

『0016』 - Solidity Types - 玩转 Solidity 数组 (Arrays)

孔壹学院：国内区块链职业教育领先品牌

作者：黎跃春，区块链、高可用架构工程师

微信：liyc1215 QQ群：348924182 博客：<http://liyuechun.org>

学习目标

1. 掌握Arrays的可变不可变的创建
2. 深度理解可变数组和不可变数组之间的区别
3. 二维数组
4. memory arrays的创建
5. bytes0 ~ bytes32、bytes与byte[]对比

固定长度的数组 (Arrays)

固定长度类型数组的声明

```
pragma solidity ^0.4.4;

contract C {

    // 数组的长度为5，数组里面的存储的值的类型为uint类型
    uint [5] T = [1,2,3,4,5];
}
```

通过length方法获取数组长度遍历数组求总和

```
pragma solidity ^0.4.4;

contract C {

    // 数组的长度为5，数组里面的存储的值的类型为uint类型
    uint [5] T = [1,2,3,4,5];
```

```
// 通过for循环计算数组内部的值的总和
function numbers() constant public returns (uint) {
    uint num = 0;
    for(uint i = 0; i < T.length; i++) {
        num = num + T[i];
    }
    return num;
}

}
```

尝试修改T数组的长度

```
pragma solidity ^0.4.4;

contract C {

    uint [5] T = [1,2,3,4,5];

    function setLength(uint len) public {

        T.length = len;
    }

}
```

The screenshot shows the Remix IDE interface. The code editor on the left displays the Solidity code. The right sidebar shows the 'Compile' tab with a red error message: `browser/ballot.sol:10:9: TypeError: Expression has to be an lvalue.` pointing to the line `T.length = len;`. A tooltip also displays this error message over the code editor.

PS:声明数组时，一旦长度固定，将不能再修改数组的长度。

尝试修改T数组内部值

```
pragma solidity ^0.4.4;
```

```

contract C {

    uint [5] T = [1,2,3,4,5];

    function setTIndex0Value() public {

        T[0] = 10;

    }

    // 通过for循环计算数组内部的值的总和
    function numbers() constant public returns (uint) {
        uint num = 0;
        for(uint i = 0; i < T.length; i++) {
            num = num + T[i];
        }
        return num;
    }

}

```

The screenshot shows the Remix IDE interface. On the left, the Solidity code is displayed in a file named `browser/ballot.sol`. The code defines a contract `C` with an array `T` of type `uint` containing the values `[1,2,3,4,5]`. It includes a function `setTIndex0Value()` that sets the first element of the array to 10, and a function `numbers()` that calculates the sum of all elements in the array. The right-hand panel contains the 'Run' tab, which shows the environment settings (JavaScript VM, account 0xca3...a733c (100 ether), gas limit 3000000, gas price 0, value 0). Below this, there is a section for the contract `browser/ballot.sol:C` with an 'At Address' field and a 'Create' button. At the bottom, there are two status bars: '0 pending transactions' and '0 contract instances'.

T 数组初始的内容为 `[1,2,3,4,5]`，总和为 15，当我点击 `setTIndex0Value` 方法将第 0 个索引的 1 修改为 10 时，总和为 24。

PS: 通过一个简单的试验可证明固定长度的数组只是不可修改它的长度，不过可以修改它内部的值，而 `bytes0 ~ bytes32` 固定大小字节数组中，大小固定，内容固定，长度和字节均不可修改。

