『0007』- Solidity状态变量、局部变量与memory 、storage之间的爱恨情仇

> 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

在上一节中，我们了解了Solidity类型中哪些是**值类型**，哪些是**引用类型**，以及**值类型**与**引用类型**的简单对比。

本篇教程中，我们将全面讲解`memory`，`storage`在Solidity开发中的作用，以及`值类型`、`引用类型`在合约中`memory / storage`关键字的区别。

一段代码清楚认识状态变量、局部变量

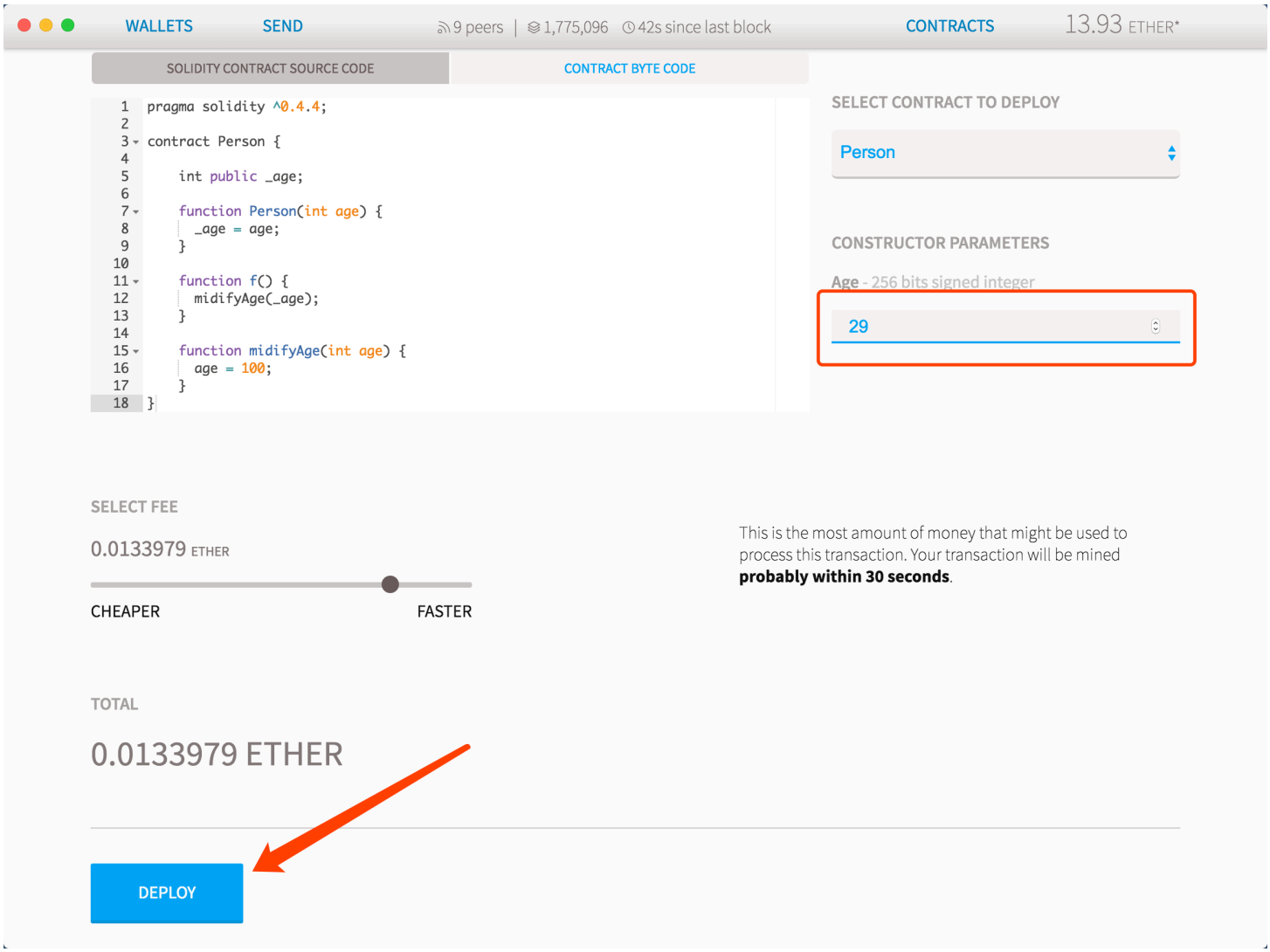
```
pragma solidity ^0.4.4;

contract Person {

    int public _age;
    string public _name;

    function Person(int age,string name) {
          _age = age;
          _name = name;
    }


    function f(string name) {
          var name1 = name;
          ...
    }
}
```

在这段代码中，`_age`，`_name`就属于**状态变量**，`Person(int age,string name)`中的`age`和`name`，还有`f(string name)`中的`name`以及`f()`函数中声明的`name1`都默认属于**本地 / 局部变量**。

