

区块链集训营

二期

登链社区 - Tiny熊

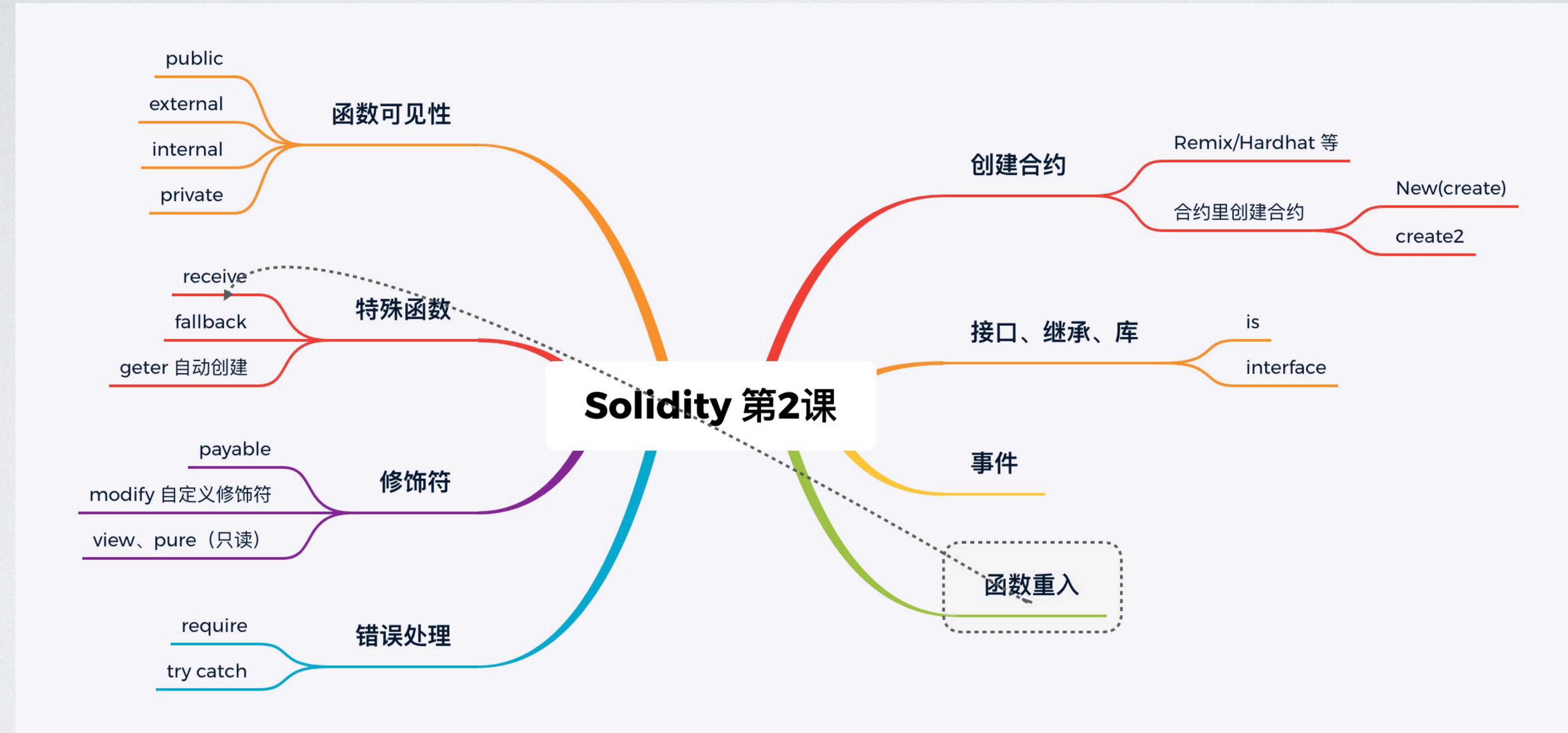
练习题

- 编写一个Bank合约：
 - 通过 Metamask 向Bank合约转账ETH
 - 在Bank合约记录每个地址转账金额
 - 编写 Bank合约 withdraw(), 实现提取出所有的 ETH