尝试通过push往T数组中添加值

```
pragma solidity ^0.4.4;

contract C {

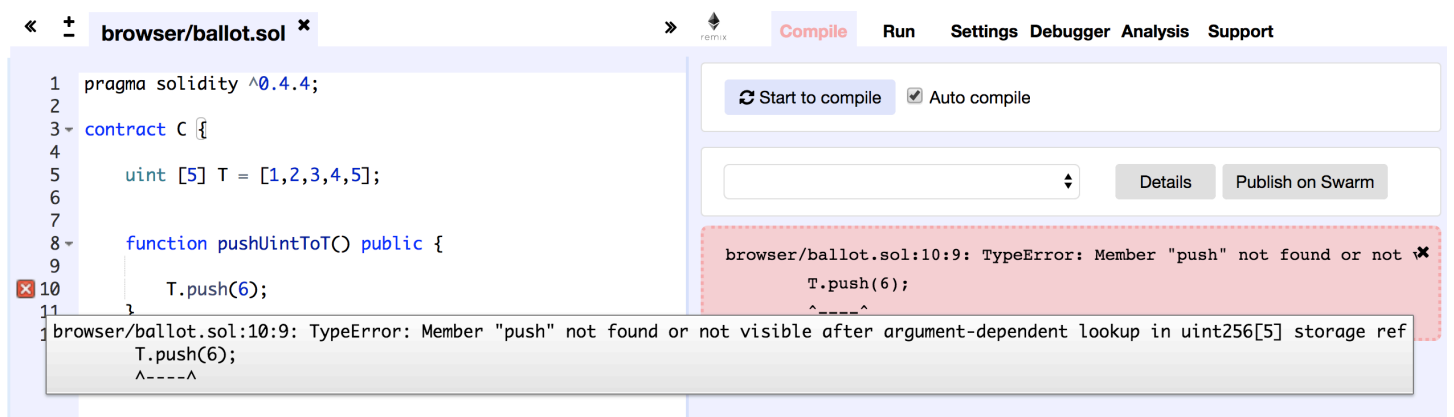
    uint [5] T = [1,2,3,4,5];

    function pushUintToT() public {

        T.push(6);

    }

}
```



PS:固定大小的数组不能调用 `push` 方法向里面添加存储内容，声明一个固定长度的数组，比如： `uint [5] T`，数组里面的默认值为 `[0,0,0,0,0]`，我们可以通过索引修改里面的值，但是不可修改数组长度以及不可通过 `push` 添加存储内容。

可变长度的Arrays

可变长度类型数组的声明

```
pragma solidity ^0.4.4;

contract C {

    uint [] T = [1,2,3,4,5];

}
```

```

function T_Length() constant returns (uint) {

    return T.length;
}

}

```

uint [] T = [1,2,3,4,5]，这句代码表示声明了一个可变长度的 T 数组，因为我们给它初始化了 5 个无符号整数，所以它的长度默认为 5。

The screenshot shows the Remix IDE interface. On the left, the Solidity code is displayed:

```

1 pragma solidity ^0.4.4;
2
3 contract C {
4
5     uint [] T = [1,2,3,4,5];
6
7     function T_Length() constant returns (uint) {
8
9         return T.length;
10    }
11
12 }

```

On the right, the 'Run' tab is active, showing the execution environment. The 'Environment' section is set to 'JavaScript VM'. The 'Account' is '0xca3...a733c (99.9999999999997704)'. The 'Gas limit' is '3000000', 'Gas Price' is '0', and 'Value' is '0'. Below this, the 'Create' button is visible. The 'Transaction' section shows '0 pending transactions'. The 'Debugger' section shows the execution of the 'T_Length' function, with the return value '0: uint256: 5' displayed. A red arrow points to this value.

通过length方法获取数组长度遍历数组求总和

```

pragma solidity ^0.4.4;

contract C {

    uint [] T = [1,2,3,4,5];

    // 通过for循环计算数组内部的值的总和
    function numbers() constant public returns (uint) {
        uint num = 0;
        for(uint i = 0; i < T.length; i++) {
            num = num + T[i];
        }
        return num;
    }
}

```

```
}
```

Remix IDE interface showing the Solidity code and the Run tab.

