链滴

优惠码高效编码方案

作者：hzchendou

原文链接：https://ld246.com/article/1634263955587

来源网站：链滴

需求简介

当前各个业务系统，只要涉及到产品销售，就离不开大大小小的运营活动需求，其中最普遍的就是兑码需求，无论是线下活动或者是线上活动，都能起到良好的宣传效果。(这里使用兑换码指代优惠码,用范围更广)

兑换码：由一系列字符组成，每一个兑换码对应系统中的一组信息，可以是优惠信息（优惠券），也以是相关奖品信息。

在实际的运营活动中，要求每一个兑换码都具有唯一性，一个兑换码对应一个优惠信息，而且需求量往比较大（这就是通常说的客户可以不用,但你必须给
oy ），兑换码信息要尽可能的简洁，并且能对这些兑换码信息进行管理（查看兑换码，失效兑换码统计兑换码的使用情况）

兑换码特点

通过上述分析，能够发现兑换码有以下几个特点：

- 兑换码具有唯一性
- 兑换码尽可能简洁
- 兑换码的量级大

并且由于营运活动的特殊性，要求兑换码能够提前生成，这样可以尽可能的为活动进行预热（-_-!!!）兑换码需要具有**唯一性**，那么每一个兑换码必须是不同的；并且要求**简洁**，那么兑换码的长度不能太；**量级大**，之前也提到兑换码属于广撒网的策略，所以利用率低，如果要落地存储的话, 会浪费大量间.

那么就需要设计一种有效的兑换码生成策略，支持预先生成，支持校验，内容简洁，生成的兑换码都有唯一性，那么这种策略就是一种特殊的编解码策略，按照约定的编解码规则支撑上述需求。

设计思路

既然是一种编解码规则，那么需要约定编码空间(也就是用户看到的组成兑换码的字符)，编码空间由符a-z,A-Z,数字0-9组成，为了增强兑换码的可识别度，剔除大写字母O以及I,可用字符如下所示，共6个字符：

abcdefghijklmnopqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXZY0123456789

之前说过，兑换码要求近可能简洁，那么设计时就需要考虑兑换码的字符数，假设上限为12位，而字空间有60位，那么可以表示的空间范围为60^12=1306069401600000000000000(也就是可以12位兑换码可以生成天量,应该够运营同学挥霍了)，转换成2进制：100100010000000010111001100110101110011000000000000000000000000(61位)

- 兑换码组成成分分析

兑换码可以预先生成，并且不需要额外的存储空间保存这些信息，每一个优惠方案都有独立的一组兑码(指运营同学组织的每一场运营活动都有不同的兑换码,不能混合使用, 例如双11兑换码不能使用在双2活动上)，每个兑换码有自己的编号，防止重复，为了保证兑换码的有效性，对兑换码的数据需要进校验，当前兑换码的数据组成如下所示(是不是和印象中的TCP)：

优惠方案id + 兑换码序列号 + 校验码

编码方案

* 兑换码序列号i，代表当前兑换码是当前活动中第i个兑换码，兑换码序列号的空间范围决定了优惠动可以发行的兑换码数目，当前采用30位bit位表示，可表示范围：1073741824（10亿优惠券）
* 优惠方案id,代表当前优惠方案的id号，优惠方案的空间范围决定了可以组织的优惠活动次数，当前用15位表示，可以表示范围：32768（考虑到运营活动的频率，以及id的初始值10000，15位足够，35天每天有运营活动，可以使用54年）
* 校验位，校验兑换码是否有效，主要为了快捷的校验兑换码信息的是否正确，其次可以起到填充数的目的，增强数据的散列性，使用13位表示校验位,其中分为两部分，前6位和后7位

兑换码编号生成算法

* 生成算法

```
public static long enRedeemNum(long couponSchemeId, long redeemSerialNum) {
    redeemSerialNum = redeemSerialNum << REDEEM_SERIAL_NUM_LS;
    long r = couponSchemeId | redeemSerialNum;
    long n = numOfOne(r);
    long re = r % DIVISOR;
    r = (r << NUMBER_OF_ONE_LS) | n;
    r = (r << REMAINDER_LS) | re;
    return r;
}
```

编码算法

```
public static long [] deRedeemNum(long redeemNum) {
    long couponSchemeId = redeemNum & COUPON_SCHEME_ID_MASK;
    long redeemSerialNum = redeemNum & REDEEM_SERIAL_NUM_MASK;
    couponSchemeId = couponSchemeId >> COUPON_SCHEME_ID_RS;
    redeemSerialNum = redeemSerialNum >> REDEEM_SERIAL_NUM_RS;
    return new long[] { couponSchemeId, redeemSerialNum };
}
```

校验算法

```
public static boolean checkVaild(long redeemNum) {
    if (redeemNum > 0) {
        long checkSum = redeemNum & REMAINDER_MASK;
        long n = (redeemNum & NUMBER_OF_ONE_MASK) >> NUMBER_OF_ONE_RS;
        long r = (redeemNum & SUM_MASK) >> SUM_RS;
        if (numOfOne(r) == n) {
            if (r % DIVISOR == checkSum) {
                return Boolean.TRUE;
            }
        }
    }
    return Boolean.FALSE;
}
```

兑换码编码到兑换码映射方式

当前可以生成唯一的兑换码编码信息，需要将此信息映射到对应的字符空间中，并且字符空间是可自义的，当前采用的进制换算的方式，将兑换码编码信息换算成指定进制的数，对于进制的每一位进行码例如：

```
十进制转换成二进制
    17
    17/2 = 8 ... 1
    8/2  = 4 ... 0
    4/2  = 2 ... 0
    2/2  = 1 ... 0
    1/2  = 0 ... 1
十进制转换成n进制类似
    ....
```

在得到的相应进制表示后，在将进制中的没一位映射到字符空间中（n表示字符空间的大小）

这个映射关系是可以自定义的

```
private static final char[] r = new char[] {
  'q', 'w', 'e', '8', 'a', 's', '2', 'd', 'z', 'x', '9', 'c', '7', 'p',
          '5', 'i', 'k', '3', 'm', 'j', 'u', 'f', 'r', '4', 'v', 'y', 'l', 't', 'n', '6', 'b', 'g', 'h'};
```

(字母大小写，数字混排)

n进制换算以及映射算法

```
public static String enRedeemCode(long redeemNum) {
    char[] buf = new char[32];
    int charPos = 32;
    while ((redeemNum / l) > 0) {
        int ind = (int) (redeemNum % l);
        buf[--charPos] = r[ind];
        redeemNum /= l;
    }
    buf[--charPos] = r[(int) (redeemNum % l)];
    String str = new String(buf, charPos, (32 - charPos));
    return str;
}
```

兑换码解码

```
public static long deRedeemCode(String redeemCode) {
    char chs[] = redeemCode.toCharArray();
    long res = 0L;
    for (int i = 0; i < chs.length; i++) {
        int ind = -1;
        for (int j = 0; j < l; j++) {
            if (chs[i] == r[j]) {
                ind = j;
                break;
            }
        }
        if (ind == -1) {
            return -1;
```

```
        }
        if (i > 0) {
        res = res * l + ind;
        } else {
            res = ind;
        }
    }
    return res;
}
```

- 计算速度
  - 生成10万个 花费59ms
  - 生成百万个，花费530ms
  - 生成一千万 花费12535ms

```
85sl4l
2ncu9r
vqdfce
t5p5bg
bnjz3t
wwxyea4
wa4bnyj
wzs8rci
wcjxigc
8t6ktp5
...
```