习题解答

- 理解合约作为一个账号、也可以持有资产
- msg.value / 如何传递 Value
- 回调函数的理解 (receive/fallback)
- Payable 关键字理解
- Mapping 使用
- 附加：Metamask 连 Hardhat 进行转账，MetaMask 导入 Hardhat 账号

第2周



SOLIDITY – 可见性

内外

内部

外部访问

内部及继承

- 使用public、private、external、internal 可见性关键字来控制变量和函数是否可以被外部使用。

```
pragma solidity >=0.8.0;

contract Available {
    function cal(uint a) public pure returns (uint b) { return a + 1; }
    function setData(uint a) internal { data = a; }
    uint public data;
}
```

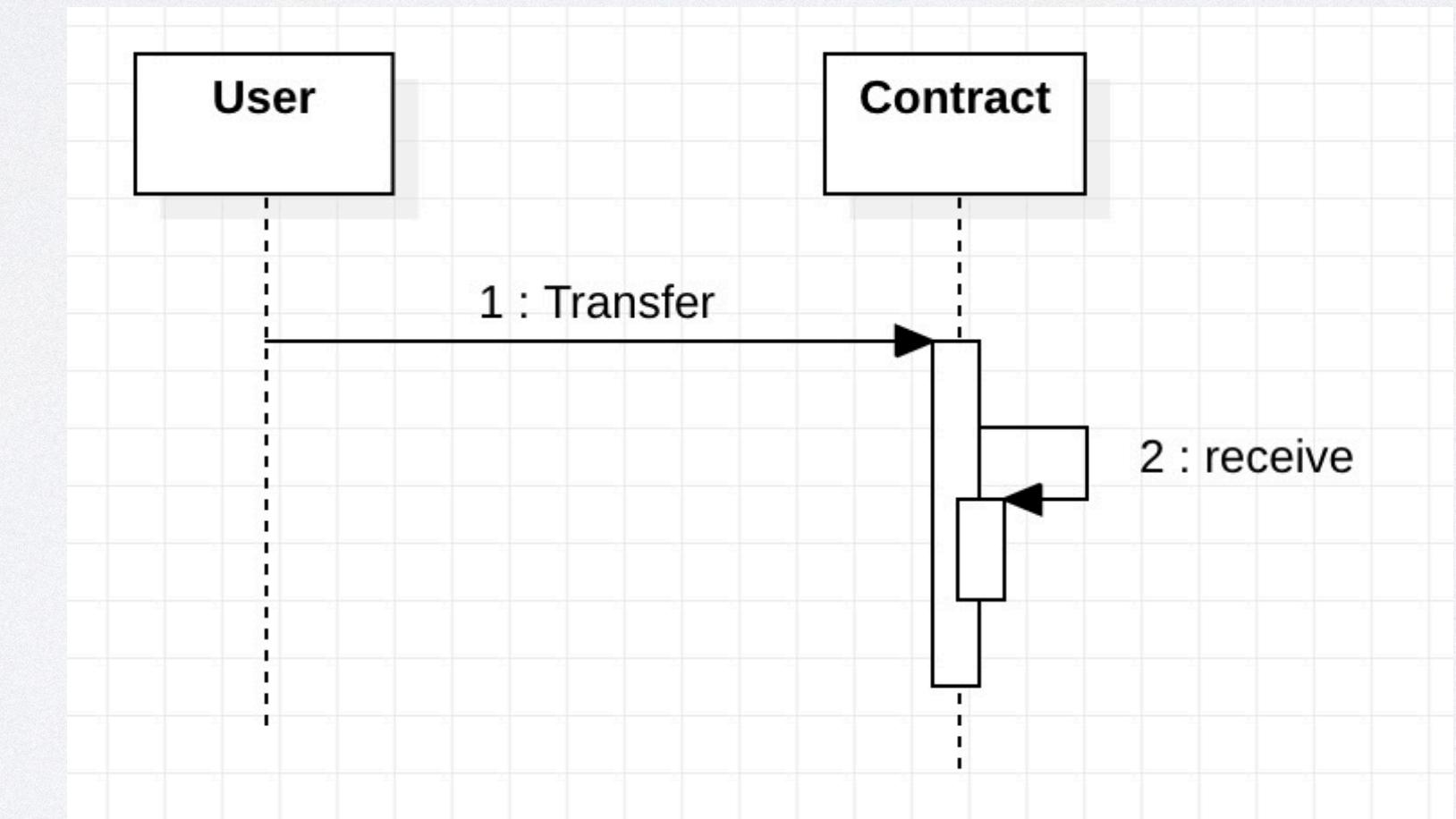
testAvailable.sol

SOLIDITY – 合约特殊函数

- 构造函数(constructor): 初始化逻辑
- getter 函数: 所有 public 状态变量创建 getter 函数
- receive 函数: 接收以太币时回调。
- fallback 函数: 没有匹配函数标识符时, fallback 会被调用, 如果是转账时, 没有receive也有调用fallback

SOLIDITY – 回调函数

- receive / fallback 充当回调函数作用
- 回调函数：有转账了，告诉我（合约）一下。



SOLIDITY – 修饰符

- View 修饰函数(视图函数)：不修改状态
testAvailable.sol
- Pure 修饰函数(纯函数)：不读取状态
- 用 payable 修饰函数，表示调用函数可以被支付ETH
 - 支付的 ETH 值，用 msg.value 获取

自定义修饰符 - 函数修改器

