第6章 面向对象

目录

CONTENTS

CONTENTS

章节

第五章 抽象合约

5.1 抽象合约

腾讯云

抽象合约

- 当合约中至少有一个函数未实现时（未实现：定义了函数名、参数和返回值，未定义函数体），该合约则为抽象合约。即使所有的函数都已实现也可以将其标记为抽象合约。
- 抽象合约使用abstract关键字标识，抽象合约不能被实例化，必须由子合约继承使用，子合约必须实现父抽象合约中未实现的函数。
- 抽象合约有利于定义合约的结构。

抽象合约

- 抽象合约示例

Test.sol 保存 编译 部署

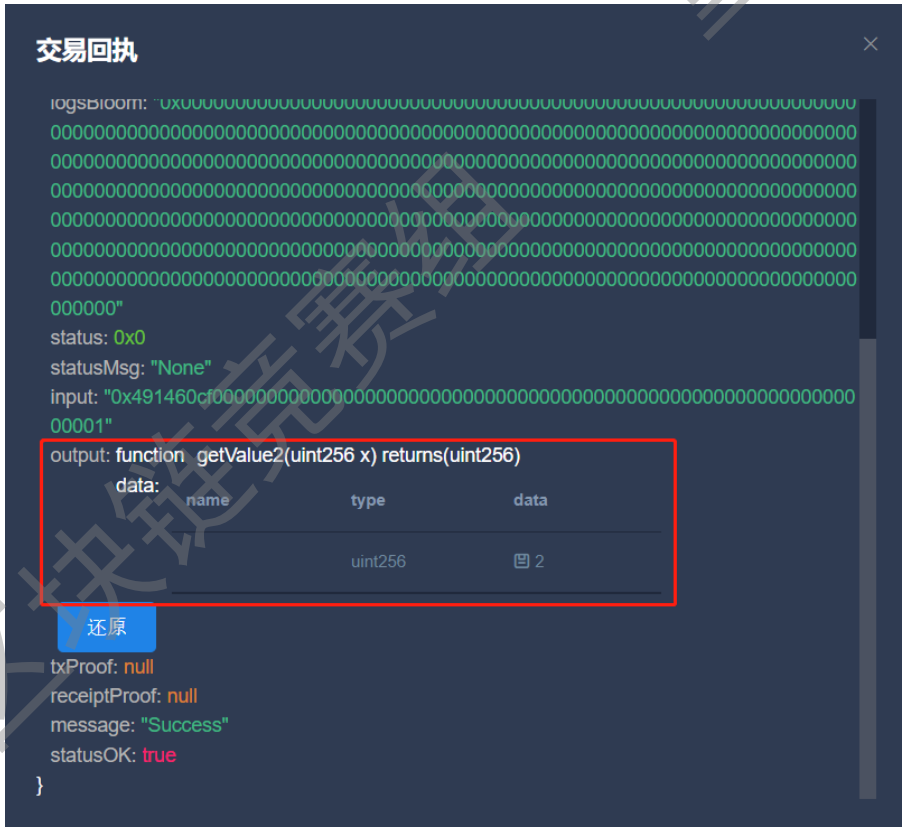
```
pragma solidity 0.6.10;


// 抽象合约Test
abstract contract Test {
    // getValue函数，返回x+1
    function getValue(uint x) public virtual returns(uint) {
        return x + 1;
    }

    // getValue2函数，只定义了参数和返回，未定义函数内容
    function getValue2(uint x) public virtual returns(uint);
}
```

Test2.sol 保存 编译 部署

```
pragma solidity 0.6.10;
import "./Test.sol";

// Test2合约继承抽象合约
contract Test2 is Test {

    // 获取输入的value+1值
    function getValue2(uint x) public override returns(uint) {
        return x + 1;
    }
}
```