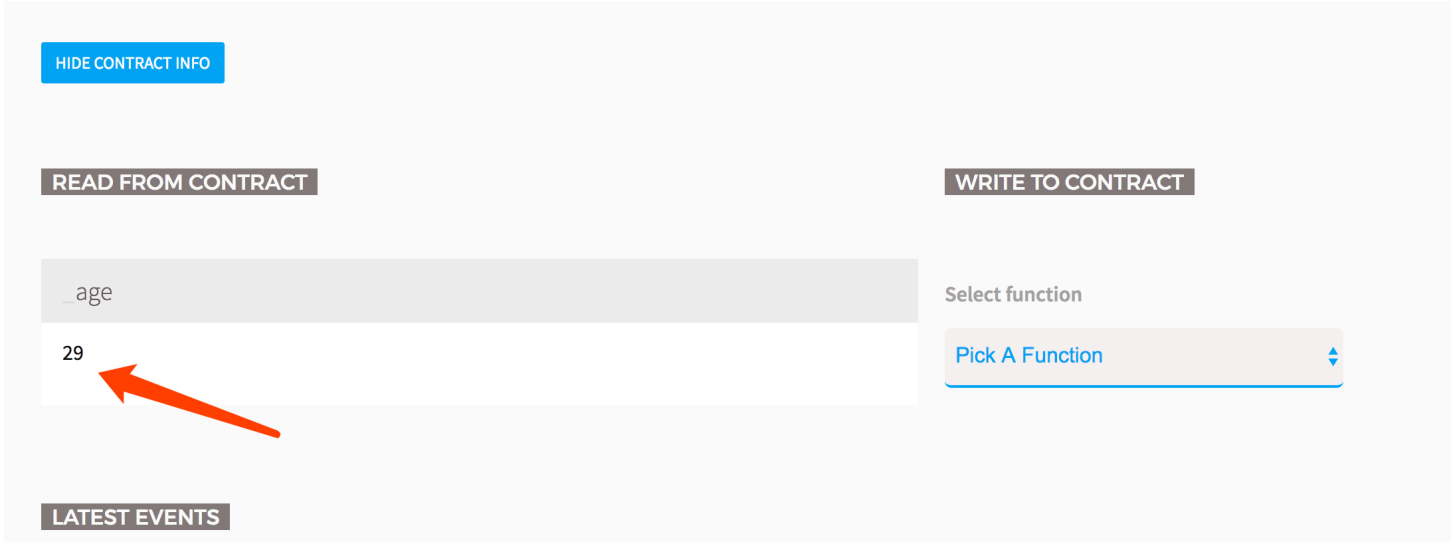
值类型代码演示

```
pragma solidity ^0.4.4;

contract Person {

    int public _age;

    function Person(int age) {
      _age = age;
    }

    function f() {
      midifyAge(_age);
    }

    function midifyAge(int age) {
      age = 100;
    }
}
```

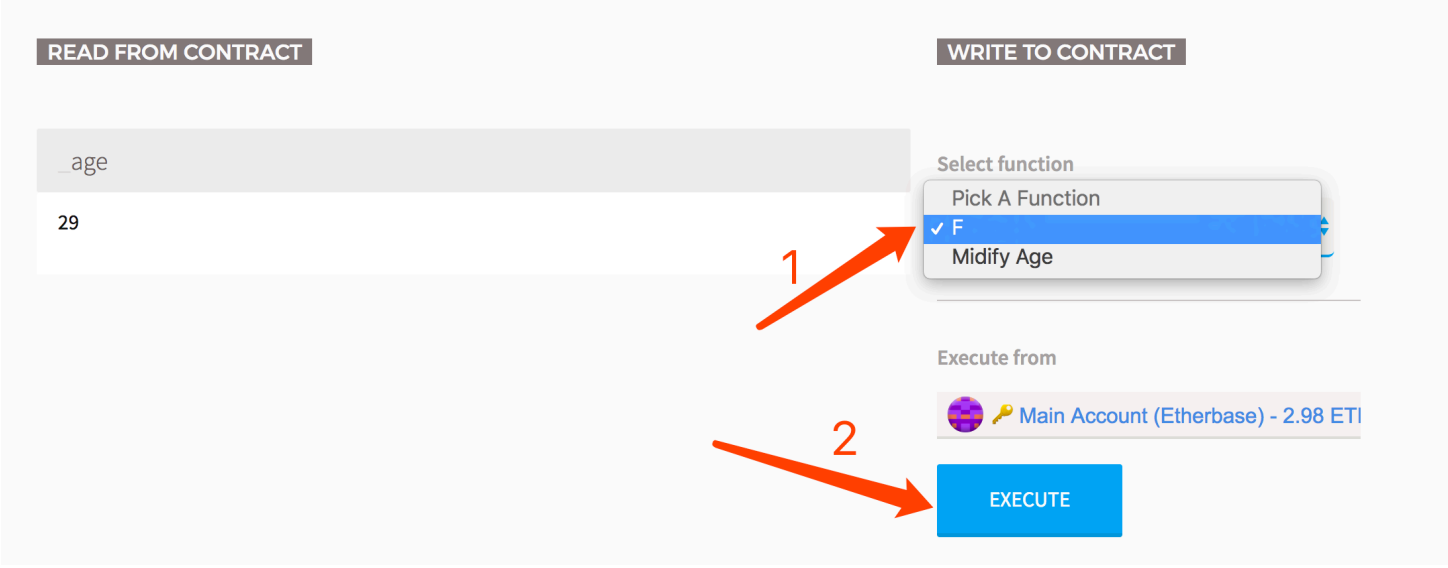
在这段代码中，在我们创建合约时，因为构造函数中需要传入一个参数`age`，如下图所示，我传入的值是`29`。

