实验六预习：Remix

Solidity官方Remix： https://remix.ethereum.org

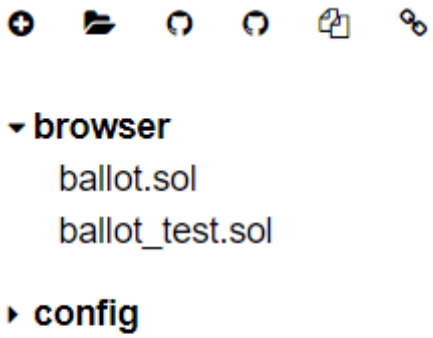
Solidity IDE中文版Remix由国内CDN加速，访问地址：http://remix.hubwiz.com。

本教程虽以中文版为例做对Remix进行介绍，但仍建议实验环境优先使用官方Remix，中文版仅作为无法打开时的备选方案。

Solidity IDE Remix为左中右三栏布局，左面板为Remix文件管理器，中间为文件编辑器， 右侧为开发工具面板：

1、Solidity IDE Remix文件管理器

Remix左面板中的文件管理器，用来列出在浏览器本地存储中保存的文件，分为browser和config两个目录， 当你第一次访问Remix的时候，在browser目录下有两个预置的代码：ballot.sol合约以及对应的单元测试 文件ballot_test.sol，点击文件名就可以在中间的文件编辑器中查看并编辑代码：



Remix文件管理器顶部的工具栏提供创建新文件、上传本地文件、发布gist等快捷功能，你可以将鼠标移到 相应的图标处停顿，然后查看功能的浮动提示信息。

为了后续功能的学习，你可以点击左上角的`+`创建一个新的solidity合约文件，在弹出的对话框中，将文件命名为hello.sol：

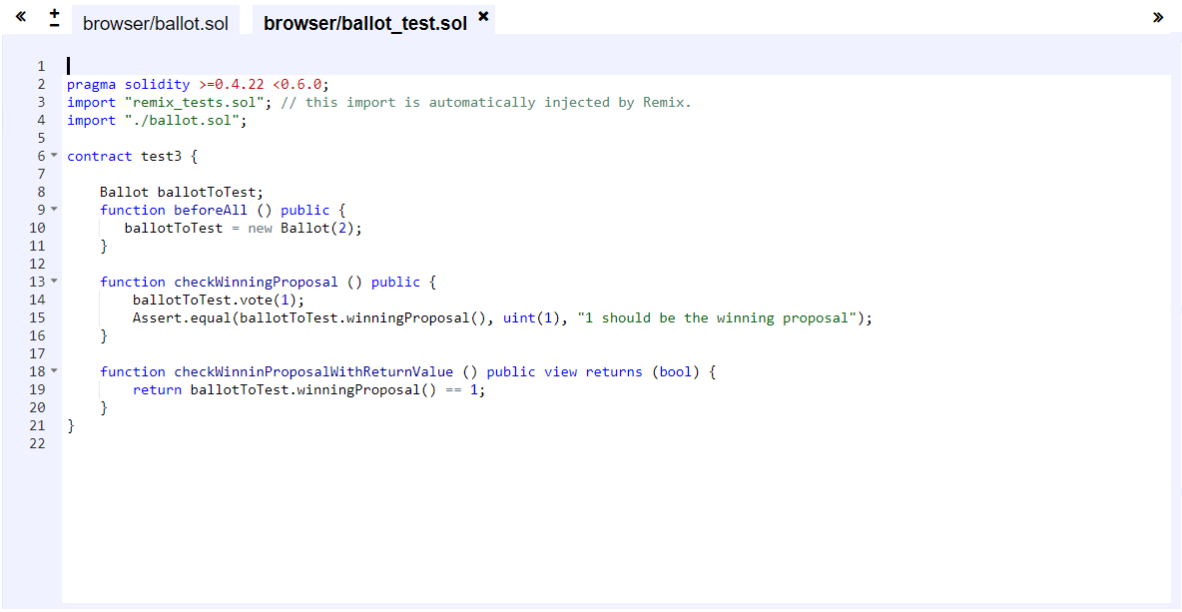
点击[ok]按钮后，你就可以看到在左面板的文件管理其中browser目录下出现了`hello.sol`文件名， 同时在中间区域的文件编辑器中自动打开了这个新创建的文件等待编辑，现在它还是空的，我们将在下面编写简单的Solidity代码。