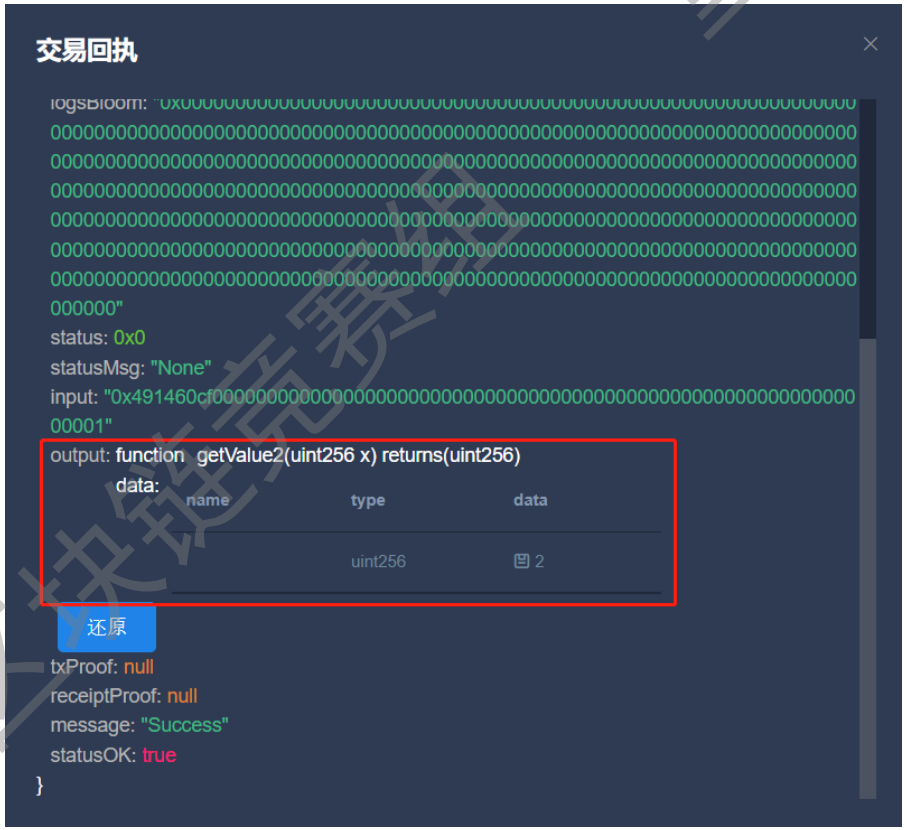
抽象合约

- 部署Test2合约，调用getValue2函数，参数x=1，查看运行结果



抽象合约

- 部署Test2合约，调用getValue2函数，参数x=1，查看运行结果



CONTENTS

章节

第六章 接口合约

6.1 接口合约

腾讯云

接口合约

- 接口合约与抽象合约类似，不同的点是不能包含任何已实现的函数，合约中只包含函数声明，因此接口合约也被称为纯抽象合约。
- 接口合约使用interface关键字声明，函数必须使用external限定符
- 接口合约中不能声明constructor构造函数
- 接口合约中不能声明全局变量

接口合约

- 接口合约示例

Math.sol 保存 编译

```
pragma solidity 0.6.10;

// 数学计算接口合约
interface Math {
    // 加法
    function add(int x, int y) external returns(int);

    // 减法
    function sub(int x, int y) external returns(int);

    // 乘法
    function mul(int x, int y) external returns(int);

    // 除法
    function div(int x, int y) external returns(int);

}
```

接口合约

- Test合约使用Math接口合约，与继承抽象合约用法一致

Test.sol　保存　编译　部署

```
pragma solidity 0.6.10;
import "./Math.sol";

contract Test is Math {
    // 加法
    function add(int x, int y) public override returns(int) {
        return x + y;
    }
    // 减法
    function sub(int x, int y) public override returns(int) {
        return x - y;
    }
    // 乘法
    function mul(int x, int y) public override returns(int) {
        return x * y;
    }
    // 除法
    function div(int x, int y) public override returns(int) {
        return x / y;
    }
}
```

CONTENTS

章节

第七章 库合约

7.1 库合约

腾讯云

库合约

- 库合约的编写方式与普通合约类似，但库合约是无状态的，不保存任何变量，只包含声明
- 库合约使用`library`关键字标识，可以供其他任意合约调用
- 使用库合约的合约隐式继承了库合约，因此能在不使用继承关键字的前提下调用库合约中的内部函数