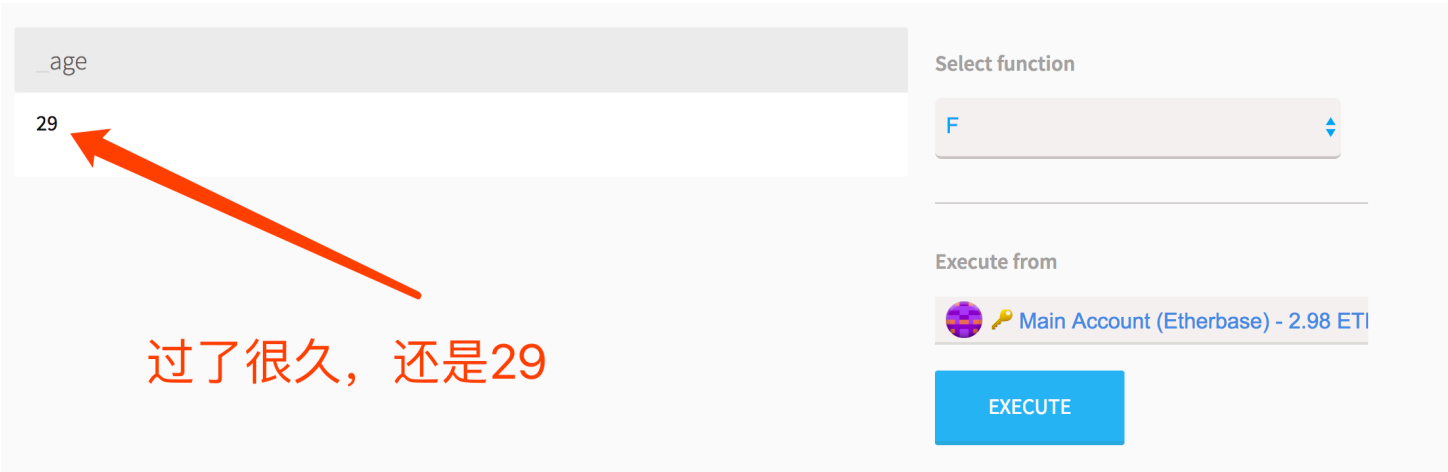


合约创建完后，我们很容易在界面中看到 `_age` 的初始值为 `29` 。



接下来我们切换到 `f` 方法，然后点击执行，因为 `_age` 是值类型，所以在函数传参或者将值类型的变量值赋值给一个新变量，当我们尝试修改新变量时，原来的 `值类型` 变量值并不会发生任何变化，在本案例中，当我们调用 `midifyAge(_age)` 代码时，我们可以理解成，创建了一个临时变量 `age` ，并且将 `_age` 的值传给了 `age` ，因为是值传递，当我们尝试在 `midifyAge` 函数中修改新变量 `age` 的值时，原来的变量值 `_age` 的值保持不变。





引用类型memory/storage

引用类型的变量有两种类型，分别是`memory`和`storage`。

memory（值传递）

```
pragma solidity ^0.4.4;

contract Person {

    string public  _name;

    function Person() {
        _name = "liyuechun";
    }

    function f() {

        modifyName(_name);
    }
```

```
    function modifyName(string name)  {

        var name1 = name;
        bytes(name1)[0] = 'L';
    }
}
```

代码解析

上面的代码中：

```
function modifyName(string name)  {

    var name1 = name;
    bytes(name1)[0] = 'L';
}
```

等价于：

```
function modifyName(string memory name)  {

    var name1 = name;
    bytes(name1)[0] = 'L';
}
```

等价于：

```
function modifyName(string memory name)  {

    string  memory name1 = name;
    bytes(name1)[0] = 'L';
}
```