Code (browser/ballot.sol):

```
1 pragma solidity ^0.4.4;
2
3 contract C {
4
5     uint [] T = [1,2,3,4,5];
6
7     // 通过for循环计算数组内部的值的总和
8     function numbers() constant public returns (uint) {
9         uint num = 0;
10        for(uint i = 0; i < T.length; i++) {
11            num = num + T[i];
12        }
13        return num;
14    }
15 }
16 }
```

Run Tab:

- Environment: JavaScript VM
- Account: 0xca3...a733c (100 ether)
- Gas limit: 3000000
- Gas Price: 0
- Value: 0

Contract: browser/ballot.sol:C

At Address: Enter contract's address - i.e. 0x60606...

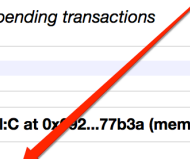
Create: [Button]

0 pending transactions

Contract State: browser/ballot.sol:C at 0x...77b3a (memory)

Variables:

Variable	Value
numbers	0: uint256: 15



尝试修改T数组的长度

```
pragma solidity ^0.4.4;

contract C {

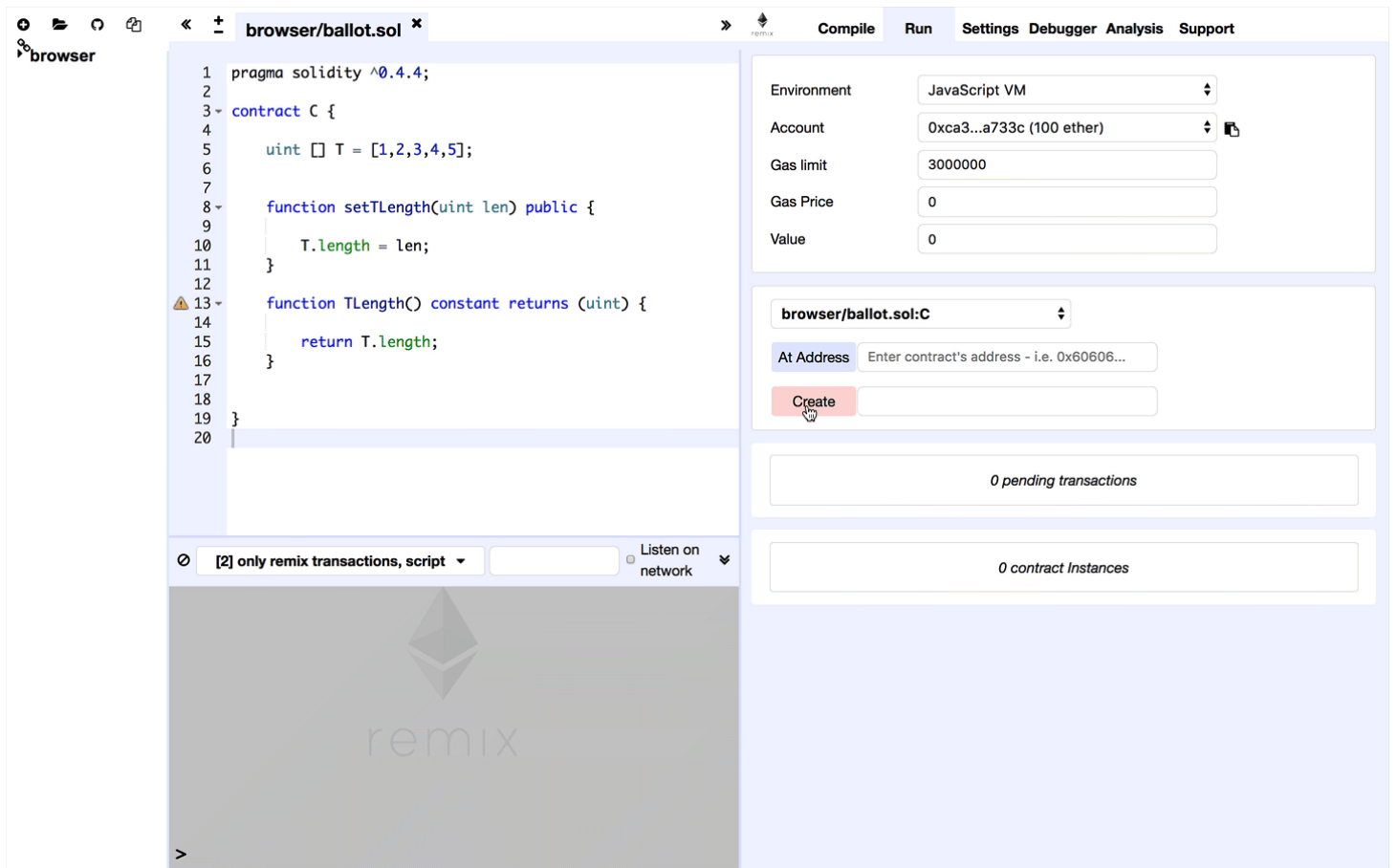
    uint [] T = [1,2,3,4,5];

    function setTLength(uint len) public {

        T.length = len;
    }

    function TLength() constant returns (uint) {

        return T.length;
    }
}
```



