

# 以太坊智能合约快速入门

## 学习智能合约编程的目的

就像1998年的互联网和2008年的移动应用，区块链正处在颠覆旧世界的前夜，而其中最激动人心的前沿领域之一莫过于智能合约了。智能合约（smart contract）是运行在去中心化区块链上的代码，它能完成验证、决策、存储等各种功能，自动执行约定好的条款。在以太坊（Ethereum）等公共链、摩根大通（JP Morgan Chase）Quorum等企业链上，智能合约都是至关重要的核心功能。

你的本职工作可能还不涉及智能合约，但紧跟时代步伐的程序员不该对这门新兴技术一无所知。智能合约的运行环境与传统的桌面程序、web应用乃至移动app都大相径庭。学习它可以让我们换一种思维模式去理解、设计和开发代码，成为能力更全面的专业程序员。

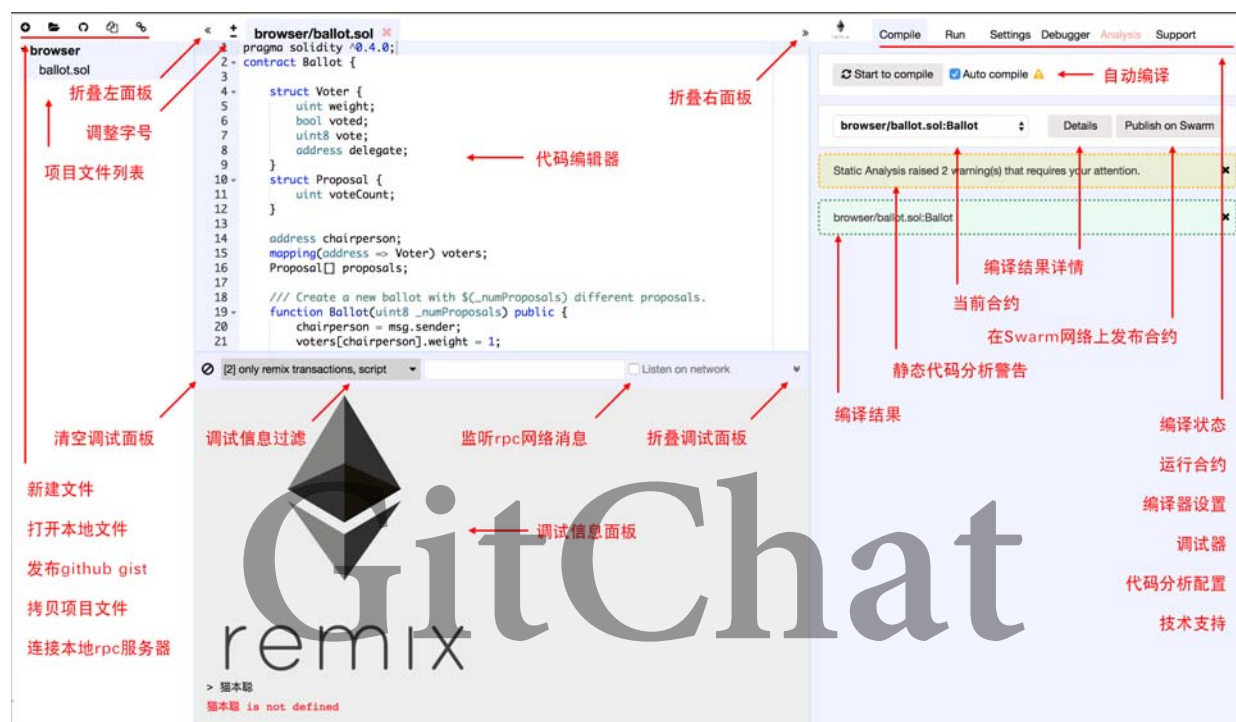
## 学习以太坊智能合约的价值

比特币就具备智能合约的能力，但它的脚本语言能实现的逻辑相当有限，连循环语句也不支持（这倒不是中本聪的疏漏，而是在灵活性和简洁性之间作的权衡）。以太坊的功能则强大许多，具备一个图灵完备的虚拟机EVM。Tezos、EOS等新生币也有类似的虚拟机，但它们的影响力和生态环境还远远不及以太坊。另外，Qtum、RSK、Quorum等平台的虚拟机都与EVM兼容。因此以太坊是目前学习智能合约编程的首选平台。掌握它之后，你不仅能写出实用的、跨平台智能合约，还能触类旁通，快速了解其他平台，或者研发出更优秀的区块链。