等价于：

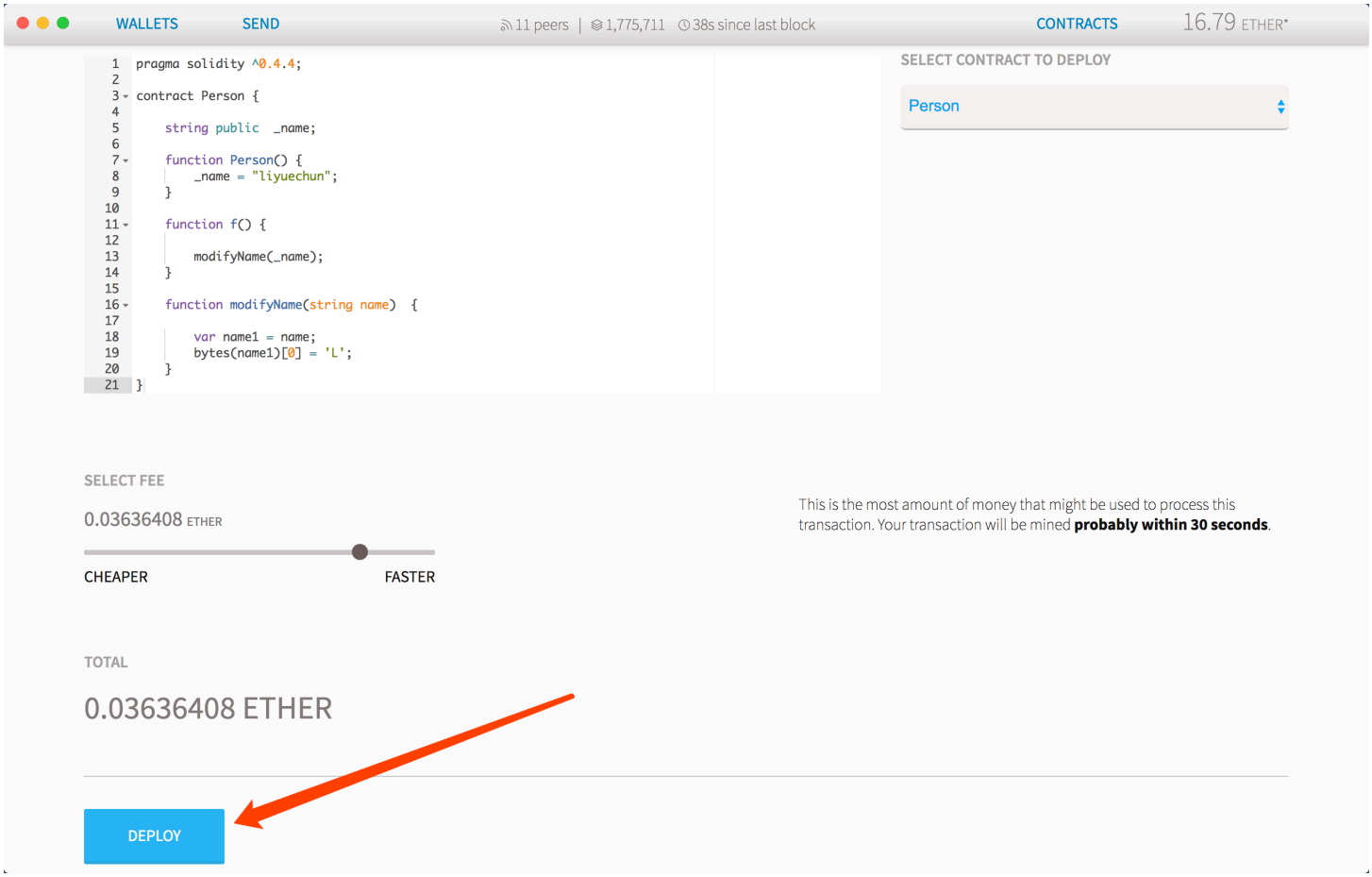
```
function modifyName(string name)  {

    string  memory name1 = name;
    bytes(name1)[0] = 'L';
}
```

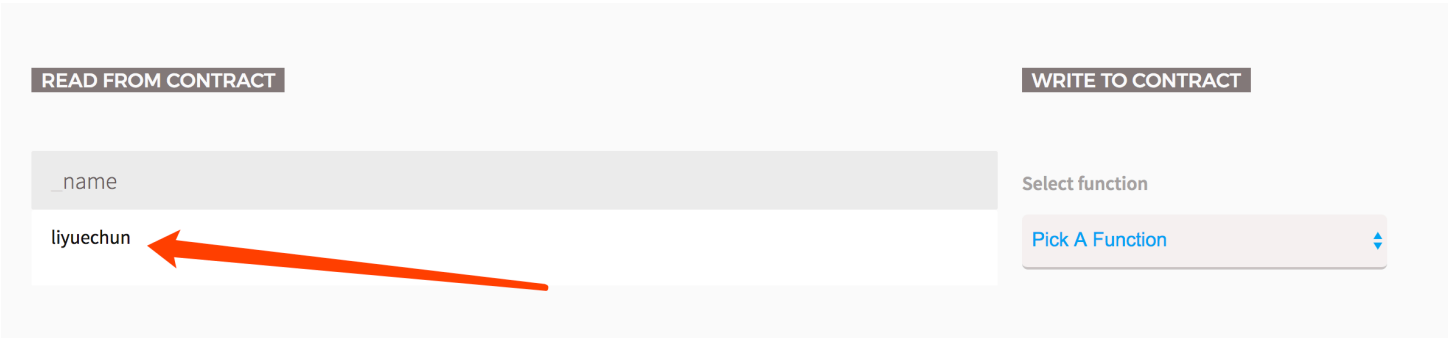
由上面几种写法，我们不难看出，当**引用类型作为函数参数**时，它的类型默认为`memory`，函数参数为`memory`类型的变量给一个变量赋值时，这个变量的类型必须和函数参数类型一致，所以我们可以写成`string memory name1 = name;`，或者`var name1 = name;`，**var**声明一个变量时，这个变量的类型最终由赋给它值的类型决定。

