

银行卡号的编码规则及校验

作者: [zhaozhizheng](#)

原文链接: <https://ld246.com/article/1587096349771>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

作者: liyongzhi1992

来源: CSDN

原文: <https://blog.csdn.net/liyongzhi1992/article/details/82773993>

银行卡号的编码规则

银行卡号由最多19位数字组成。

前6位数字被称为发行者识别号码 (Issuer Identification Number, 缩写为IIN), 也称为发卡行识别码 (Bank Identification Number, 简称BIN), 常说的卡BIN就是指它, 由中国银联代国内各发卡机构统一向ISO申请。6位IIN的第一位, 是主要行业标识符 (Major industry identifier, 缩写为MII) 分配如下:

0 - ISO/TC 68和其他行业分配

1 - 航空业

2 - 航空业, 金融业和其他未来行业分配

3 - 旅游业和娱乐业

4 - 银行业和金融业

5 - 银行业和金融业

6 - 商业和银行业/金融业

7 - 石油业和其他未来行业分配

8 - 医疗业, 电信业和其他未来行业分配

9 - 由各国标准团体分配

中间的7~18位由发卡行自定义, 表示不同的个人账户号码, 最大12位;

最后1位是校验码, 使用Luhn算法计算。

需要注意: 2017年发布的ISO/IEC 7812-1中, 删除了对MII的定义描述, 并将IIN码由6位扩展到了8, 但是由于总位数仍然最多19位, 所以中间的个人账户号码对应的最大位数由12位减少至10位。所卡BIN不再只是6位, 也需要考虑兼容8位。

Luhn算法

Luhn算法, 也称为“模10”算法, 是一种简单的校验和 (Checksum) 算法, 一般用于验证身份识别码, 例如信用卡号码、国际移动设备识别码 (International Mobile Equipment Identity, 缩写为MEI), 美国供应商识别码, 加拿大社会保险号码, 以色列身份证号码, 希腊社会安全号码等。

Luhn算法在ISO/IEC 7812-1中定义, 它不是一种安全的加密哈希函数, 设计它的目的只是防止意外错而不是恶意攻击, 即我们常说的防君子不防小人。

使用Luhn算法校验的步骤:

从右边第1个数字 (校验数字) 开始偶数位乘以2;

把步骤1种获得的乘积的各位数字与原号码中未乘2的各位数字相加;

如果步骤2得到的总和模10为0, 则校验通过。

举例说明:

笔者一张过期的信用卡号码为6225760008219524, 根据上述步骤进行校验。

序号	卡号	步骤1	步骤2	步骤3
----	----	-----	-----	-----

16 6 6 × 2 = 12 1 + 2 = 3
 15 2 2 + 2 = 5
 14 2 2 × 2 = 4 + 4 = 9
 13 5 5 + 5 = 14
 12 7 7 × 2 = 14 + 1 + 4 = 19
 11 6 6 + 6 = 25
 10 0 0 × 2 = 0 + 0 = 25
 9 0 0 + 0 = 25
 8 0 0 × 2 = 0 + 0 = 25
 7 8 8 + 8 = 33
 6 2 2 × 2 = 4 + 4 = 37
 5 1 1 + 1 = 38
 4 9 9 × 2 = 18 + 1 + 8 = 47
 3 5 5 + 5 = 52
 2 2 2 × 2 = 4 + 4 = 56
 1 4 4 + 4 = 60

60 % 10 = 0, 校验通过。大家可以据此尝试校验自己的银行卡号。

了解Luhn算法校验银行卡号的方法后，很容易推导出使用Luhn算法计算数字字符串的校验数字的方法，仍然以上述银行卡号为例，去掉校验数字后，剩余622576000821952，假设校验数字为，仍然根上述步骤计算。

序号	卡号	步骤1	步骤2	步骤3
16	6	6 × 2 = 12	1 + 2	= 3
15	2	2	+ 2	= 5
14	2	2 × 2 = 4	+ 4	= 9
13	5	5	+ 5	= 14
12	7	7 × 2 = 14	+ 1 + 4	= 19
11	6	6	+ 6	= 25
10	0	0 × 2 = 0	+ 0	= 25
9	0	0	+ 0	= 25
8	0	0 × 2 = 0	+ 0	= 25
7	8	8	+ 8	= 33
6	2	2 × 2 = 4	+ 4	= 37
5	1	1	+ 1	= 38
4	9	9 × 2 = 18	+ 1 + 8	= 47
3	5	5	+ 5	= 52
2	2	2 × 2 = 4	+ 4	= 56
1			+ = 56 +	