PS: 对可变长度的数组而言，可随时通过 `length` 修改它的长度。

尝试通过push往T数组中添加值

```
pragma solidity ^0.4.4;

contract C {

    uint [] T = [1,2,3,4,5];

    function T_Length() constant public returns (uint) {

        return T.length;
    }

    function pushUintToT() public {

        T.push(6);
    }

    function numbers() constant public returns (uint) {
        uint num = 0;
        for(uint i = 0; i < T.length; i++) {
            num = num + T[i];
        }
    }
}
```

```

    return num;
}
}

```

The screenshot shows the Remix IDE interface. On the left, the Solidity code for 'browser/ballot.sol' is displayed:

```

1 pragma solidity ^0.4.4;
2
3 contract C {
4     uint[] T = [1,2,3,4,5];
5
6     function T_Length() constant public returns (uint) {
7         return T.length;
8     }
9
10    function pushUIntToT() public {
11        T.push(6);
12    }
13
14    function numbers() constant public returns (uint) {
15        uint num = 0;
16        for(uint i = 0; i < T.length; i++) {
17            num = num + T[i];
18        }
19        return num;
20    }
21 }
22
23
24
25

```

On the right, the 'Run' tab is active, showing deployment settings:

- Environment: JavaScript VM
- Account: 0xca3...a733c (100 ether)
- Gas limit: 3000000
- Gas Price: 0
- Value: 0

Below the settings, the contract 'browser/ballot.sol:C' is selected. The 'At Address' field is empty, and the 'Create' button is highlighted. Below this, it shows '0 pending transactions'.

At the bottom, the console shows the transaction details for the creation of 'browser/ballot.sol:C':

```

[vm] from:0xca3...a733c, to:browser/ballot.sol:C.(constructor), val
ue:0 wei, data:0x606...80029, 0 logs, hash:0xcde...46ccc

```

PS: 当往里面增加一个值，数组的个数就会加1，当求和时也会将新增的数字加起来。

二维数组 - 数组里面放数组

```

pragma solidity ^0.4.4;

contract C {

    uint [2][3] T = [[1,2],[3,4],[5,6]];

    function T_len() constant public returns (uint) {

        return T.length; // 3
    }

}

```

uint [2][3] T = [[1,2],[3,4],[5,6]] 这是一个三行两列的数组，你会发现和Java、C语言等的其它语言中二维数组里面的列和行之间的顺序刚好相反。在其它语言中，上面的内容应该是

这么存储 `uint [2][3] T = [[1,2,3],[4,5,6]]` 。

上面的 数组T 是 `storage` 类型的数组，对于 `storage` 类型的数组，数组里面可以存放任意类型的值（比如：其它数组，结构体，字典 / 映射等等）。对于 `memory` 类型的数组，如果它是一个 `public` 类型的函数的参数，那么它里面的内容不能是一个 `mapping`(映射 / 字典)，并且它必须是一个 `ABI` 类型。