库合约

- 库合约示例

SafeSub.sol 保存 编译 部署

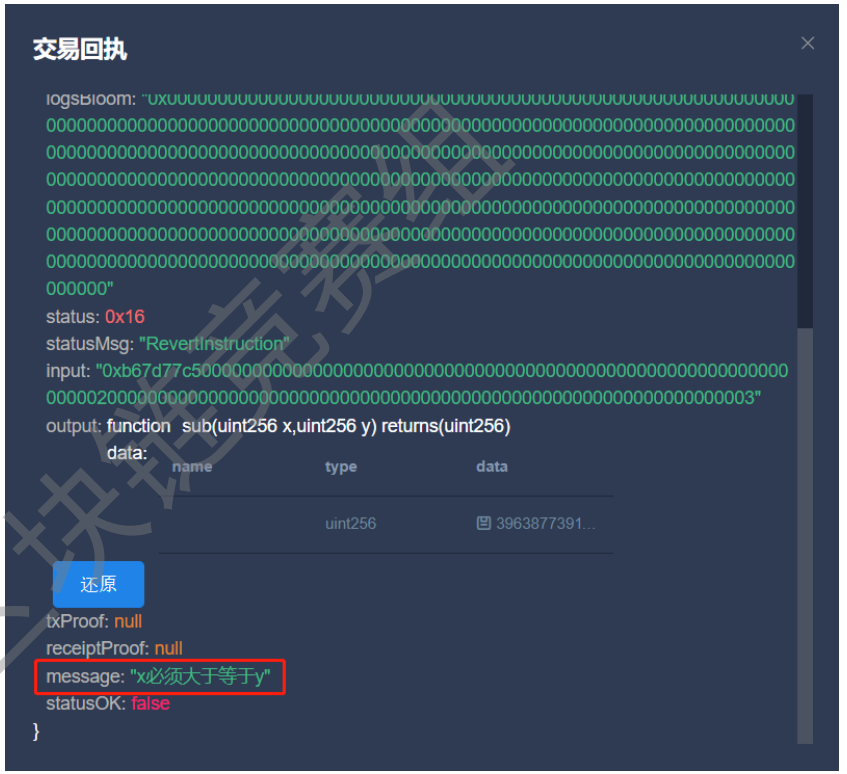
```
1  pragma solidity 0.6.10;
2
3
4  // 定义库合约SafeSub
5  library SafeSub {
6
7      // 使用internal限定符
8      function sub(uint x, uint y) internal pure returns(uint) {
9          require(x >= y, "x必须大于等于y");
10         return x - y;
11     }
12
13 }
```

Test.sol 保存 编译

```
1  pragma solidity 0.6.10;
2  import "./SafeSub.sol";
3
4
5  contract Test {
6
7      // 使用库合约的合约隐式继承了库合约
8      // 因此可以调用internal限定的函数
9      function sub(uint x, uint y) public returns(uint) {
10         return SafeSub.sub(x, y);
11     }
12 }
```

库合约

- 部署Test合约，调用sub函数，在没有使用继承的前提下仍然成功调用了SafeSub合约中internal限定符的sub函数



库合约

- using for，使用using A for B可以将库合约A中的所有函数添加到类型B上，当A中的函数被调用时，会将B类型作为第一个参数传入函数，这类似于其他面向对象编程中的this和self
- using A for *可以将库合约A中的函数添加到任意类型上