用任何文本编辑器加命令行就能开发Solidity程序。Ethereum/Mist客户端里也配备了Solidity编译器。但对于初学者而言，猫本聪强烈推荐Remix。它是一款非常好用的在线编程环境，不必安装任何本地软件就能零阻力上手。以后的课程里还会介绍truffle等框架和测试先行的开发模式（TDD），助你写出更复杂、安全的智能合约。

## 熟悉Remix环境

用浏览器访问 <https://remix.ethereum.org> 会看到以下的主界面：



首次运行时，Remix会打开一个ballot.sol的样例文件（Solidity的文件扩展名通常用.sol）。ballot.sol演示了投票合约的基本实现方法，暂时不必理会。

右侧面板的Compile页面显示的是Solidity编译结果和警告。默认设置下，我们每次修改代码都会触发自动编译（Auto compile），不必手工点击“Start to compile”按钮。编译成功时下方会显示绿框；错误会出现在红框里；黄框则表示警告。

**Compile****Run****Settings****Debugger****Analysis****Support**

Start to compile

☒ Auto compile

browser/ballot.sol:Ballot



Details

Publish on Swarm

Static Analysis raised 2 warning(s) that requires your a

browser/ballot.sol:Ballot



有些警告（比如样例代码的两个gas requirement warnings）是误报，不必在意。如果你有编译结果洁癖症，不能容忍警告，那么请切换到Analysis页面，按下图所示关闭gas costs检查：

# GitChat

**Security**

- ☒ Transaction origin: Warn if tx.origin is used
- ☒ Check effects: Avoid potential reentrancy bugs
- ☒ Inline assembly: Use of Inline Assembly
- ☒ Block timestamp: Semantics maybe unclear
- ☒ Low level calls: Semantics maybe unclear
- ☒ Block.blockhash usage: Semantics maybe unclear

关闭误报的gas警告

**Gas & Economy**

- ☐ Gas costs: Warn if the gas requirements of functions are too high.
- ☒ This on local calls: Invocation of local functions via this

**Miscellaneous**

- ☒ Constant functions: Check for potentially constant functions
- ☒ Similar variable names: Check if variable names are too similar

**Run**☒ Auto run

Compile页面之外，最常用的是Run页面。我们在这里可以创建合约、调用合约函数、查看运行状态：

remix

CompileRunSettingsDebuggerAnalysisSupport

Environment

JavaScript VM

运行环境

Account

0xca3...a733c (100 ether)

测试帐号

复制帐号地址

Gas limit

3000000

燃料上限

Value

0

初始以太

browser/sleepism.sol:Sleepism

当前合约

At Address

Enter contract's address - i.e. 0x60606...

合约地址

Create

创建合约

0 pending transactions

Address 下拉框中的是系统自动生成的五个钱包地址，每个各含 100 以太。“以太”（ether）是以太坊的货币单位，缩写是 ETH。一个 ETH 等于十的十八次方个最小单位——wei（“维”）。此处的钱包和以太都是模拟的，在 Remix 环境之外没有任何价值。

在以太坊主链上，无论部署还是运行合约都需要花费真正的以太币。每一步操作都会生成一个交易，等矿工把交易打包入块后才能得到确认。而在 Remix 的仿真环境下，我们可以不费分毫地在本地内存中测试程序，所有交易也能迅速得到确认。

这个页面下的运行环境（environment）、燃料上限（gas limit）和初始以太（value）都不需要改默认值。

[Compile](#)[Run](#)[Settings](#)[Debugger](#)[Analysis](#)[Support](#)