- 用 modifier 定义一个修改器

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}
```

- 用修改器修饰一个函数，用来添加函数的行为，如检查输入条件、控制访问、重入控制

```
function withdraw() public onlyOwner {
    // do something
}
```

第2周

testModifier

```
pragma solidity ^0.8.0;
contract testModifier {
    address public owner;
    uint private deposited;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }

    function withdraw() public onlyOwner {
        payable(owner).transfer(deposited);
    }
}
```

函数修改器修饰函数时，函数体被插入到“_”

SOLIDITY – 错误处理

- 在程序发生错误时的处理方式：EVM通过回退状态来处理错误的，以便保证状态修改的**事务性**
- assert()和**require()**用来进行条件检查，并在条件不满足时抛出异常
- revert("msg")：终止运行并撤销状态更改
- Error 定义错误

```
require(msg.sender == owner, "Not owner");
```

```
error NotOwner();
if(msg.sender != owner) revert NotOwner();
```

SOLIDITY – TRY/CATCH

- 当出现调用外部函数异常时，不想中断程序怎么办？
 - 0.6 以前只能通过底层调用 call()
- try/catch: 捕获合约中**外部调用**的异常
 - 注意：out of gas 错误不是程序异常，错误不能捕获。

第2周

```
pragma solidity ^0.8.0;