若要等式 $(56 +) \% 10 = 0$ 成立，很容易得到 $= 4$ ，与实际银行卡号相符。

代码实现如下：

```
/**
 * Luhn算法工具类
 *
 *
 *
 *
 *
 *
 *
 */
public class LuhnUtil {

    /**
     * 校验字符串
     *
     * <p>
     * 1. 从右边第1个数字（校验数字）开始偶数位乘以2；<br>
     * 2. 把在步骤1种获得的乘积的各位数字与原号码中未乘2的各位数字相加；<br>
     * 3. 如果在步骤2得到的总和模10为0，则校验通过。
     * </p>
     *
     * @param withCheckDigitString 含校验数字的字符串
     * @return true - 校验通过<br>
     *         false-校验不通过
     * @throws IllegalArgumentException 如果字符串为空或不是8~19位的数字
     */
    public static boolean checkString(String withCheckDigitString) {
        if (withCheckDigitString == null) {
            throw new IllegalArgumentException();
        }
        // 6位IIN+最多12位自定义数字+1位校验数字
        // 注意ISO/IEC 7812-1:2017中重新定义8位IIN+最多10位自定义数字+1位校验数字
        // 这里为了兼容2017之前的版本，使用8~19位数字校验
        if (!withCheckDigitString.matches("^\\d{8,19}$")) {
            throw new IllegalArgumentException();
        }
        return sum(withCheckDigitString) % 10 == 0;
    }

    /**
     * 计算校验数字
     *
     * <p>
     * 1. 从右边第1个数字（校验数字）开始偶数位乘以2；<br>
     * 2. 把在步骤1种获得的乘积的各位数字与原号码中未乘2的各位数字相加；<br>
```

```

* 3. 用10减去在步骤2得到的总和模10，得到校验数字。
* </p>
*
* @param withoutCheckDigitString 不含校验数字的字符串
* @return 校验数字
* @throws IllegalArgumentException 如果字符串为空或不是7~18位的数字
*/
public static int computeCheckDigit(String withoutCheckDigitString) {
    if (withoutCheckDigitString == null) {
        throw new IllegalArgumentException();
    }
    // 6位IIN+最多12位自定义数字
    // 注意ISO/IEC 7812-1:2017中重新定义8位IIN+最多10位自定义数字
    // 这里为了兼容2017之前的版本，使用7~18位数字校验
    if (!withoutCheckDigitString.matches("^\\d{7,18}$")) {
        throw new IllegalArgumentException();
    }
    // 因为是不含校验数字的字符串，为了统一sum方法，在后面补0，不会影响计算
    return 10 - sum(withoutCheckDigitString + "0") % 10;
}

/**
* 根据Luhn算法计算字符串各位数字之和
* <p>
* 1. 从右边第1个数字（校验数字）开始偶数位乘以2；<br>
* 2. 把在步骤1种获得的乘积的各位数字与原号码中未乘2的各位数字相加。<br>
* </p>
*
* @param str
* @return
*/
private static int sum(String str) {
    char[] strArray = str.toCharArray();
    int n = strArray.length;
    int sum = 0;
    for (int i = n; i >= 1; i--) {
        int a = strArray[n - i] - '0';
        // 偶数位乘以2
        if (i % 2 == 0) {
            a *= 2;
        }
        // 十位数和个位数相加，如果不是偶数位，不乘以2，则十位数为0
        sum = sum + a / 10 + a % 10;
    }
    return sum;
}
}

```

实际应用

银行卡信息查询

在金融行业软件系统中，银行卡号可谓是重要元素，但是，银行卡号本身承载的信息并不多，从上面述的内容来看，卡BIN是我们从银行卡号中获取的唯一信息，然而，卡BIN对于绝大多数人来说使用起来并不友好，因为映射关系并不容易获取。

当然，互联网上内容包罗万象，区区卡BIN何足挂齿，网站、论坛到处都可下载，不过很多人忽略了个问题，卡BIN是需要更新的，互联网上可以轻易下载到的卡BIN只是某个时间的快照，实效性难以证。

笔者还看到有人提及支付宝的查询接口（ccdcapi.alipay.com/validateAnd...），笔者使用之前的过卡号，接口返回cardType="CC"表示信用卡，bank="CMB"表示招商银行。支付宝作为国内最大的三方支付平台，其卡BIN的准确性与实效性理论上可以得到保证，但是笔者并不清楚其卡BIN来源，计是银行或银联。