创建 Memory Arrays

创建一个长度为 `length` 的 `memory` 类型的数组可以通过 `new` 关键字来创建。`memory` 数组一旦创建，它不可通过 `length` 修改其长度。

```
pragma solidity ^0.4.4;

contract C {

    function f(uint len) {
        uint[] memory a = new uint[](7);
        bytes memory b = new bytes(len);
        // 在这段代码中 a.length == 7 、 b.length == len
        a[6] = 8;
    }
}
```

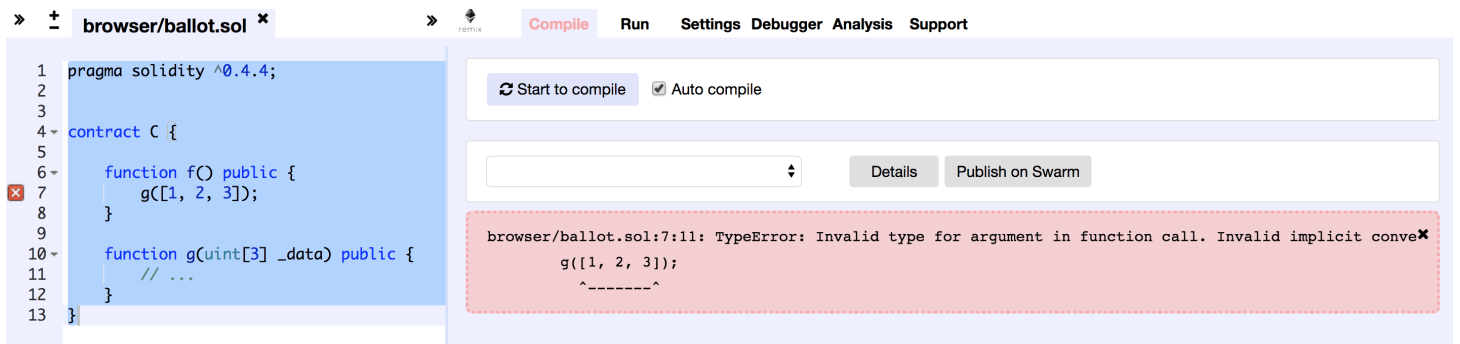
数组字面量 Array Literals / 内联数组 Inline Arrays

```
pragma solidity ^0.4.4;

contract C {

    function f() public {
        g([1, 2, 3]);
    }

    function g(uint[3] _data) public {
        // ...
    }
}
```



在上面的代码中，`[1, 2, 3]` 是 `uint8[3] memory` 类型，因为 `1`、`2`、`3` 都是 `uint8` 类型，他们的个数为 `3`，所以 `[1, 2, 3]` 是 `uint8[3] memory` 类型。但是在 `g` 函数中，参数类型为 `uint[3]` 类型，显然我们传入的数组类型不匹配，所以会报错。

正确的写法如下：

```
pragma solidity ^0.4.4;

contract C {

    function f() public {
        g([uint(1), 2, 3]);
    }

    function g(uint[3] _data) public {
        // ...
    }
}
```

在这段代码中，我们将 `[1, 2, 3]` 里面的第 `0` 个参数的类型强制转换为 `uint` 类型，所以整个 `[uint(1), 2, 3]` 的类型就匹配了 `g` 函数中的 `uint[3]` 类型。

memory类型的固定长度的数组不可直接赋值给 `storage / memory` 类型的可变数组

- `TypeError: Type uint256[3] memory is not implicitly convertible to expected type uint256[] memory.`