库合约

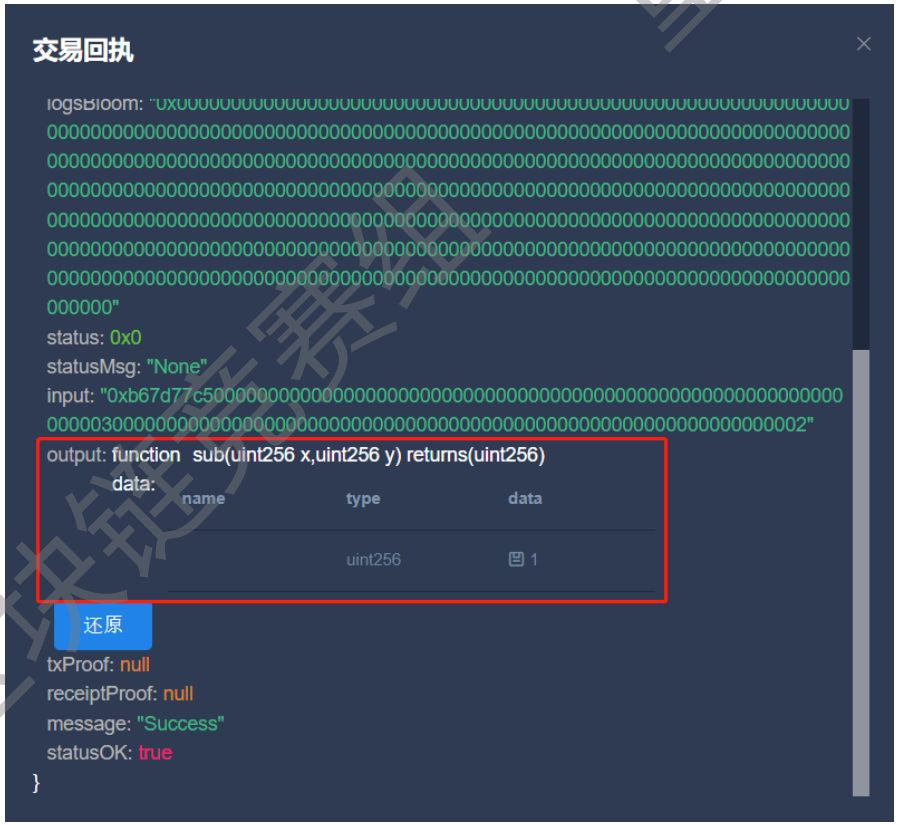
- using A for B 合约示例

Test.sol 保存 编译

```
1  pragma solidity 0.6.10;
2  import "./SafeSub.sol";
3
4
5  contract Test {
6
7      // 将SafeSub中的所有函数添加到uint类型上
8      using SafeSub for uint;
9
10     function sub(uint x, uint y) public returns(uint) {
11         // 使用x.sub(y)的方式调用
12         return x.sub(y);
13     }
14 }
```

库合约

- 部署合约，调用sub函数，查看运行结果



库合约

- using A for * 合约示例

SafeSub.sol 保存 编译 部署 合约调用

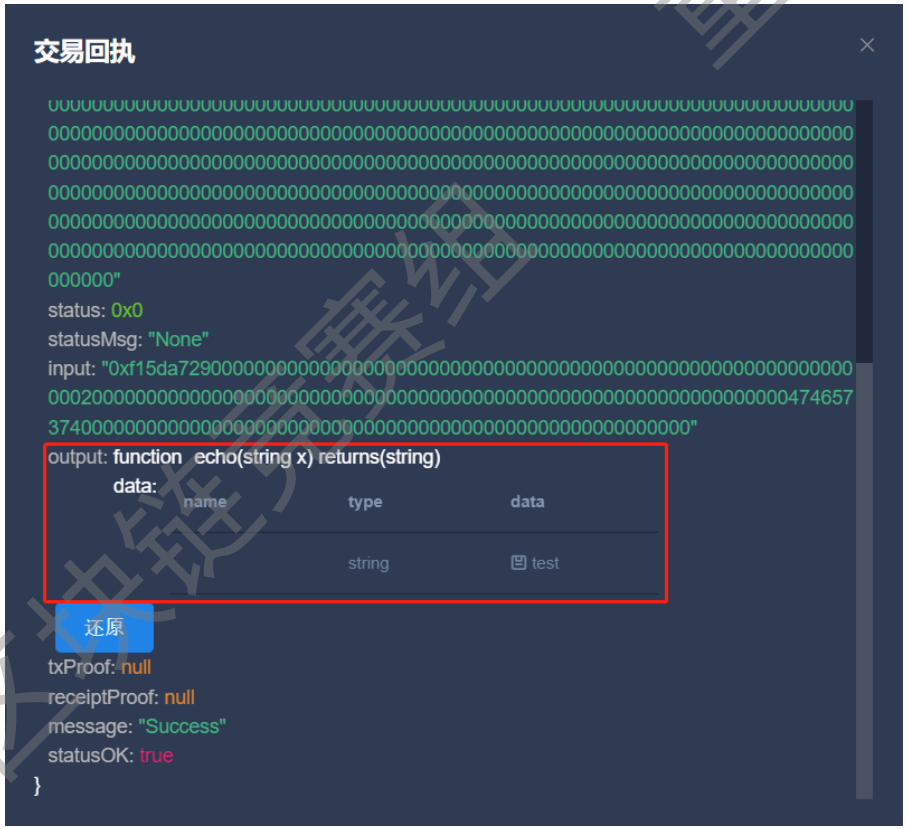
```solidity
pragma solidity 0.6.10;


// 定义库合约SafeSub
library SafeSub {

    // 使用internal限定符
    function sub(uint x, uint y) internal pure returns(uint) {
        require(x >= y, "x必须大于等于y");
        return x - y;
    }

    function echo(string memory x) internal pure returns(string memory) {
        return x;
    }

}
```

