# [Solidity](https://so.csdn.net/so/search?q=Solidity&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_26769677/article/details/_blank)**安全**

#### **1. 尽量避免外部调用**

调用不受信任的外部合约可能会引发一系列意外的风险和错误。外部调用可能在其合约和它所依赖的其他合约内执行恶意代码。

#### **2. 清楚“send()”、“transfer()”、以及“call.value()”的效率和稳定性**

当转账Ether时，需要仔细权衡“someAddress.send()”、“someAddress.transfer()”、和“someAddress.call.value()()”之间的差别。

x.transfer(y)和if (!x.send(y)) throw;是等价的。send是transfer的底层实现，建议尽可能直接使用transfer。

someAddress.send()和someAddress.transfer() 能保证可重入安全 。 尽管这些外部智能合约的函数可以被触发执行，但补贴给外部智能合约的2,300 gas，意味着仅仅只够记录一个event到日志中。

someAddress.call.value()() 将会发送指定数量的Ether并且触发对应代码的执行。被调用的外部智能合约代码将享有所有剩余的gas，通过这种方式转账是很容易有可重入漏洞的，非常 不安全。

使用send() 或transfer() 可以通过制定gas值来预防可重入， 但是这样做可能会导致在和合约调用fallback函数时出现问题，由于gas可能不足，而合约的fallback函数执行至少需要2,300 gas消耗。  
  
在需要对外未知地址转账Ether时使用send() 或transfer()，已知明确内部无恶意代码的地址转账Ether使用call.value()()

#### **3.处理外部调用错误**

address.call()，address.callcode()， address.delegatecall()和address.send,这些底层方法不会抛出异常(throw)，只会在遇到错误时返回false。

contract calls （比如，ExternalContract.doSomething())）会自动传递异常，（比如，doSomething()抛出异常，那么ExternalContract.doSomething() 同样会进行throw） )。

如果你选择使用底层方法，需要通过检查返回值来对可能出现的错误进行处理。

#### **4.不要假设你知道外部调用的控制流程**

无论是使用raw calls 或是contract calls，如果这个ExternalContract是不受信任的都应该假设存在恶意代码。即使ExternalContract不包含恶意代码，但它所调用的其他合约代码可能会包含恶意代码。

#### **5.对于外部合约优先使用pull 而不是push**

为了最小化外部调用失败带来的损失，最好的做法是将外部调用函数与其余代码隔离，最终是由收款发起方负责发起调用该函数。（事先设置需要付给某一方的资产的值，表明接收方可以从当前账户撤回资金的额度，然后由接收方调用当前合约提现函数完成转账这种方法同时也避免了造成 gas limit相关问题。）

#### **6.标记不受信任的合约**

当你自己的函数调用外部合约时，你的变量、方法、合约接口的命名应该表明和他们交互可能是不安全的。

#### **7.使用assert()强制不变性**

当断言条件不满足时将触发断言保护 —— 比如不变的属性发生了变化。举个例子，代币在以太坊上的发行比例在代币的发行合约里可以通过这种方式得到解决。断言保护经常需要和其他技术组合使用，比如当断言被触发时先挂起合约然后升级。（否则将一直触发断言，你将陷入僵局）

断言保护 不是 严格意义的余额检测， 因为智能合约可以不通过deposit() 函数被 强制发送Ether

#### **8.正确使用assert()和require()**

require(condition)被用来验证用户的输入，如果条件不满足便会抛出异常，应当使用它验证所有用户的输入。 assert(condition) 在条件不满足也会抛出异常，但是最好只用于固定变量：内部错误或你的智能合约陷入无效的状态。

#### **9.小心整数除法的四舍五入**

所有整数除数都会四舍五入到最接近的整数。 如果需要更高精度，可以使用乘数，或存储分子和分母。

#### **10.记住Ether可以被强制发送到账户**

攻击者可以强制发送wei到任何账户，而且这是不能被阻止的（即使让fallback函数throw也不行）

攻击者可以仅仅使用1 wei（1 以太币 = 1000000000000000000 Wei）来创建一个合约，然后调用selfdestruct(victimAddress)。在victimAddress中没有代码被执行，所以这是不能被阻止的。

#### **11.不要假设合约创建时余额为零**

攻击者可以在合约创建之前向合约的地址发送wei。合约不能假设它的初始状态包含的余额为零

#### **12.记住链上的数据是公开的**

当开发一个依赖随机数生成器的应用时，正确的顺序应当是

（1）玩家提交行动计划，

（2）生成随机数，

（3）玩家支付。产生随机数是一个值得研究的领域；

当前最优的解决方案包括比特币区块头，hash-commit-reveal方案（比如，一方产生number后，将其散列值提交作为对这个number的“提交”，然后在随后再暴露这个number本身）和 RANDAO。  
- 选择hash-commit机制

#### **13.权衡Abstract合约和Interfaces**

Interfaces和Abstract合约都是用来使智能合约能更好的被定制和重用。Interfaces存在一些限制比如不能够访问storage或者从其他Interfaces那继承，通常这些使Abstract合约更实用。

#### **14.在双方或多方参与的智能合约中，参与者可能会“脱机离线”后不再返回**

不要让退款和索赔流程依赖于参与方执行的某个特定动作而没有其他途径来获取资金。比如，在石头剪刀布游戏中，一个常见的错误是在两个玩家提交他们的行动计划之前不要付钱。然而一个恶意玩家可以通过一直不提交它的行动计划来使对方蒙受损失 —— 事实上，如果玩家看到其他玩家泄露的行动计划然后决定他是否会损失（发现自己输了），那么他完全有理由不再提交他自己的行动计划。这些问题也同样会出现在通道结算。

当这些情形出现导致问题后：（1）提供一种规避非参与者和参与者的方式，可能通过设置时间限制，和（2）考虑为参与者提供额外的经济激励，以便在他们应该这样做的所有情况下仍然提交信息。

#### **15.简写Fallback函数**

Fallback函数在合约执行消息发送没有携带参数（或当没有匹配的函数可供调用）时将会被调用，而且当调用 .send() or .transfer()时，只会有2,300 gas 用于失败后fallback函数的执行（合约收到Ether也会触发fallback函数执行）。如果希望能够监听.send（）或.transfer（）接收到Ether，则可以在fallback函数中使用event（让客户端监听相应事件做相应处理）。谨慎编写fallback函数以免gas不够用。