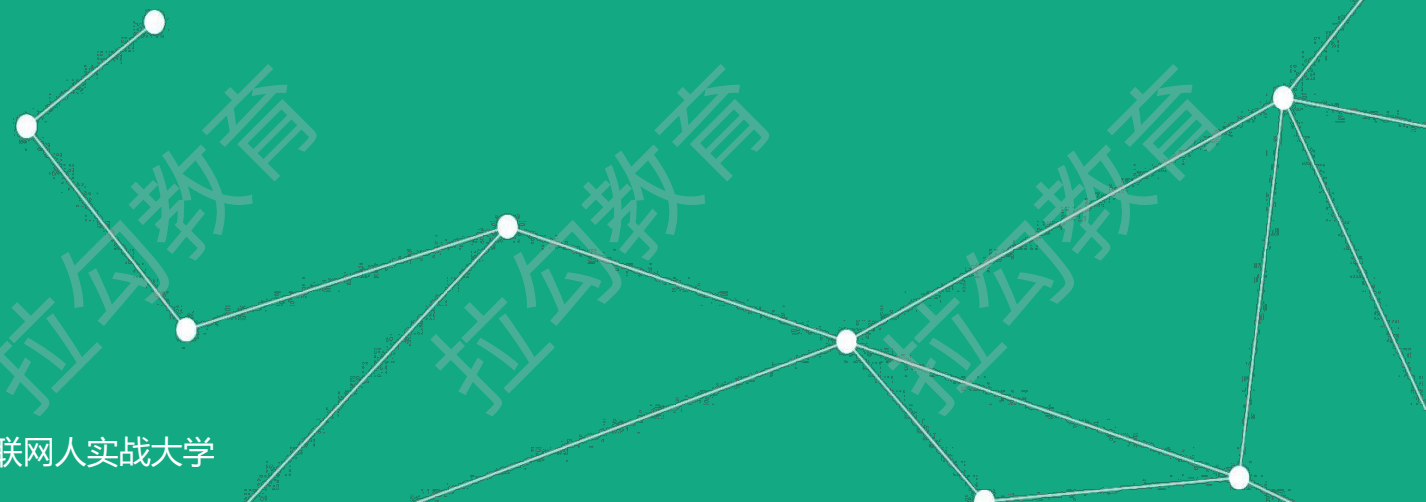
课时 05

哈希函数的本质及生成方式

1. 哈希表与哈希函数
2. hashCode 函数中的“魔数”（Magic Number）
3. 区块链挖矿的本质



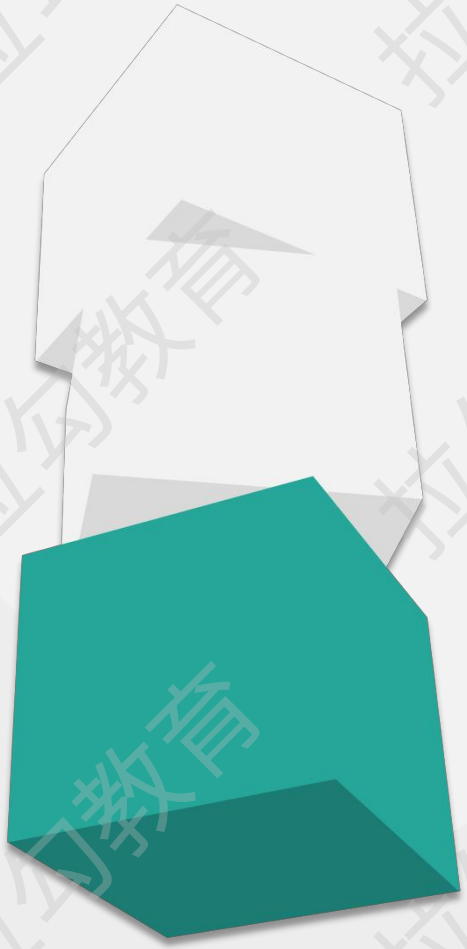
哈希表与哈希函数

哈希表本质上是一个数组，要访问一个数组中某个特定的元素，那么需要知道这个元素的索引

哈希函数的定义是将任意长度的一个对象映射到一个固定长度的值上

而这个值可以称作是哈希值（Hash Value）

哈希表与哈希函数



1. 任何对象作为哈希函数的输入都可以得到一个相应的哈希值

2. 两个相同的对象作为哈希函数的输入，它们总会得到一样的哈希值

3. 两个不同的对象作为哈希函数的输入，它们不一定会得到不同的哈希值

哈希表与哈希函数

按照 Java String 类里的哈希函数公式来计算出不同字符串的哈希值

$$s.charAt(0) * 31^{n-1} + s.charAt(1) * 31^{n-2} + ... + s.charAt(n-1)$$

下面所有字符的数值都是按照 ASCII 表获得，具体的数值可以在https://www.ascii-code.com/

当输入“ABC”字符串时，其哈希值为：

$$"ABC" = 'A' * 31^2 + 'B' * 31 + 'C' = 65 * 31^2 + 66 * 31 + 67 = 64578$$

哈希表与哈希函数

计算字符串 "Aa" 和 "BB" 的哈希值

```
"Aa" = 'A' * 31 + 'a' = 65 * 31 + 97 = 2112
```

```
"BB" = 'B' * 31 + 'B' = 66 * 31 + 66 = 2112
```

可以看到不同的两个字符串其实是会输出相同的哈希值出来的

这时候就会造成哈希碰撞

哈希表与哈希函数

在计算机里

一个 32 位 int 类型的整数里最高位如果是 0 则表示这个数是非负数，如果是 1 则表示是负数

如果当字符串通过计算算出的哈希值大于 2 的 32 次方减 1 时，也就是大于 32 位整数所能表达的最大正整数了，则会造成溢出，此时哈希值就变为负数了

hashCode 函数中的“魔数”（Magic Number）

Java String 类里的 hashCode 函数，一直在使用一个 31 这样的正整数来进行计算，这是为什么呢？

Java Openjdk-jdk11 中 String.java 的源码：

```
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        hash = h = isLatin1() ? StringLatin1.hashCode(value)
                              :
StringUTF16.hashCode(value);

    }
    return h;
}
```

hashCode 函数中的“魔数”（Magic Number）

StringLatin1 类中的 hashCode 函数：

```
