

# 『0015』 - Solidity Types - 动态大小字节数组（Dynamically-sized byte array）、固定大小字节数组（Fixed-size byte arrays）、string之间的转换关系

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>

## 大纲

1. 固定大小字节数组（Fixed-size byte arrays）之间的转换
2. 固定大小字节数组(Fixed-size byte arrays)转动态大小字节数组(Dynamically-sized byte array)
3. 固定大小字节数组（Fixed-size byte arrays）不能直接转换为string
4. 动态大小字节数组(Dynamically-sized byte array)转string
  - 本身就是动态大小字节数组
  - 固定大小字节数组转string，需先转动态字节数组，再转string

## 固定大小字节数组（Fixed-size byte arrays）之间的转换

固定大小字节我们可以通过 `bytes0 ~ bytes32` 来进行声明，固定大小字节数组的长度不可变，内容不可修改。接下来我们通过下面的代码看看固定大小字节之间的转换关系。

```
pragma solidity ^0.4.4;

contract C {

    bytes9 name9 = 0x6c697975656368756e;

    function bytes9ToBytes1() constant returns (bytes1) {
```

[illegible]

## 固定大小字节数组(Fixed-size byte arrays)转动态大小字节数组(Dynamically-sized byte array)

```
pragma solidity ^0.4.4;

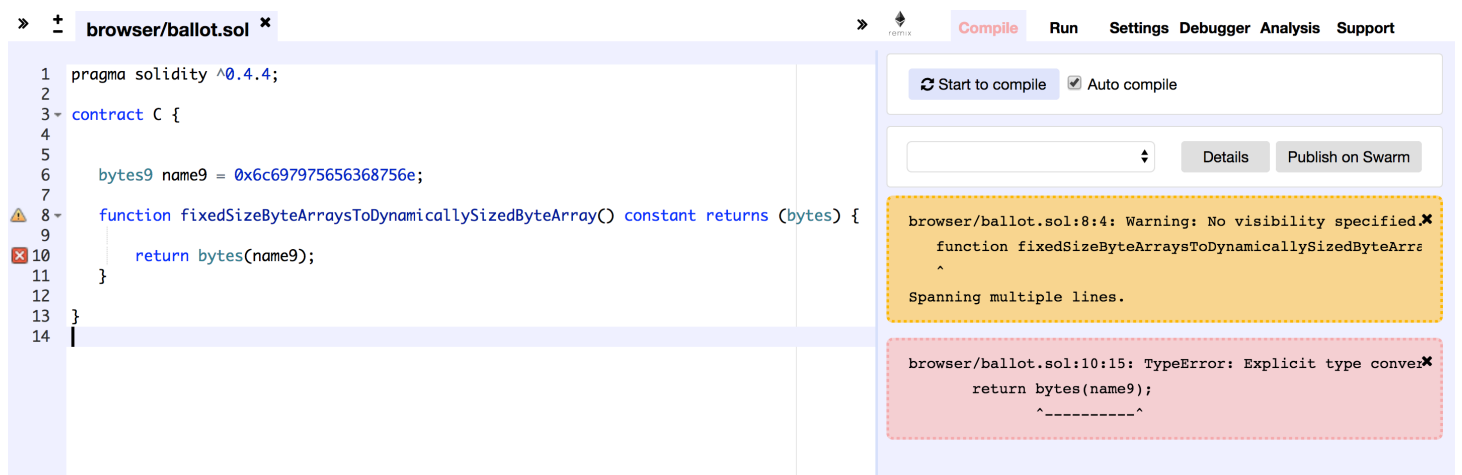
contract C {

    bytes9 name9 = 0x6c697975656368756e;

    function fixedSizeByteArraysToDynamicallySizedByteArray() constant returns (bytes) {

        return bytes(name9);
    }
}
```

对于刚接触的童鞋，很多人都会用上面的方法进行转换，以为理所当然，殊不知编译运行时，代码报错，原因如下：



**备注：**简言之，`固定大小字节数组` 和 `动态大小字节数组` 之间不能简单直接转换。

下面是 `固定大小字节数组` 转 `动态大小字节数组` 正确的姿势。

```
pragma solidity ^0.4.4;

contract C {

    bytes9 name9 = 0x6c697975656368756e;

    function fixedSizeByteArraysToDynamicallySizedByteArray() constant returns (bytes) {

        bytes memory names = new bytes(name9.length);
```

```

        for(uint i = 0; i < name9.length; i++) {

            names[i] = name9[i];
        }

        return names;
    }
}

```

The screenshot displays the Remix IDE interface. On the left, the Solidity code editor shows a contract named 'C' with a function 'fixedSizeByteArraysToDynamicallySizedByteArray' that takes a constant return type and returns a 'bytes' array. The code uses a 'for' loop to copy data from 'name9' to 'names'. The right sidebar contains the 'Environment' tab with settings for 'JavaScript VM', 'Account' (0xca3...a733c), 'Gas limit' (3000000), 'Gas Price' (0), and 'Value' (0). Below this is the 'Contract' tab showing 'browser/ballot.sol:C' with an 'At Address' field and a 'Create' button. The bottom section shows the 'Transaction Log' with a pending transaction for 'creation of browser/ballot.sol:C' and a call to the 'fixedSizeByteArraysToDynamicallySizedByteArray' function, which returns a 'bytes' array of length 20.