2、Solidity IDE Remix编辑器及终端

Solidity IDE Remix中间区域为上下布局，分别提供文件编辑功能和终端访问功能。

2.1 Remix文件编辑器

Solidity IDE Remix中间区域上方的文件编辑器支持同时打开多个文件，当前激活的文件，其文件名以粗体显示：



Remix文件编辑器顶部左右两侧的箭头，分别用来切换左右面板的显示与隐藏；左上角的`+`和`-`， 分别用来放大或缩小编辑器里的文本字体大小。

现在我们激活`hello.sol`文件，然后输入简单的合约代码：

```
pragma solidity ^0.5.1;
contract Hello{
    function echo(string memory text) public pure returns(string memory) {
        return text;
    }
}
```

基本上这是最简单的以太坊合约了，它只有一个`echo()`方法，作用就是把输入的字符串 再原样返回。

2.2 Remix终端

Solidity IDE Remix中间区域下方为终端，可以输入JavaScript命令与Remix IDE或区块链节点交互：

Remix终端内置了web3.js 1.0.0、ether.js、swarmgy以及当前载入的Solidity编译器，因此你可以 在终端内使用熟悉的web3 API与当前连接的区块链节点交互。

Remix终端同时也内置了remix对象，可以利用它来脚本化地操作Solidity Remix IDE，例如载入指定url的gist，或者执行当前显示的代码。将终端显示向上滚动到开始位置，就可以看到remix对象的 常用方法描述。

Remix终端的另一个作用是显示合约执行或静态分析的运行结果。例如，当你部署一个合约后或执行 一个合约方法后，就会在终端看到它的执行信息：

点击信息行右侧的下拉图标，就可以查看该信息的详情；点击[debug]按钮，就会打开右侧面板中的 调试页对合约进行单步或断点调试。

Remix终端顶部的工具栏提供了切换终端显示状态、清理终端输出等功能，显示待定交易的量， 选择监听交易的范围，也可以搜索历史交易。

3、Solidity IDE Remix功能面板

Solidity IDE Remix的右侧为功能面板，以选项页的方式提供编译、运行、静态分析、测试、 调试、设置和技术支持功能。

3.1 编译选项页

在编译选项页，你可以点击下拉框切换当前要使用的Solidity编译器版本：

然后点击[开始编译]按钮，就会编译Remix文件编辑器中当前选中的代码文件，比如我们的 hello.sol文件。编译完成后，如果没有编译错误，就可以看到合约名字Hello出现在编译 选项页的合约下拉框中：

可以点击[swarm]按钮将编译好的合约上传到Swarm网络，或者点击[详情]按钮查看编译 结果详情，也可以点击[ABI]或[字节码]按钮，分别将合约的ABI与字节码拷贝到系统剪切板 以便在其他程序中使用。

3.2 运行

在**运行**选项页，可以部署编译好的合约，也可以执行已部署合约的方法：

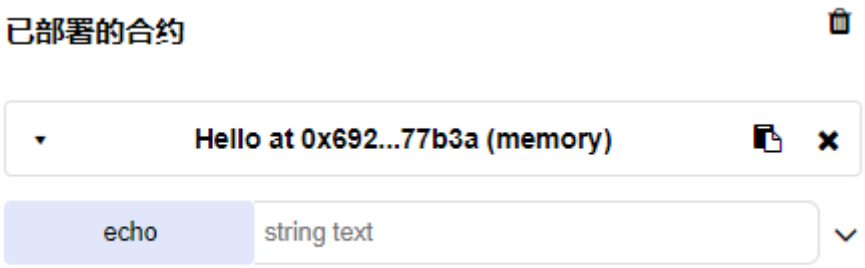
节点环境选项提供三种选择：JS虚拟机、注入Web3对象或使用web3提供器。

- JS虚拟机是一个JS版本的以太坊虚拟机实现，它运行在你的浏览器内，因此你不需要考虑 节点配置或者担心损失以太币，最适合学习和快速原型验证。
- 如果你的浏览器安装了Metamask插件，或者使用Mist之类的以太坊兼容浏览器，那么也 可以选择第二个环境：使用注入的Web3对象。
- 如果你有自己的节点，那么可以选择第三个选项*使用web3提供器*来让Remix连接 到你的节点上，不过如果要连接的节点是接入以太坊主网的，要注意每一次交易都是 有成本的！