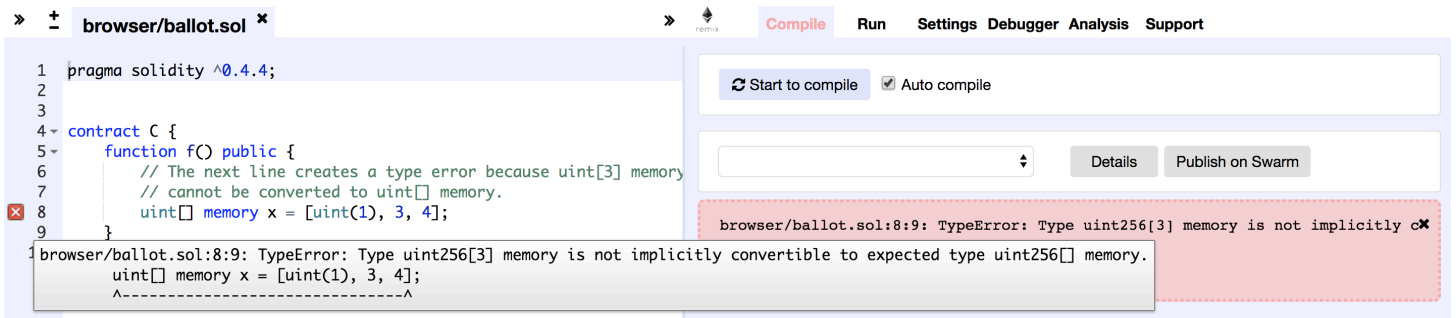
```
pragma solidity ^0.4.4;

contract C {
    function f() public {

        uint[] memory x = [uint(1), 3, 4];
    }
}
```

```
browser/ballot.sol:8:9: TypeError: Type uint256[3] memory is not implicitly convertible to expected type uint256[] memory.
```

```
    uint[] memory x = [uint(1), 3, 4];  
    ^-----^
```



- TypeError: Type uint256[3] memory is not implicitly convertible to expected type uint256[] storage pointer

```
pragma solidity ^0.4.4;  
  
contract C {  
    function f() public {  
  
        uint[] storage x = [uint(1), 3, 4];  
    }  
}
```

```
browser/ballot.sol:8:9: TypeError: Type uint256[3] memory is not implicitly convertible to expected type uint256[] storage pointer.
```

```
    uint[] storage x = [uint(1), 3, 4];  
    ^-----^
```

- 正确使用

```
pragma solidity ^0.4.4;  
  
contract C {  
    function f() public {  
  
        uint[3] memory x = [uint(1), 3, 4];  
    }  
}
```

创建固定大小字节数组 / 可变大小字节数组

之前我们的文章中深入讲解了 `bytes0 ~ bytes32`、`bytes` 以及 `string` 的使用。`bytes0 ~ bytes32` 创建的是固定字节大小的字节数组，长度不可变，内容不可修改。而 `string` 是特殊的可变字节数组，它可以转换为 `bytes` 以通过 `length` 获取它的字节长度，亦可通过索引修改相对应的字节内容。

