状态变量的存储模型(Layout of State Variables in Storage)

大小固定的变量（除了`映射`，`变长数组`以外的所有类型）在存储(storage)中是依次连续从位置`0`开始排列的。如果多个变量占用的大小少于32字节，会尽可能的打包到单个`storage`槽位里，具体规则如下：

- 在storage槽中第一项是按低位对齐存储（lower-order aligned）（译者注：意味着是大端序了，因为是按书写顺序。）。
- 基本类型存储时仅占用其实际需要的字节。
- 如果基本类型不能放入某个槽位余下的空间，它将被放入下一个槽位。
- `结构体`和`数组`总是使用一个全新的槽位，并占用整个槽(但在结构体内或数组内的每个项仍遵从上述规则)

优化建议

当使用的元素占用少于32字节，你的合约的gas使用也许更高。这是因为EVM每次操作32字节。因此，如果元素比这小，EVM需要更多操作来从32字节减少到需要的大小。

因为编译器会将多个元素打包到一个`storage`槽位，这样就可以将多次读或写组合进一次操作中，只有在这时，通过缩减变量大小来优化存储结构才有意义。当操作函数参数和`memory`的变量时，因为编译器不会这样优化，所以没有上述的意义。

最后，为了方便EVM进行优化，尝试有意识排序`storage`的变量和结构体的成员，从而让他们能打包得更紧密。比如，按这样的顺序定义，`uint128, uint128, uint256`，而不是`uint128, uint256, uint128`。因为后一种会占用三个槽位。

非固定大小

结构体和数组里的元素按它们给定的顺序存储。

由于它们不可预知的大小。`映射`和`变长数组`类型，使用`Keccak-256`哈希运算来找真正数据存储的起始位置。这些起始位置往往是完整的堆栈槽。

`映射`和`动态数组`根据上述规则在位置`p`占用一个未满的槽位(对`映射`里嵌套`映射`，数组中嵌套数组的情况则递归应用上述规则)。对一个动态数组，位置`p`这个槽位存储数组的元素个数(字节数组和字符串例外，见下文)。而对于`映射`，这个槽位没有填充任何数据(但这是必要的，因为两个挨着的`映射`将会得到不同的哈希值)。数组的原始数据位置是`keccak256(p)`；而`映射`类型的某个键`k`，它的数据存储位置则是`keccak256(k . p)`，其中的`.`表示连接符号。如果定位到的值以是一个非基本类型，则继续运用上述规则，是基于`keccak256(k . p)`的新的偏移`offset`。

`bytes`和`string`占用的字节大小如果足够小，会把其自身长度和原始数据存在当前的槽位。具体来说，如果数据最多31位长，高位存数据(左对齐)，低位存储长度`lenght * 2`。如果再长一点，主槽位就只存`lenght * 2 + 1`。原始数据按普通规则存储在`keccak256(slot)`

所以对于接下来的代码片段：

```
pragma solidity ^0.4.4;

contract C {
    struct s { uint a; uint b; }
    uint x;
    mapping(uint => mapping(uint => s)) data;
}
```

按上面的代码来看，结构体从位置0开始，这里定义了一个结构体，但并没有对应的结构体变量，故不占用空间。`uint x`实际是`uint256`，单个占32字节，占用槽位0，所以映射`data`将从槽位1开始。

`data[4][9].b`的位置在`keccak256(uint256(9) . keccak256(uint256(4) . uint256(1))) + 1`

有人在这里尝试直接读取区块链的存储值，https://github.com/ethereum/solidity/issues/1550

> 处于某些特定的环境下，可以看到评论框，欢迎留言交流^_^。

« 独立的汇编语言 内存变量的布局（Layout in Memory） »