Your current Solidity version is  
0.4.18+commit.9cf6e910.Emscripten.clang

Select new compiler version



☐ Text Wrap

☐ Enable Optimization

写下本文时0.4.18+commit.9cf6e910是最新的稳定版。你若是喜爱紧随潮流，可以在下拉框里选用0.4.19的nightly版本（其实差异很小）。

Settings页面下的文本换行（text wrap）和代码优化（enable optimization）两个选项都不需要改默认值。

Debugger和Support页面目前用不到，可以忽略。

以上某些以太坊术语可能会让初学者一头雾水，但那暂时不重要，不妨碍我们学写第一个合约程序。成功运行合约后再回头解释这些概念会更容易。

## 第一个合约

科普文《[分叉、睡教和SegWit2X](#)》为了解释软硬分叉的区别举了个宗教的例子：猫本聪自封教主，创立“睡教”。初始教规只有一条：每天必须睡足450分钟以上才能领取一枚睡币。让我们来把它试写成一个Solidity合约。

contract Sleepism {}定义了一个名为“Sleepism”（睡教）的合约（contract）。合约相当于面向对象编程语言中的类（class），用来封装相关的数据和函数。

让我们往空合约里添加几个基本功能：

```
pragma solidity ^0.4.18;

contract Sleepism {
    string public constant hierarch = "Nekomoto";
    uint minDailySleep; // minutes

    function Sleepism() public {
        minDailySleep = 450;
    }

    function hasEnoughDailySleep(uint dailySleep) public view
returns (bool) {
        return dailySleep >= minDailySleep;
    }
}
```

首先声明一个叫hierarch（教主）的字符串常量，满足教主幼稚的虚荣心：

```
string public constant hierarch = "Nekomoto";
```

Solidity是强类型（strongly-typed）语言，变量和常量声明要以数据类型开头。string代表字符串类型。

public修饰符表示hierarch对外可见。具体地说，Solidity编译器会自动生成一个不带参数的名为hierarch()的公开getter函数，直接返回hierarch的值“Nekomoto”。但要注意，与Java之类的传统语言很不同，编译器并不会生成setter函数。也就是说，外部不能直接修改一个变量的值，哪怕它带有public属性。

constant修饰符与C++中的const很像，把hierarch标记为一个常量，hierarch初始化

细心的读者会注意到，minDailySleep的标注中既没有public也没有constant。所以它对外不可直接访问，而且值可变，这为教规的变更留下了余地。

```
function Sleepism() public {  
    minDailySleep = 450;  
}
```

函数声明以function开始。名字与合约名相同的函数是合约的构造函数，只在创建合约时运行一次。构造函数不是必须定义的。在以上样例中，我们给minDailySleep变量赋初始值450。

随后定义的hasEnoughDailySleep函数用来判断每天的睡眠时间是否足够：

```
function hasEnoughDailySleep(uint dailySleep) public view  
returns (bool) {  
    return dailySleep >= minDailySleep;  
}
```

函数hasEnoughDailySleep接受一个uint类型的参数dailySleep。public修饰符表明函数公开，可以被外部调用。returns (bool)表示函数返回一个布尔值。函数体非常简单，如果输入参数大于等于minDailySleep则返回true，否则返回false。

紧随public的view也是个修饰符，表示函数会读取合约变量（例子中的minDailySleep）但不会修改合约的任何状态（例如重新赋值任何变量）。如果一个函数不仅不修改状态，连合约变量也不访问，那么它是个所谓的纯函数，可以用pure修饰符描述。view和pure是Solidity 0.4.16引入的新关键字，取代了先前的constant函数修饰符。