创建可变字节数组除了可以通过 `bytes b = new bytes(len)` 来创建外，我们亦可以通过 `byte[] b` 来进行声明。

而 `bytes0 ~ bytes32` 我们可以通过 `byte[len] b` 来创建，`len` 的范围为 `0 ~ 32`。不过这两种方式创建的不可变字节数组有一小点区别，`bytes0 ~ bytes32` 直接声明的不可变字节数组中，长度不可变，内容不可修改。而 `byte[len] b` 创建的字节数组中，长度不可变，但是内容可修改。

```
pragma solidity ^0.4.4;

contract C {

    bytes9 a = 0x6c697975656368756e;
    byte[9] aa = [byte(0x6c),0x69,0x79,0x75,0x65,0x63,0x68,0x75,0x6e];

    byte[] cc = new byte[](10);

    function setAIndex0Byte() public {
        // 错误，不可修改
        a[0] = 0x89;
    }

    function setAAIndex0Byte() public {

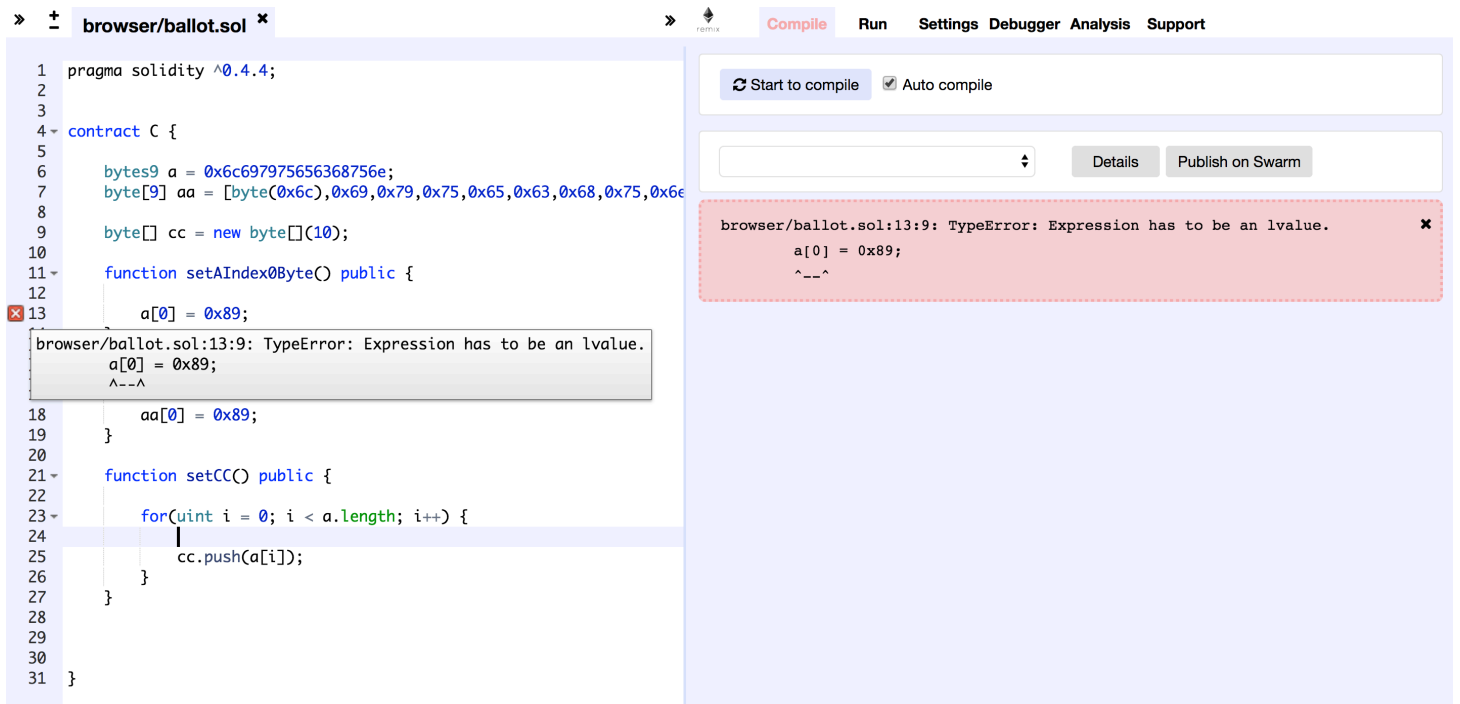
        aa[0] = 0x89;
    }

    function setCC() public {

        for(uint i = 0; i < a.length; i++) {

            cc.push(a[i]);
        }
    }

}
```



总结

本篇文章系统讲解了可变与不可变数组的创建、以及二位数组与其它语言中二位数组的区别，同时讲解了如何创建 `memory` 类型的数组以及对 `bytes0 ~ bytes32`、`bytes`与`byte[]` 对比分析。

技术交流

- 区块链技术交流QQ群：348924182
- 「区块链部落」官方公众号



长按，识别二维码，加关注