在上面的代码中，我们根据固定字节大小数组的长度来创建一个 `memory` 类型的动态类型的字节数组，然后通过一个 `for` 循环 将固定大小字节数组中的字节按照索引赋给动态大小字节数组即可。

## 固定大小字节数组（Fixed-size byte arrays）不能直接转换为string

```

pragma solidity ^0.4.4;

contract C {

```

```

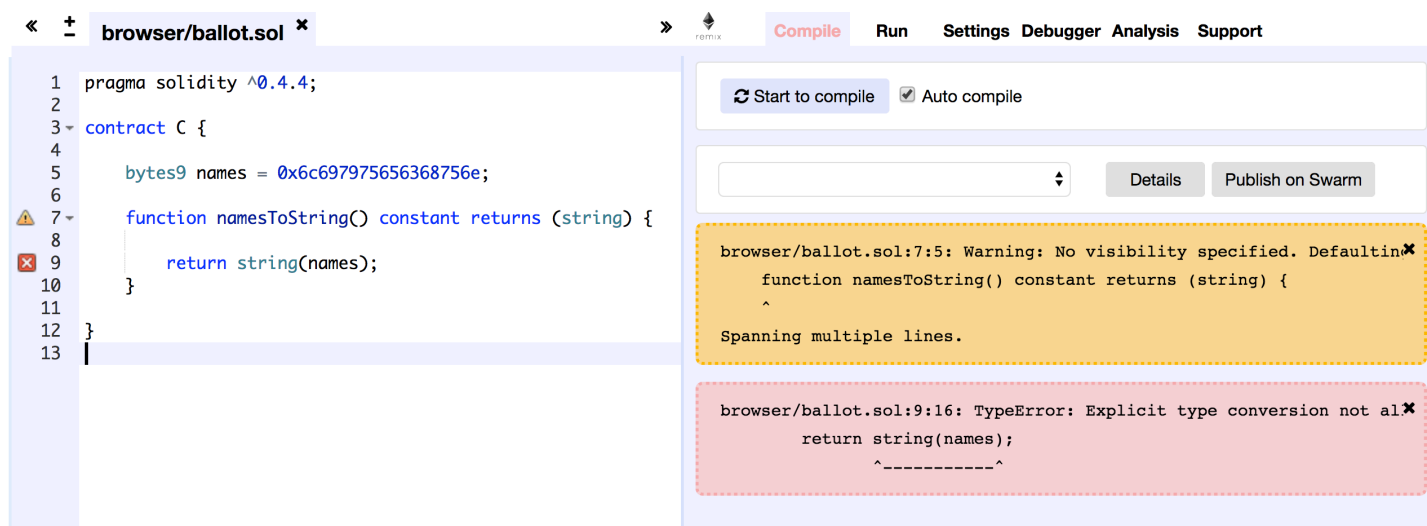
bytes9 names = 0x6c697975656368756e;

function namesToString() constant returns (string) {

    return string(names);
}

}

```



## 动态大小字节数组(Dynamically-sized byte array)转string

**重要：**因为string是特殊的动态字节数组，所以string只能和动态大小字节数组(Dynamically-sized byte array)之间进行转换，不能和固定大小字节数组进行转行。

- 如果是现成的动态大小字节数组(Dynamically-sized byte array)，如下：

```

pragma solidity ^0.4.4;

contract C {

    bytes names = new bytes(2);

    function C() {

        names[0] = 0x6c;
        names[1] = 0x69;
    }

    function namesToString() constant returns (string) {

```

```

    return string(names);
}

}

```

The screenshot shows the Remix IDE interface. On the left, the Solidity code for a contract named 'C' is displayed. The code defines a 'names' array of 2 bytes and a 'namesToString()' function that returns the string representation of the array. The right panel shows the 'Run' tab with the 'JavaScript VM' environment selected. The 'Account' field is set to '0xca3...a733c (99.999999999999769€)'. The 'Gas limit' is 3000000, 'Gas Price' is 0, and 'Value' is 0. Below this, the 'Create' button is visible. The bottom right panel shows the 'browser/ballot.sol:C' contract at address '0x692...77b3a (memory)'. The 'namesToString' function is highlighted, showing the result '0: string: li'.

```

1 pragma solidity ^0.4.4;
2
3 contract C {
4
5     bytes names = new bytes(2);
6
7     function C() {
8
9         names[0] = 0x6c;
10        names[1] = 0x69;
11    }
12
13
14    function namesToString() constant returns (string) {
15
16        return string(names);
17    }
18
19 }
20

