在合约中，一个合约继承了一个接口，如果合约还有接口未包含的函数A，则用接口创建合约实例的时候，本实例不能调用A函数

验证为：

 interface Idemo {

    function set(uint a) external;

    function get() external view returns(uint);

 }

import "./Idemo.sol";

contract demo is Idemo{

    uint num;

    function set(uint a) external{

        num= a;

    }

    function get() external view returns(uint){

        return num;

    }

    function add(uint a,uint b) public pure  returns (uint) {

        return a+b;

    }

}

import "./demo.sol";

contract verify is demo{

    // IERC721 BAYC = IERC721(0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D);

    Idemo demo = Idemo(0xd9145CCE52D386f254917e481eB44e9943F39138);

    uint a =1;

    uint b =2;

    uint result;

    function verifying() public  {

        // result = demo.add(a,b);//报错找不到函数

        demo.get();

    }

}

对于数组：constant修饰符不能修饰数组，另外memory修饰的数组可以由new创建，这是动态数组

对于使用oz的upgrade，数组必须在initialize函数里面赋值，直接声明并赋值不得行

对于不生成getter函数的变量，则可以设置为private或者internal，以节约合约字节码

Modifier可以接收参数，并根据此参数做出处理

对于public的静态数组：会生成getter方法，调用时需要传入数组下标

示例： contract Storage {

    uint256[4] public arry;

    constructor() {

        arry = [1,2,3,4];

    }

}

对于这样的arry数组，调用时注册ABI：function arry(uint256) public view returns(uint256)

调用时就直接传入uint256类型的数组下标就行了

内联汇编：

function at(address addr) public view returns (bytes memory code) {

  assembly {

      // 获取代码大小，这需要汇编语言

      let size := extcodesize(addr)

      // 分配输出字节数组 – 这也可以不用汇编语言来实现

      // 通过使用 code = new bytes(size)

      code := mload(0x40)

      // 包括补位在内新的 “memory end”

      mstore(0x40, add(code, and(add(add(size, 0x20), 0x1f), not(0x1f))))

      // 把长度保存到内存中

      mstore(code, size)

      // 实际获取代码，这需要汇编语言

      extcodecopy(addr, add(code, 0x20), 0, size)

  }

}

**code** 被使用为一个动态字节数组，因为返回类型声明为 **bytes memory**，另外mstore(code, size)这样是把大小存入code变量，因为动态字节数组第一个槽存的是数组大小，后面才是存的具体数据

另外动态数组和mapping的存储方式详见官方文档