好了，如果没有输入错误的话，你会看到以下的绿框提示：



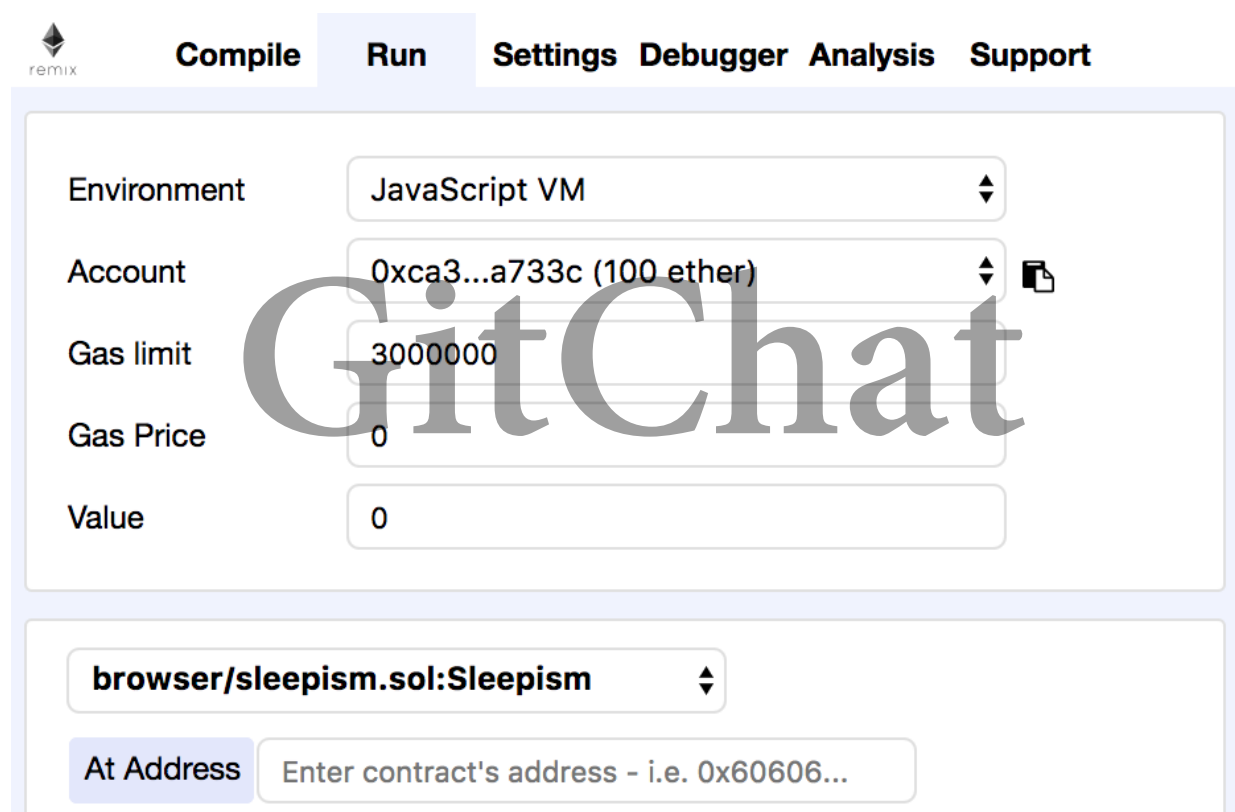


以太坊中的燃料（gas）并不是另一种币，只是计量合约运算量的一个单位。开销小的操作燃料费低，比如两个整数的加法只要3 gas。而开销大的操作则比较昂贵，比如往合约状态里写入一个整数虽然运算量极小，但会占用宝贵的区块链状态空间，所以耗费高达20000 gas。

运行合约时累积的gas数乘以你自己指定的燃料单价（gas price）就等于最终向矿工支付的运算报酬，单位是wei（以太坊的最小货币单位）。如果它低于你预设的燃料上限（gas limit）与燃料单价的乘积，多余部分会退还给原地址。万一超过上限就会发生燃料耗尽的错误，合约状态被回滚，白白付了交易费却什么也得不到。

以太坊的燃料机制迫使用户为合约的创建和运行付出代价，有效防止了区块链被垃圾合约填满空间，也能中断无限循环，防止节点白白浪费计算资源。但它也增加了复杂性，让我们在执行合约时要小心控制燃料参数，写合约时也要考虑如何节省燃料。

现在我们切换到Run页面，准备部署合约：



The image shows the 'Run' tab in the Remix IDE. The interface includes a top navigation bar with 'Compile', 'Run', 'Settings', 'Debugger', 'Analysis', and 'Support'. The 'Run' tab is active, displaying a form for deploying a contract. The form has the following fields:


- Environment:** A dropdown menu set to 'JavaScript VM'.
- Account:** A dropdown menu showing '0xca3...a733c (100 ether)' with a copy icon to the right.
- Gas limit:** A text input field containing '3000000'.
- Gas Price:** A text input field containing '0'.
- Value:** A text input field containing '0'.

Below these fields, there is a section for the contract to be deployed:

- A dropdown menu showing 'browser/sleepism.sol:Sleepism'.
- A button labeled 'At Address' next to a text input field containing the placeholder 'Enter contract's address - i.e. 0x60606...'.