```

Environment: JavaScript VM

Account: 0xca3...a733c (99.999999999999769€)

Gas limit: 3000000

Gas Price: 0

Value: 0

browser/ballot.sol:C

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

Create

0 pending transactions

browser/ballot.sol:C at 0x692...77b3a (memory)

namesToString 0: string: li

- 如果是固定大小字节数组转string，那么就需要先将字节数组转动态字节数组，再转字符串

```

pragma solidity ^0.4.4;

contract C {

    function byte32ToString(bytes32 b) constant returns (string) {

        bytes memory names = new bytes(b.length);

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

            names[i] = b[i];
        }

        return string(names);
    }

}

```



```
function bytes32ToString(bytes32 x) constant returns (string) {
    //0x6c697975656368756e0000000000000000000000000000000000000000000000000000000000000000
    bytes memory bytesString = new bytes(32);
    uint charCount = 0;
    for (uint j = 0; j < 32; j++) {

        // 6 * 2 12

        // 0000001 1000

        // 0x697975656368756e000000000000000000000000000000000000000000000000
000000
        // byte char = byte(bytes32(uint(x) * 2 ** (8 * j)));
        byte char = byte(bytes32(uint(x) << (8 * j)));

        if (char != 0) {
            bytesString[charCount] = char;

            // 0 0x6c
            // 1 0x69
            // 2 0x79
            // 3 0x95
            // .....
            // 8 0x6e

            charCount++;
        }
    }
    bytes memory bytesStringTrimmed = new bytes(charCount); //0x6c697975
656368756e
    for (j = 0; j < charCount; j++) {
        bytesStringTrimmed[j] = bytesString[j];
    }
    return string(bytesStringTrimmed);
}
```



[illegible]

`byte char = byte(bytes32(uint(x) * 2 ** (8 * j)))` 在上面的代码中，估计大家最难看懂的就是这一句代码，我们通过下面的案例给大家解析：

```
pragma solidity ^0.4.4;

contract C {

    // 0x6c

    function uintValue() constant returns (uint) {

        return uint(0x6c);
    }

    function bytes32To0x6c() constant returns (bytes32) {

        return bytes32(0x6c);
    }

    function bytes32To0x6cLeft00() constant returns (bytes32) {

        return bytes32(uint(0x6c) * 2 ** (8 * 0));
    }

    function bytes32To0x6cLeft01() constant returns (bytes32) {
```

```

    return bytes32(uint(0x6c) * 2 ** (8 * 1));
}

function bytes32To0x6cLeft31() constant returns (bytes32) {

    return bytes32(uint(0x6c) * 2 ** (8 * 31));
}
}

```

The screenshot shows the Remix IDE interface. On the left, the Solidity code for a contract named 'C' is displayed. It includes a pragma statement for Solidity ^0.4.4, a contract definition, and four functions: `uintValue()`, `bytes32To0x6c()`, `bytes32To0x6cLeft00()`, `bytes32To0x6cLeft01()`, and `bytes32To0x6cLeft31()`. The `bytes32To0x6cLeft31()` function is highlighted with a red box. On the right, the 'Run' tab is active, showing the execution environment. The 'Environment' section lists 'JavaScript VM', 'Account' (0xca3...a733c), 'Gas limit' (3000000), 'Gas Price' (0), and 'Value' (0). Below this, the 'Create' button is visible. The 'Memory' section shows the memory layout for the contract 'C' at address 0x692...77b3a. It lists variables: `uintValue` (0: uint256: 108), `bytes32To0x6cLeft31` (0: bytes32: 0x6c00), `bytes32To0x6cLeft01` (0: bytes32: 0x00), `bytes32To0x6cLeft00` (0: bytes32: 0x00), and `bytes32To0x6c` (0: bytes32: 0x00). Red arrows point from the function names in the code to their corresponding memory entries.

- `bytes32(uint(0x6c) * 2 ** (8 * 31));` 左移31位
- `bytes32(uint(0x6c) * 2 ** (8 * 1));` 左移1位

通过 `byte(bytes32(uint(x) * 2 ** (8 * j)))` 获取到的始终是第0个字节。

## 总结

`string` 本身是一个特殊的动态字节数组，所以它只能和 `bytes` 之间进行转换，不能和固定大小字节数组进行直接转换，如果是固定字节大小数组，需要将其转换为动态字节大小数组才能进行转换。

## 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