Test.sol 保存 编译 部署

```solidity
pragma solidity 0.6.10;
import "./SafeSub.sol";


contract Test {

    // 将SafeSub中的所有函数添加到任意类型上
    using SafeSub for *;

    function sub(uint x, uint y) public returns(uint) {
        // 使用x.sub(y)的方式调用
        return x.sub(y);
    }


    function echo(string memory x) public returns(string memory) {
        // 使用x.echo()的方式调用
        return x.echo();
    }

}
```

库合约

- 部署合约，调用echo函数，查看运行结果



常用库合约

- WeBASE-Front给提供了很多常用的工具合约，包括地址、加密、字符串处理、数学计算等模块，供开发时使用。
- 选择“合约管理”-“合约仓库”菜单，点击“合约工具箱”标签中的“预览和说明”进入合约工具箱。

库合约

- Address库合约，提供对地址的判断功能
  - isContract：判断地址是否是合约地址
  - isEmptyAddress：判断地址是否为空地址

Address.sol

```
1  pragma solidity ^0.4.24;
2
3  library Address {
4
5      function isContract(address addr) internal view returns(bool) {
6          uint256 size;
7          assembly { size := extcodesize(addr) }
8          return size > 0;
9      }
10
11     function isEmptyAddress(address addr) internal pure returns(bool){
12         return addr == address(0);
13     }
14 }
```

库合约

- LibString库合约，提供对字符串的快捷操作
  - bytesToString/bytes32ToString：将bytes/bytes32转换为字符串类型
  - stringToBytes/stringToBytes32：将字符串转换为bytes/bytes32类型

LibString.sol

```
8
9   pragma solidity >=0.4.24 <0.6.10;
10
11  library LibString {
12
13      using LibString for *;
14
15      function bytesToString(bytes memory _bytes) internal pure returns (string memory){}
22
23      function stringToBytes(string memory _string) internal pure returns (bytes memory){}
27
28
29      function bytes32ToString(bytes32 _bytes32) internal pure returns(string memory){}
45
46
47      function stringToBytes32(string memory source) internal pure returns(bytes32 result){}
```

库合约

- LibString库合约
  - compare/compareNoCase：比较两个字符串的大小（noCase不区分大小写）
  - equals/equalsNoCase：比较两个字符串是否相等（ noCase不区分大小写）
  - substr：根据索引获取子字符串
  - concat：拼接1-3个字符串

LibString.sol

```
74   function compare(string memory _self, string memory _str) internal pure returns (int8 _ret) {}
91
92   function compareNoCase(string memory _self, string memory _str) internal pure returns (int8 _ret) {}
119
120  function equals(string memory _self, string memory _str) internal pure returns (bool _ret) {}
133
134  function equalsNoCase(string memory _self, string memory _str) internal pure returns (bool _ret) {}
155
156  function substr(string memory _self, uint _start, uint _len) internal returns (string memory _ret) {}
177
178  function concat(string memory _self, string memory _str) internal returns (string memory _ret) {}
193
194  function concat(string memory _self, string memory _str1, string memory _str2)
195      internal returns (string memory _ret) {}
217
218  function concat(string memory _self, string memory _str1, string memory _str2, string memory _str3)
219      internal returns (string memory _ret) {}
```

库合约

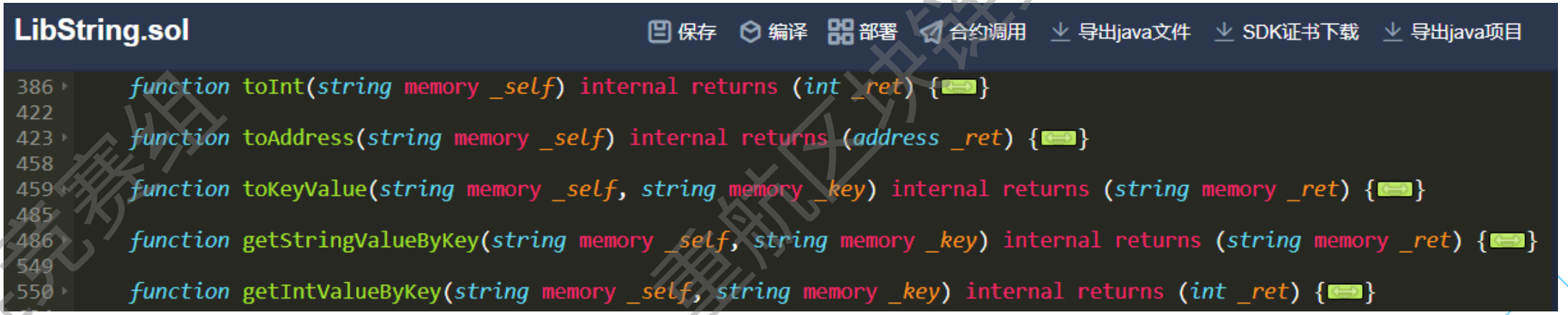
- LibString库合约
  - trim：删除字符串中左右两端的的空格/指定字符
  - indexOf：获取字符串中指定字的索引，offset表示从指定索引向后查询
  - split：将字符串按指定字符分割