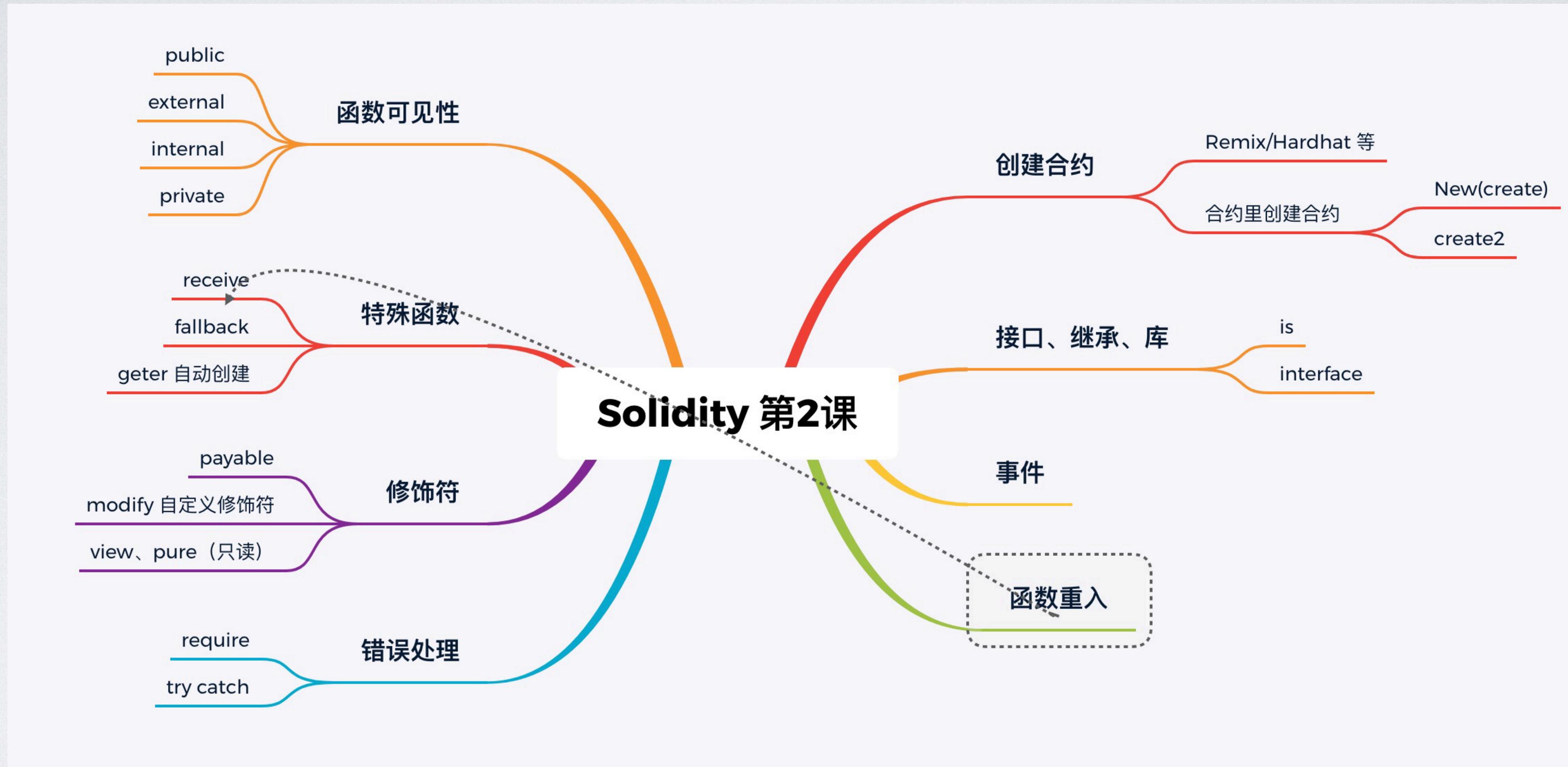
contract Foo {
    function myFunc(uint x) public pure returns (uint ) {
        require(x != 0, "require failed");
        return x + 1;
    }
}

contract trycatch {

    Foo public foo;
    uint public y;
    constructor() {
        foo = new Foo();
    }

    function tryCatchExternalCall(uint _i) public {
        try foo.myFunc(_i) returns (uint result) {
            y = result;
        } catch {
            ..
        }
    }
}
```

trycatch.sol



SOLIDITY – 创建合约

- 创建合约的几个方法：
 - 外部部署 (Remix/Hardhat/Truffle) Web3.js / Ethers.js
 - 合约使用New 关键字()
 - 最小代理合约 (复用代码)：
 - <https://eips.ethereum.org/EIPS/eip-1167>
 - <https://github.com/optionality/clone-factory>
 - Create2 (New + salt)
 - C c = **new** C{salt: _salt}();

SOLIDITY – 合约地址

- create 地址不易控制
 - 根据创建者 (sender) 的地址以及创建者发送过的交易数量 (nonce) 来计算确定
 - keccak256(rlp.encode([normalize_address(sender), **nonce**]))[12:]
- Create2 确定性的创建合约， 提前确定地址有什么好处?
 - keccak256(0xff ++ sender ++ **salt** ++ keccak256(init_code))[12:]

SOLIDITY – 继承

- 和大多数高级语言一样，Solidity 也支持继承
- 使用关键字 `is`
- 继承时，链上实际只有一个合约被创建，基类合约的代码会被编译进派生合约。
- 派生合约可以访问基类合约内的所有非私有（`private`）成员，因此内部（`internal`）函数和状态变量在派生合约里是可以直接使用的

第2周

test1s.sol

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor() {
        a = 1;
    }
}

contract B is A {
    uint public b ;
    constructor() {
        b = 2;
    }
}
```