点击“Create”按钮后，Remix会自动发起一笔交易，创建我们的合约。这一切都发生在内存的仿真环境里，并不涉及真正的区块链，所以确认速度极快。燃料费会从当前测试地址0xca3...a733c中扣除，当然那并不消耗真的以太币。

成功创建合约后的界面：



CompileRunSettingsDebuggerAnalysisSupport

EnvironmentJavaScript VM

Account0xca3...a733c (99.9999999999998227)

Gas limit3000000

Gas Price0

Value0

browser/sleepism.sol:Sleepism

At AddressEnter contract's address - i.e. 0x60606...

Create

0 pending transactions

browser/sleepism.sol:Sleepism at 0x692...77b3a (memory)

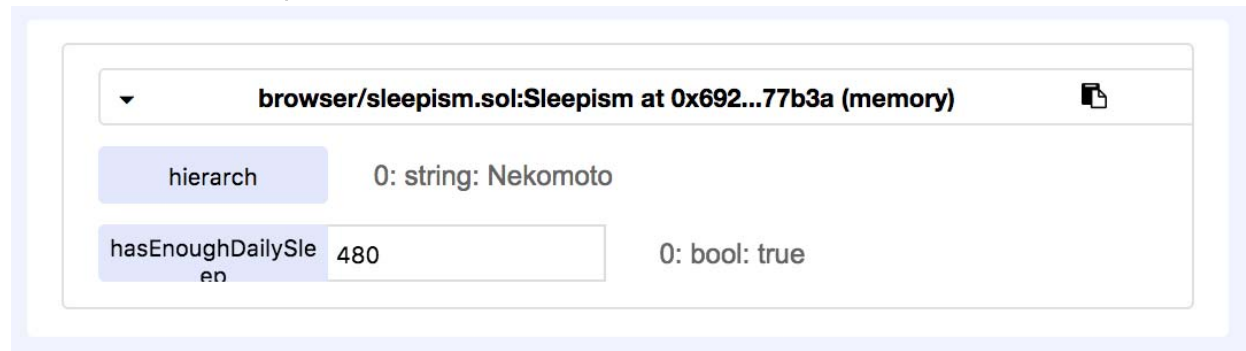
hierarch

0: string: Nekomoto

hasEnoughDailySleep

uint256 dailySleep0: bool: false

450，所以返回true)。：



恭喜你！你已经成功创建、运行了一个（还不那么智能）的合约！

## 状态修改与隐式变量

让我们回到代码编辑器，给合约添加功能。

现在的合约在构造函数里写死了每天450分钟的睡眠下限，没有灵活性。教主打算动态调整这个值怎么办呢？聪明的读者一定想到了，可以添加一个这样的setter函数：

```
function setMinDailySleep(uint dailySleep) public {  
    minDailySleep = dailySleep;  
}
```

setMinDailySleep函数接受一个与minDailySleep同类型的新值，赋给minDailySleep，不返回任何值。public修饰符让我们可以从外部调用它。由于这个函数修改了合约状态，所以不能用view或pure修饰符声明。

新的需求来了。由于一天总共才1440分钟，所以函数应该在赋值前检查输入参数，忽略越界的值：

```
function setMinDailySleep(uint dailySleep) public {
```

```

function Sleepism() public {
    minDailySleep = 450;
    owner = msg.sender; // <- NEW
}
...

```

address是Solidity基本类型之一，用于存储钱包或合约的地址。声明地址变量owner后，我们在Sleepism的构造函数中给它赋值：owner = msg.sender; 这里的msg无需声明，它是系统隐式传入的一个全局参数，包含了当前调用者的地址（.sender）、传入的以太数量（.value）、剩余的燃料量（.gas）等信息。由于合约将由教主创建，所以owner中会记录教主的以太坊地址。

接着我们只要在setter函数中添加地址检查逻辑，就能保证只有owner可以修改minDailySleep了：

```

function setMinDailySleep(uint dailySleep) public {
    if (msg.sender == owner && dailySleep <= 1440) {
        minDailySleep = dailySleep;
    }
}