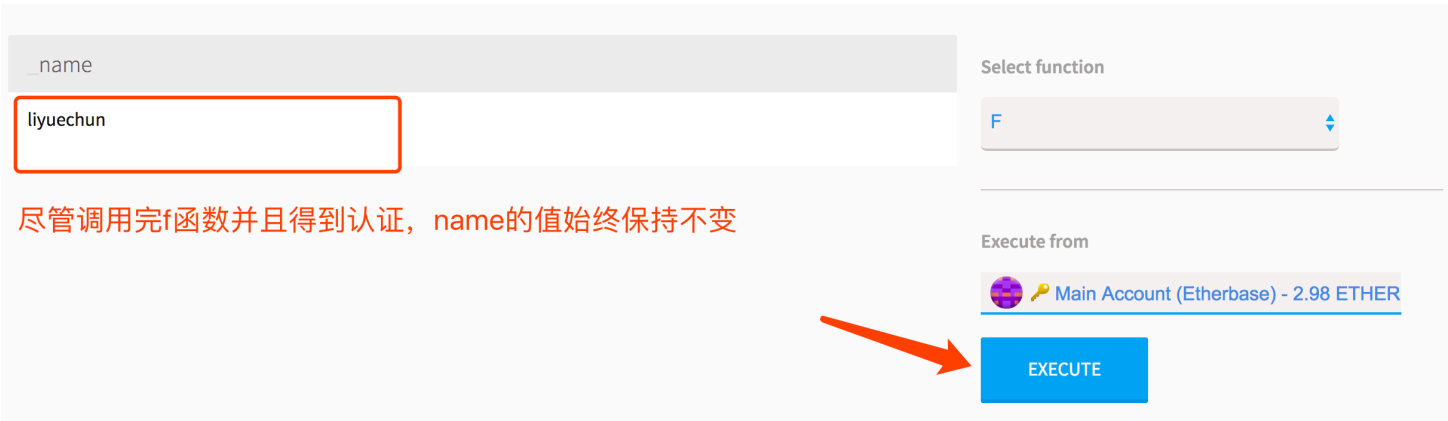
任何函数参数当它的类型为引用类型时，这个函数参数都默认为memory类型，memory类型的变量会临时拷贝一份值存储到内存中，当我们将这个参数值赋给一个新的变量，并尝试去修改这个新的变量的值时，最原始的变量的值并不会发生变化。

例如：

在本案例中，当创建合约时，`_name`的值为`liyuechun`，当我们调用`f()`函数时，`f()`函数中会将`_name`的值赋给临时的`memory`变量`name`，换句话说，因为`name`的类型为`memory`，所以`name`和`_name`会分别指向不同的对象，当我们尝试去修改`name`指针指向的值时，`_name`所指向的内容不会发生变化。







storage（指针传递）

当函数参数为`memory`类型时，相当于**值传递**，而`storage`类型的函数参数将是指针传递。

如果想要在`modifyName`函数中通过传递过来的指针修改`_name`的值，那么必须将函数参数的类型显示设置为`storage`类型，`storage`类型拷贝的不是值，而是`_name`指针，当调用`modifyName(_name)`函数时，相当于同时有`_name`，`name`，`name1`三个指针同时指向同一个对象，我们可以通过三个指针中的任何一个指针修改他们共同指向的内容的值。

```
pragma solidity ^0.4.4;

contract Person {

    string public  _name;

    function Person() {
        _name = "liyuechun";
    }

    function f() {

        modifyName(_name);
    }

    function modifyName(string storage name)  {
```

```
        var name1 = name;
        bytes(name1)[0] = 'L';
    }
}
```