public static int hashCode(byte[] value) {
        int h = 0;
        for (byte v : value) {
            h = 31 * h + (v & 0xff);
        }
        return h;
    }
```

hashCode 函数中的“魔数”（Magic Number）

StringUTF16类中的hashCode函数如下面所示：

```
public static int hashCode(byte[] value)
{
    int h = 0;
    int length = value.length >> 1;
    for (int i = 0; i < length; i++) {
        h = 31 * h + getChar(value, i);
    }
    return h;
}
```

hashCode 函数中的“魔数”（Magic Number）

拉勾 LAGOU.COM

Java String 类里的 hashCode 函数，一直在使用一个 31，为什么？

Goodrich 和 Tamassia 两位计算机科学家对超过 50000 个英文单词进行了哈希值运算，使用常数 31、33、37、39 和 41 作为乘数因子，每个常数算出的哈希值碰撞的次数都小于 7 个

数学的角度，选择一个质数作为乘数因子可以让哈希碰撞减少

其次，上面那两个 hashCode 源码，都各有一条 31 * h 的语句，可被自动优化成“(h << 5) - h”这样一条位运算加上一个减法指令，而不必执行乘法指令，可大大提高运算哈希函数的效率

区块链挖矿的本质

如果已知一个哈希值，即便给出了哈希函数的公式也很难算出原来的输入到底是什么?

$$s.charAt(0) * 31^{n-1} + s.charAt(1) * 31^{n-2} + ... + s.charAt(n-1)$$

根据这个公式，已知哈希值是 123456789，那输入的字符串是什么呢？

只能采用暴力破解法，也就是一个一个的字符串去尝试，直到尝试出这个哈希值为止

区块链挖矿的本质

对于区块链挖矿来说，这个“矿”其实就是一个字符串

“矿工”，也就是进行运算的计算机，必须在规定的时间内找到一个字符串，使得在进行了哈希函数运算之后得到一个满足要求的值

区块链挖矿的本质

以比特币为例，它采用了 SHA256 的哈希函数来进行运算，无论输入的是什么，SHA256 哈希函数的哈希值永远都会是一个 256 位的值。而比特币的奖励机制简单来说是通过每 10 分钟放出一个哈希值，让“矿工们”利用 SHA256(SHA256(x)) 这样两次的哈希运算，来找出满足一定规则的字符串出来

比如

比特币会要求找出通过上面 SHA256(SHA256(x)) 计算之后的哈希值，这个 256 位的哈希值中的前 50 位都必须为 0，谁先找到满足这个要求的输入值 x，就等于“挖矿”成功，给予奖励一个比特币

即便知道了哈希值，也很难算出这个 x 是什么，所以只能一个一个地去尝试。而市面上所说的挖矿机，其原理是希望能提高运算的速度，让“矿工”尽快地找到这个 x 出来