```

有的JavaScript程序员可能习惯用三个等号“===”做等价判断。Solidity里没有这个操作符，因为它不允许对类型不兼容的值作运算。

完整代码如下：

```

pragma solidity ^0.4.18;

contract Sleepism {
    string public constant hierarch = "Nekomoto";
    uint minDailySleep; // minutes
    address owner; // <- NEW

    function Sleepism() public {

```

```
}  
}
```

## 事件

现在的setMinDailySleep函数不返回任何值。调用它后我们不能立刻知道是否成功更新了minDailySleep。一种解决方法当然是给函数加上返回值。另一种更灵活的方式是记录事件（events）。我们不仅能在无返回值的构造函数里记录事件，还能在一个函数里记录多个事件。它可以算是Solidity里最接近print或console.log的输出机制了。




使用事件前需要先在合约体里用event关键字声明。与函数一样，事件也可以带参数：

```
event MinDailySleep(uint newValue);
```

以上这行代码声明了一个名为MinDailySleep的事件类型，附带一个uint型的参数（参数名可以省略，但为了代码可读性，建议起名）。声明完毕后，每当minDailySleep的值变化时，我们可以像调用函数一样记录事件。完整代码：

```
pragma solidity ^0.4.18;  
  
contract Sleepism {  
    string public constant hierarch = "Nekomoto";  
    uint minDailySleep; // minutes  
    address owner;  
  
    event MinDailySleep(uint newValue); // <- NEW  
  
    function Sleepism() public {  
        minDailySleep = 450;  
        MinDailySleep(minDailySleep); // <- NEW  
        owner = msg.sender;  
    }  
}
```

重新部署合约后，在调试面板中点“Details”按钮，我们能在展开的详细信息中看到新记录的MinDailySleep事件和附带的参数：

decoded output	-
logs	  [ { "topic": "b3185641e97a4b936237bd124f7340603162b8d59639a92d31df71b67ff43834", "event": "MinDailySleep", "args": [ "450" ] } ]
value	 0 wei

## 收款与转账

智能合约最有用也最独特的功能之一就是自动收款和转账。我们只要给Sleepism合约添加几行程序，就让教主接受教徒的捐款了：

```
uint public totalDonation;
event Donation(address from, uint value);

function donate() public payable {
    if (msg.value > 0) {
        totalDonation += msg.value;
        Donation(msg.sender, msg.value);
        owner.transfer(msg.value);
    }
}
```

uint public totalDonation;定义了一个无符号整型的公共变量totalDonation，记录总捐款值，单位wei。

event Donation(address from, uint value);声明了一个Donation事件，记录每笔捐款的汇款

```

uint public totalDonation;
uint minDailySleep; // minutes
address owner;

event MinDailySleep(uint newValue);
event Donation(address from, uint value);

function Sleepism() public {
    minDailySleep = 450;
    MinDailySleep(minDailySleep);
    owner = msg.sender;
}

function hasEnoughDailySleep(uint dailySleep) public view
returns (bool) {
    return dailySleep >= minDailySleep;
}

function setMinDailySleep(uint dailySleep) public {
    if (msg.sender == owner && dailySleep <= 1440) {
        minDailySleep = dailySleep;
        MinDailySleep(minDailySleep);
    }
}

function donate() public payable {
    if (msg.value > 0) {
        totalDonation += msg.value;
        Donation(msg.sender, msg.value);
        owner.transfer(msg.value);
    }
}

```

测试payable函数的步骤稍有不同。让我们先重新部署合约。创建成功后，先在Account列表中另选一个地址（不然相当于教主自己给自己捐款），然后在Value一栏里填上捐款金额（比如1 ether），然后点击 donate。



Compile

Run

Settings Debugger Analysis Support

Environment

JavaScript VM

Account

0xca3...a733c (96.99999999999993152)

Gas limit

3000000

Gas Price

0

Value

1

browser/sleepism.sol:Sleepism

At Address

Enter contract's address - i.e. 0x60606...