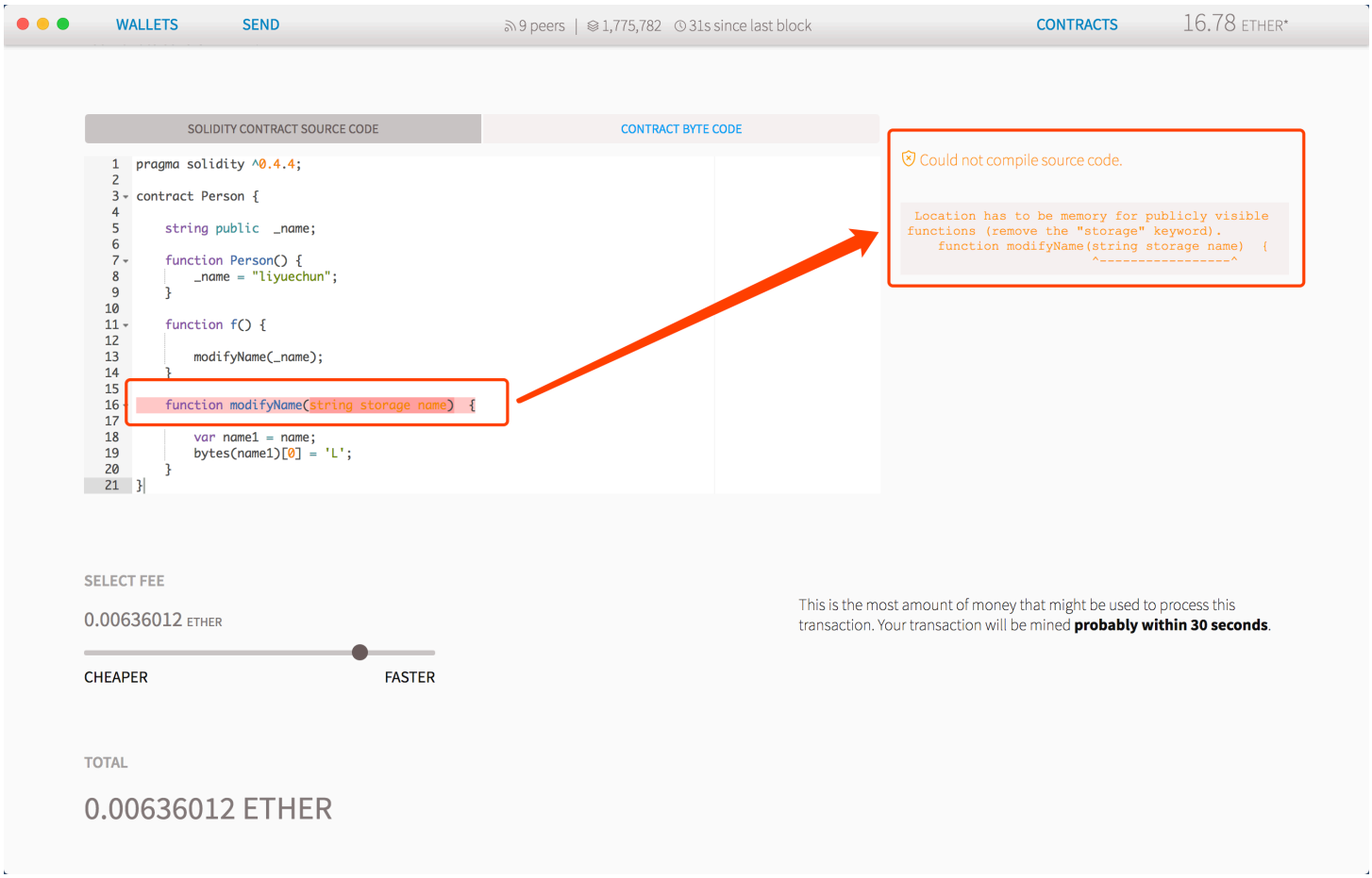
备注：

```
function modifyName(string storage name)  {

    var name1 = name;
    bytes(name1)[0] = 'L';
}
```

等价于：

```
function modifyName(string storage name)  {

    string storage name1 = name;
    bytes(name1)[0] = 'L';
}
```

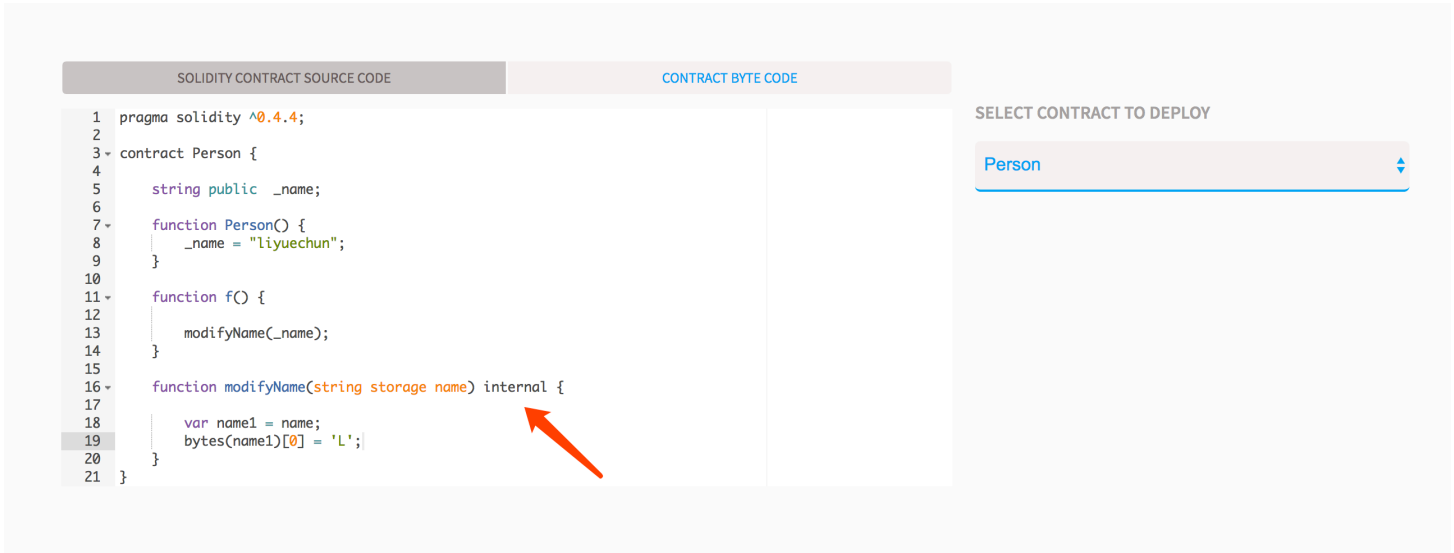
接下来我们将上面的代码拷贝到`Ethereum Wallet`中，你会发现有一个地方会报错。如下图所示：

