊需求，所以直接采用了转换16进制这种快捷偷懒的方式。

欢迎关注个人微信公众号：读书健身编程