Create

0 pending transactions



browser/sleepism.sol:Sleepism at 0x0c2...739ef (memory)



totalDonation

0: uint256: 0

hierarch

0: string: Nekomoto

hasEnoughDailySleep

uint256 dailySleep

donate

setMinDailySleep

uint256 dailySleep





十秒确认后就能得到合约的地址了。下图是用MyEtherWallet部署合约的示例。

MyEtherWallet 3.10.6.0 English Gas Price: 21 Gwei Network: Ropsten (MyEtherWallet)

New Wallet Send Ether & Tokens Swap Send Offline Contracts ENS Check TX Status View Wallet Info Help

## Interact with Contract or Deploy Contract

**Byte Code**

```
6060604052341561000f57600080fd5b6101c26001819055507fb3185641e97a4b936237bd
124f7340603162b8d59639a92d31df71b67ff4383460015460405180828152602001915050
60405180910390a133600260006101000a81548173fffffffffffffffffffffffffffffffff
ffffffff021916908373fffffffffffffffffffffffffffffffffffffffff16021790555061
039b806100a16000396000f30060606040526004361061006d576000357c01000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
61007257806312243081146100ad578063ed88c68e146100d0578063ee2ac05f146100da57
8063f50bab7314610103575b600080fd5b341561007d57600080fd5b610093600480803590
```

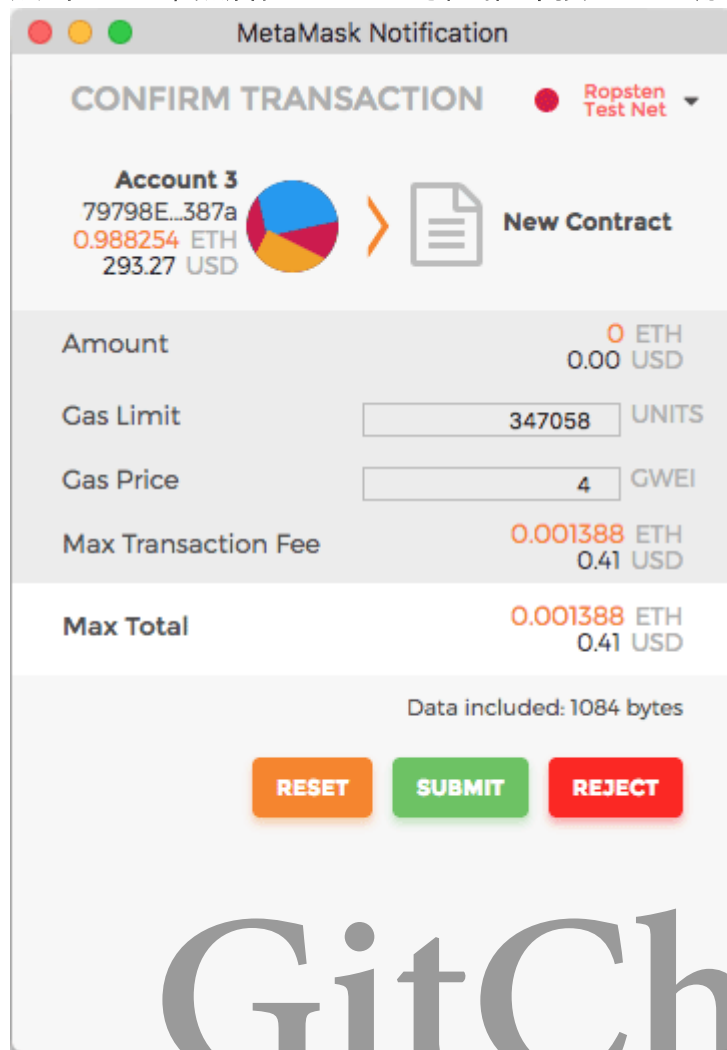
**Gas Limit**

**Sign Transaction**

第二种方法（快捷但需要安装额外软件）：

1. 在Chrome浏览器中安装Metamask插件（metamask.io）。
2. 在Remix的Run页面下，把Environment改成Injected Web3。

3. 点击Create，然后在Metamask弹出框中按Submit确认交易。



## 结语

恭喜你！你已经具备最基本的Solidity技能了。智能合约学习之路还很漫长，猫本聪建议你通读Solidity文档（<https://solidity.readthedocs.io>），阅读github上的各种开源合约，多多练习，成长为一名未来就业市场上炙手可热的智能合约工程师。

—— 猫本聪 ——