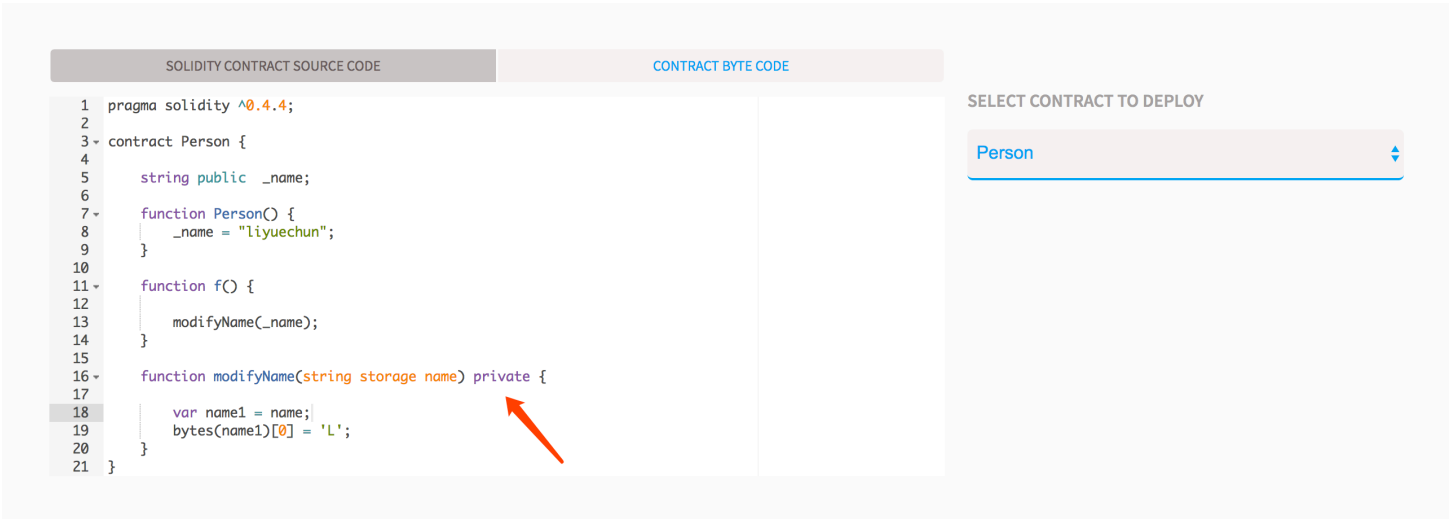


报错的内容为：

```
Location has to be memory for publicly visible
functions (remove the "storage" keyword).
   function modifyName(string storage name)  {
                       ^-----------------^
```