LibString.sol　保存　编译　部署　合约调用　导出java文件　SDK证书下载　导出java项目

```
247     function trim(string memory _self) internal returns (string memory _ret) {}
277
278     function trim(string memory _self, string memory _chars) internal returns (string memory _ret) {}
326
327     function indexOf(string memory src, string memory value) internal pure returns (int) {}
330
331     function indexOf(string  memory src, string memory value, uint offset) internal pure returns (int) {}
345
346     function split(string memory src, string memory separator) internal pure returns (string[] memory splitArr) {}
```

库合约

- LibString库合约
  - toInt：将字符串格式数字转换为整型
  - toAddress：将字符串格式地址转换为地址类型
  - toKeyValue：将两个字符串转换为key:value的格式
  - getStringValueByKey：根据key从key:value格式中获取value
  - getIntValueByKey：根据key从key:value格式中获取value，并转为整型



库合约

- LibString库合约
  - toUppercase：将字符串转换为大写
  - toLowercase：将字符串转换为小写
  - keyExists：检查key:value字符串中是否存在指定key
  - inArray/inArrayNoCase：判断数组中是否存在指定元素/忽略大小写

LibString.sol　保存　编译　部署　合约调用　导出java文件　SDK证书下载

```
625     function toUppercase(string memory src) internal pure returns(string memory){}
636
637     function toLowercase(string memory src) internal pure returns(string memory){}
648
649     function keyExists(string memory _self, string memory _key) internal returns (bool _ret) {}
660
661     function inArray(string memory _self, string[] storage _array) internal returns (bool _ret) {}
670
671     function inArrayNoCase(string memory _self, string[] storage _array) internal returns (bool _ret) {}
680
681 }
```

库合约

- SafeMath库合约，处理了边界问题，提供更安全的数学计算
  - mul、div、sub、add、mod对应乘、除、减、加、取模运算

SafeMath.sol　保存　编译　部署　合约调用

```
1  pragma solidity 0.6.10;
2
3
4  library SafeMath {
5      function mul(uint256 a, uint256 b) internal pure returns (uint256) {...}
18
19     function div(uint256 a, uint256 b) internal pure returns (uint256) {...}
27
28     function sub(uint256 a, uint256 b) internal pure returns (uint256) {...}
34
35     function add(uint256 a, uint256 b) internal pure returns (uint256) {...}
41
42     function mod(uint256 a, uint256 b) internal pure returns (uint256) {...}
46 }
```

库合约

- Roles库合约，提供了快捷的角色管理
  - add：添加指定角色
  - remove：删除指定角色
  - has：判断是否存在指定角色

Roles.sol

```
pragma solidity ^0.4.24;

library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}
```

库合约

- Counters库合约，提供整型的原子计算
  - current：当前大小
  - increment：自增1
  - decrement：自减1

Counters.sol

```
pragma solidity ^0.4.25;
import "./SafeMath.sol";


library Counters {
    using SafeMath for uint256;

    struct Counter {
        // This variable should never be directly accessed by users of the libra
        // the library's function. As of Solidity v0.5.2, this cannot be enforce
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}
```

作业

1.什么是抽象合约，它有什么特点？

2.什么是接口合约，它有什么特点？

3.什么是库合约，它有什么特点？

4.导入库合约后，using A for B代表什么意思？

5.该库合约实现什么功能？

```
pragma solidity ^0.4.24;

library Address {

    function isContract(address addr) internal view returns(bool) {
        uint256 size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }

    function isEmptyAddress(address addr) internal pure returns(bool){
        return addr == address(0);
    }
}
```

谢谢