数据从源头获取，准确性与实效性必然更胜一筹，笔者这里要推荐的卡BIN查询接口正是银联提供，联在其开放平台（open.unionpay.com/）提供了若干接口，其中就包括了银行卡信息查询（open.unionpay.com/tjweb/api/d...）接口，该接口可以根据银行卡卡号，返回发卡行、发卡行机构代码、卡质、卡类别、卡种、卡品牌、卡产品、卡等级、卡介质、所属总行机构中文名称、所属总行机构中文称。

接口返回的内容可以说是相当全面了，但是笔者开发项目中并未使用过该接口，因此不能担保其性能不过银联的权威性毋庸置疑，如需商业合作可以放心致电洽谈，银联的服务还是很专业的。

资费详情：本产品推广期为2017年10月1日至2018年9月30日，推广期内本产品可免费使用。推广期后的收费标准将提前30个工作日在本平台另行公示。

到目前为止，笔者没有看到收费标准公示，是否可以理解为仍然可以免费使用？

验证银行卡信息

银联开放平台（open.unionpay.com/）同时为我们提供了另外一个优秀的接口，验证银行卡信息（open.unionpay.com/tjweb/api/d...）。我们可以通过以下5种方式进行验证，灵活组合，按需选择：

两要素：银行卡号+姓名

两要素：银行卡号+身份证号

三要素：银行卡号+姓名+身份证号

四要素：银行卡号+姓名+身份证号+银行预留手机号

六要素：银行卡号+姓名+身份证号+银行预留手机号+CVN2+有效期

根据个人项目经验，推荐借记卡使用四要素验证，贷记卡使用六要素验证，毕竟验证要素越多，验证越可靠。

程序设计

有了Luhn算法以及银联开放平台提供的2个接口，我们应该如何设计我们的程序（产品）呢？

对输入要素进行初步校验

使用Luhn算法校验银行卡号，使用上篇身份证号码的编码规则及校验介绍的公式校验身份证号。

银行预留手机号的校验

这里需要注意的是：银行预留手机号，有时候你认为预留了，但是可能没有预留。如果客户坚持自己写信息无误，那么请客户致电银行查询预留手机号是解决问题的一个思路。

这里说的银行预留手机号的校验，主要是为了校验手机是否是客户本人身上，可以通过短信验证码的式校验，其必要性：一是，从安全性出发，手机的窃取往往比银行卡相关信息的窃取成本更高；二是

毕竟银联开放平台提供的2个接口后续使用过程中极有可能都是需要收费的，一条短信的价格往往更便宜，优先排除掉不合格的信息，从而提高后续验证的成功率。

调用银行卡信息查询接口

经过初步的信息校验通过，我们可以调用银联的第1个接口了，银行卡信息查询，银联返回的信息比全面，笔者的建议是全部记录下来，当系统中积累的银行卡数据足够多的时候，通过一定的数据分析段，恭喜你，你拥有了自己的卡BIN数据，每月更新一次，足矣。

从这个接口，可以看出银联的高明之处，它不直接告诉你卡BIN，让你每次都要调用它的接口查询，人以渔何以授人以鱼。（大家在制定接口时，要注意：从安全性和不可替代性出发，核心数据不能泄，尽量使用验证代替查询，这样也有利于提高接口使用率，增加用户粘性；同时，记录尽量多的请求数据，便于日后的数据分析，也符合当前大数据的趋势。）

调用这个接口，一方面可以积累数据，另一方面可以确定客户的银行卡性质，是借记卡还是贷记卡；记卡和贷记卡在进行验证时，验证要素略有差异，笔者开发项目过程中接触的银行卡验证，除了银行号、姓名、身份证号、手机号四要素之外，借记卡可以额外验证密码，贷记卡可以额外验证CVN2、效期，目前发行的芯片借记卡也有CVN2、有效期，但是笔者项目中从来没有对此验证过，不知银联个接口是否可以验证。

有时，为了提升客户体验，借记卡的密码，贷记卡的CVN2、有效期也不做验证。便利性与安全性是要互相妥协的，但是它们不是鱼与熊掌，通过技术手段完全可以鱼与熊掌兼得。

调用验证银行卡信息接口

最后，我们终于可以调用银联的第2个接口了。

数据安全存储

到这里，你以为结束了吗？其实还没有，还有一个重要的问题，那就是数据安全存储。

银行卡号，身份证号，手机号，建议加密存储；

借记卡密码，贷记卡CVN2、有效期，建议不存储；

数据安全存储，不单单指数据库，建议肉眼可见的介质之上都做安全存储，比如应用程序日志、数据步文件等；

数据传输过程中，理应也如此，银联接口中CVN2、有效期就要求RSA加密。