报错原因：因为函数默认为`public`类型，但是当我们的函数参数如果为`storage`类型时，函数的类型必须为`internal`或者`private`

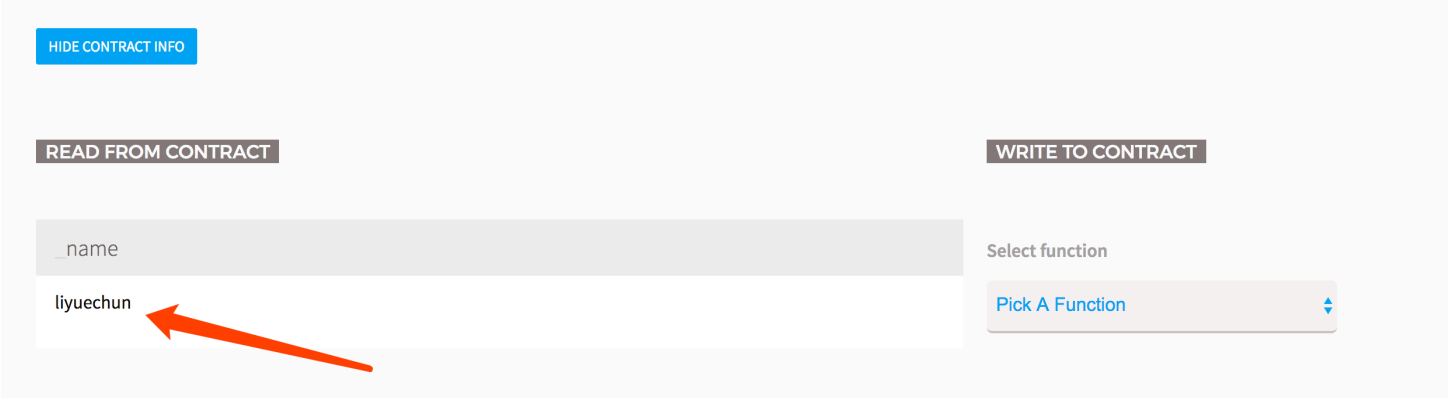


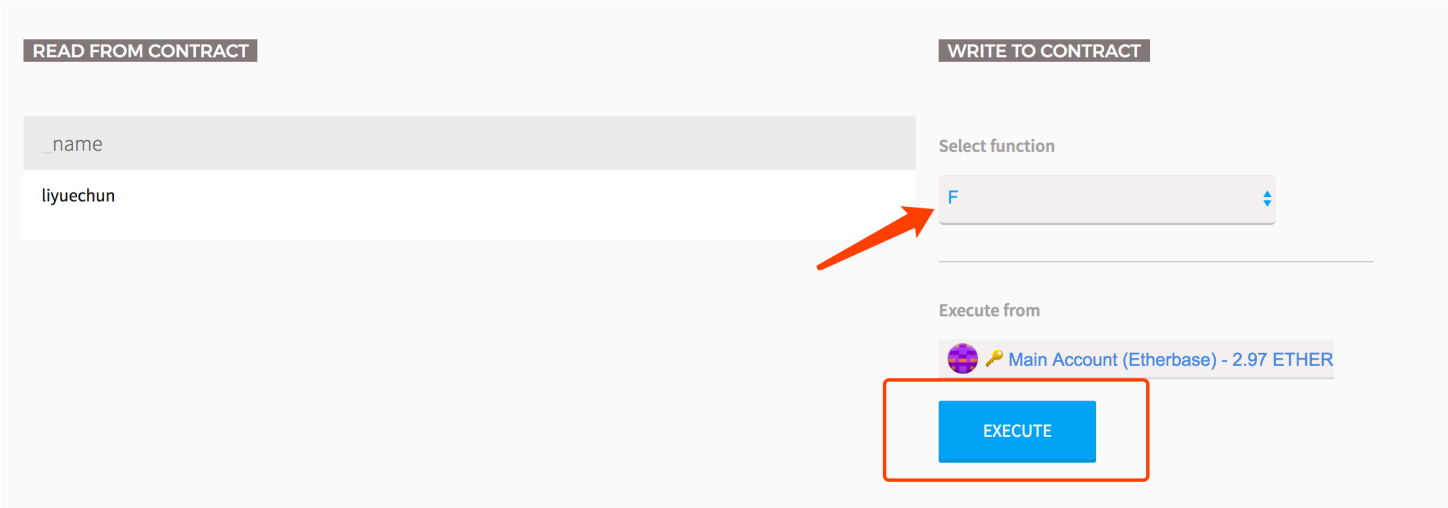


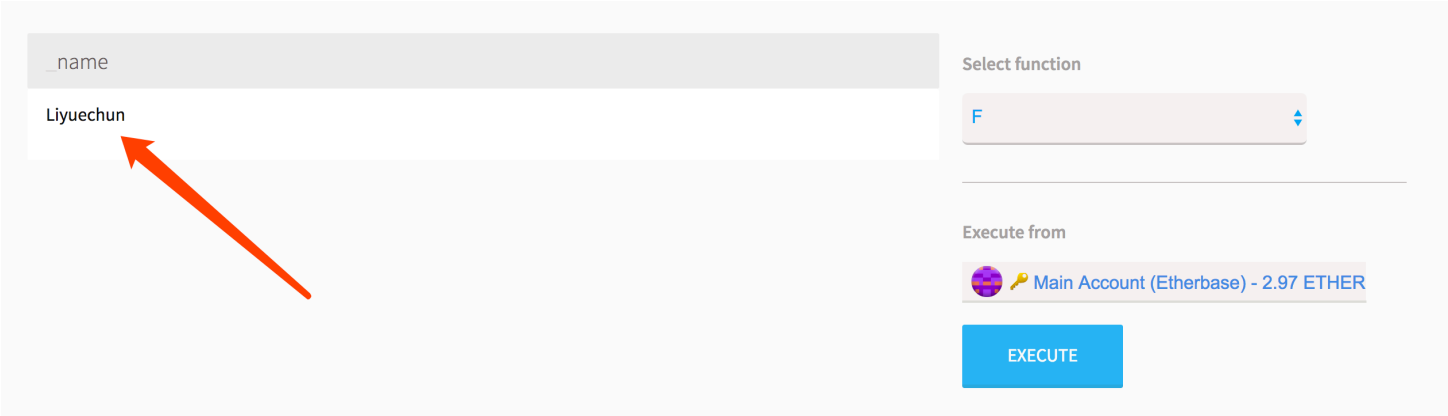
完整无错误代码如下：

```
pragma solidity ^0.4.4;

contract Person {

    string public  _name;

    function Person() {
        _name = "liyuechun";
    }

    function f() {

        modifyName(_name);
    }

    function modifyName(string storage name) internal {

        var name1 = name;
        bytes(name1)[0] = 'L';
    }
}
```

- 部署代码







小结

本篇文章主要全面讲解`memory`,`storage`在Solidity开发中的作用，以及`值类型`、`引用类型`在合约中`memory`/`storage`关键字的区别。在本篇教程中，我们使用`storage`关键字时，我们同时使用了两个关键字`internal`和`private`，下一篇教程中，春哥将为大家介绍合约中**状态变量**和**函数**的**权限**问题。

技术交流

- 区块链技术交流QQ群：348924182

- 「区块链部落」官方公众号