在部署B时候，可以查看到a为1， b为2。

SOLIDITY – 继承、抽象合约

- abstract 抽象合约
 - 不能被部署，可包含没有实现的纯虚函数
- super：调用父合约函数
- virtual：表示函数可以被重写
- override：表示重写了父合约函数

testAbstract.sol

```
pragma solidity ^0.8.0;

abstract contract A {
    uint public a;
    function add(uint x) public virtual;
}

contract B is A {
    uint public b ;
    constructor() {
        b = 2;
    }

    function add(uint x) public override virtual  {
        b += x;
    }
}
```

SOLIDITY – 接口

- **类型声明**（函数的抽象），广泛用于合约之间的调用
- 无任何实现的函数
- 不能继承自其他接口
- 没有构造方法
- 没有状态变量

第2周

testInterface.sol

```
pragma solidity ^0.8.0;

contract Counter {
    uint public count;

    function increment() external {
        count += 1;
    }
}

interface ICounter {
    function count() external view returns (uint);
    function increment() external;
}

contract MyContract {
    function incrementCounter(address _counter) external {
        ICounter(_counter).increment();
    }

    function getCount(address _counter) external view returns (uint) {
        return ICounter(_counter).count();
    }
}
```

SOLIDITY – 库

- 与合约类似（一个特殊合约），是函数的封装，用于代码复用。
- 如果库函数都是 `internal` 的，库代码会嵌入到合约。
- 如果库函数有`external`或 `public`，库需要单独部署，并在部署合约时进行链接，使用委托调用
- 没有状态变量
- 不能给库发送 Ether
- 给类型扩展功能：Using lib for type; 如：`using SafeMath for uint;`

第2周

testLib.sol

```
pragma solidity ^0.8.0;

library SafeMath {
    function add(uint x, uint y) internal pure returns (uint) {
        uint z = x + y;
        require(z >= x, "uint overflow");

        return z;
    }
}

contract TestLib {
    using SafeMath for uint;

    function testAdd(uint x, uint y) public pure returns (uint) {
        return x.add(y);
    }
}
```

SOLIDITY – 链接外部库

```
const ExLib = await hre.ethers.getContractFactory("Library");
const lib = await ExLib.deploy();
await lib.deployed();

await hre.ethers.getContractFactory("TestExLib", {
  libraries: {
    Library: lib.address,
  },
});
```

SOLIDITY – 事件

- 合约与外部世界的重要接口，通知外部世界链上状态的变化
- 事件有时也作为便宜的存储
- 使用关键字 event 定义事件，事件不需要实现
- 使用关键字 emit 触发事件
- 事件中使用indexed修饰，表示对这个字段建立索引，方便外部对该字段过滤查找

