知识点：调用其他合约的方法时，如果目标合约没有对应方法，则调用fallback函数。

用途：向合约地址转账，需要目标合约内有一个这样的函数：（带有payable关键字的匿名函数）

function () public payable { // code block }

code block可以有代码，但是要小（能够用32bytes以内的solidity abi描述）

2.

知识点：调用其他合约的方法，一般会改变msg.sender、一定会改变context（主要指其他合约不可操作的变量）。

解释：跨合约的调用，底层采用.call/.delegatecall/.callcode实现，它们都会修改函数执行的上下文环境为调用者环境（<.call>所在合约的环境）；.call会同时改变msg.sender、.delegatecall不会。

用途：其实这一点没有什么用途，因为很少直接使用.call——通常会使用创建一个合约实例的方法；.delegatecall则是为了方便使用其他合约的函数，可惜的是，在调用的函数中所做的修改不能直接作用到调用合约的参数，在项目中的用处不大，更不要说它为了避免安全风险要做很多额外的工作了。这一条主要是让你在掉进坑里的时候能快点爬出来。

3.

知识点：调用library库的方法，不会改变msg.sender和context。

解释：library库的使用，像是隐式继承，而solidity采用代码复制的方法实现继承，所以这些内容不会变化。

4.

知识点：ERC20标准代币（zeppelin）与地址的关系：

4.1. 最初的拥有者由代币合约部署时指定；

4.2. transfer/approve/transferFrom等函数都与msg.sender有关（2,3点在这个时候就很有用了），在一些复杂的调用过程中，时刻跟踪msg.sender的值，有助于快速掌握一次转账是谁在操作，或者一次approve是谁允许别人动自己的奶酪

5.

知识点：人为控制（或者说随意地指定）msg.sender和context，很难。

解释：跨合约的函数调用必然会改变context、继承又不能改变msg.sender，所以在项目中同时涉及token转移和自定义结构体传递的时候，很多东西就都注定了只能是那个样子的，就像是面对一个大范围的aoe，没有多少操作空间。