第2周

testEvent.sol

```
pragma solidity ^0.8.0;

contract testEvent {
    constructor() {}

    event Deposit(address indexed _from, uint _value);

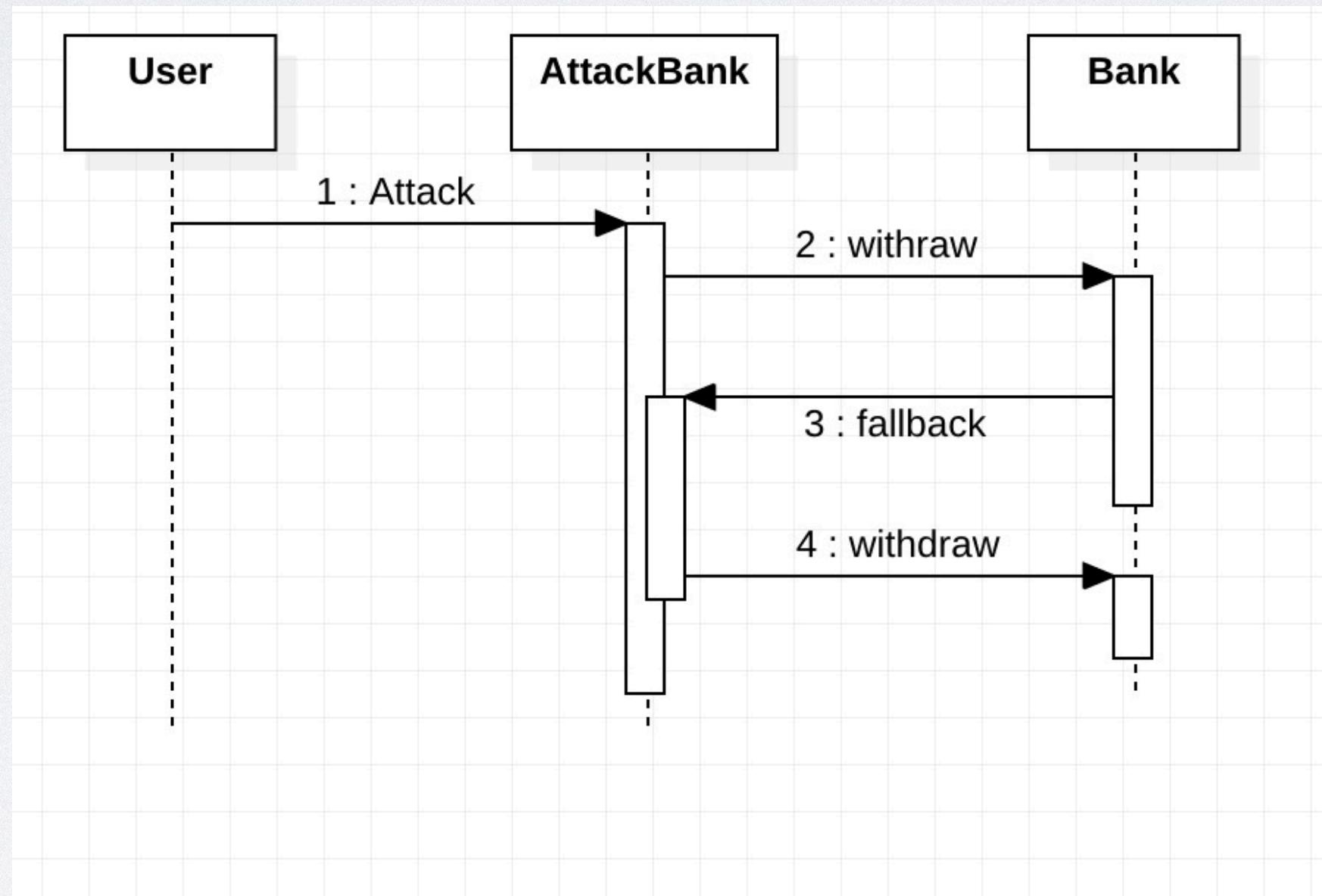
    function deposit(uint value) public {
        emit Deposit(msg.sender, value);
    }
}
```

事件将记录在日志之中，后面的课程将介绍如何从交易收据中解析日志

SOLIDITY – 重入攻击问题

- 调用外部函数时，要时刻注意重入问题：

- 重入



testReplay.sol

SOLIDITY – 防范重入攻击问题

- 先检查 - 再修改 - 最后交互 (checks-effect-interaction)
- 重入锁控制

testReplay.sol

推荐SOLIDITY学习资料

- <https://ethernaut.openzeppelin.com/>
- <https://cryptozombies.io/en/course/>
- <https://solidity-by-example.org/>
- <https://learnblockchain.cn/column/1>
- <https://decert.me/tutorial/solidity/intro/>

Q & A

练习题

- 编写合约Score，用于记录学生（地址）分数：
 - 仅有老师（用modifier权限控制）可以添加和修改学生分数
 - 分数不可以大于 100；
- 编写合约Teacher 作为老师，通过 IScore 接口调用修改学生分数。