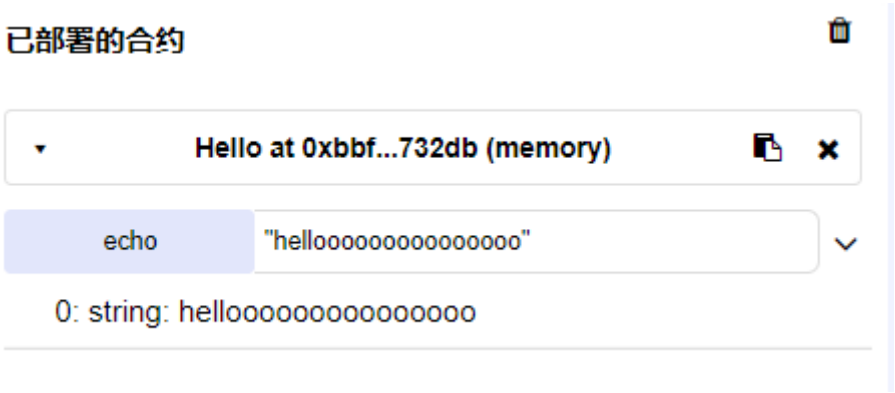
本实验将全程使用JS虚拟机，有兴趣的同学可以安装metamask插件自行研究

如果之前有编译好的合约，在运行选项页就可以看到这个合约的名字，例如我们的Hello。 点击[部署]按钮就可以将这个合约部署到我们选定的节点环境了：

现在可以看到，*已部署的合约*区域，已经出现我们的合约了。点击这个合约实例， 可以看到我们为Hello合约定义的echo方法自动显示出来了：



在方法名后面的输入框里输入方法参数，例如”hellooooooooooooo”，然后点击方法名， 就可以执行合约的方法了：



可以看到返回值的确和输入的参数确实是一样的，目标达成。

3.3 其他选项页

Solidity Remix集成开发环境还有很多功能，这里只对其他的选项页做简单介绍：

- **分析**选项页提供对Solidity合约代码的静态分析选项。
- **测试**选项页提供单元测试能力，你可以生成一个测试文件，或者执行一组测试。
- **调试器**选项页可以单步跟踪合约的执行、查看合约状态或局部变量等。
- **设置**选项提供Solidity Remix IDE本身的一些参数调整能力，例如设置编辑器文本自动折行、 启用插件、设置gist访问令牌，或者切换Remix IDE的皮肤主题 —— 目前只有三个：浅色、深色和净色。

4、Solidity基础语法规则

4.1 合约

Solidity 的代码都包裹在**合约**里面. 一份 `合约` 就是以太应币应用的基本模块， 所有的变量和函数都属于一份合约, 它是你所有应用的起点.

一份名为 `HelloWorld` 的空合约如下:

```
contract HelloWorld {
}
```

4.2 版本指令

所有的 Solidity 源码都必须冠以 "version pragma" — 标明 Solidity 编译器的版本. 以避免将来新的编译器可能破坏你的代码。

例如: `pragma solidity ^0.5.13;` (当前 Solidity 的最新稳定版本是 0.5.13).

当然我们也可以指定一个版本区间，比如 `pragma solidity >=0.4.12 <0.6.0;` 很好理解就不解释了。

综上所述， 下面就是一个最基本的合约 — 每次建立一个新的项目时的第一段代码:

```
pragma solidity ^0.5.13;

contract HelloWorld {
}
```

4.3 状态变量和整数

状态变量：被永久地保存在合约中。也就是说它们被写入以太币区块链中. 可以想象成写入一个数据库。

```
contract Example {
  // 这个无符号整数将会永久的被保存在区块链中
  uint myUnsignedInteger = 100;
}
```

在上面的例子中，定义 `myUnsignedInteger` 为 `uint` 类型，并赋值100。

无符号整数`uint`： `uint` 无符号数据类型， 指**其值不能是负数**，对于有符号的整数存在名为 `int` 的数据类型。

> 注: Solidity中， `uint` 实际上是 `uint256`代名词， 一个256位的无符号整数。你也可以定义位数少的uints — `uint8`， `uint16`， `uint32`， 等..... 但一般来讲更愿意使用简单的 `uint`， 除非在某些特殊情况下。

字符串`string`： 字符串用于保存任意长度的 UTF-8 编码数据。 如： `string greeting = "Hello world!"`。

4.4 数学运算

在 Solidity 中，数学运算很直观明了，与其它程序设计语言相同:

- 加法: `x + y`
- 减法: `x - y`,
- 乘法: `x * y`
- 除法: `x / y`
- 取模 / 求余: `x % y` *(例如, `13 % 5` 余 `3`)*

Solidity 还支持 **乘方操作** (如：x 的 y次方） // 例如： 5 ** 2 = 25

```
uint x = 5 ** 2; // equal to 5^2 = 25
```

4.5 结构体

有时你需要更复杂的数据类型，Solidity 提供了 **结构体**:

```
struct Person {
  uint age;
  string name;
}
```

结构体允许你生成一个更复杂的数据类型，它有多个属性。

4.6 数组

如果你想建立一个集合，可以用 ***数组***这样的数据类型. Solidity 支持两种数组: ***静态***数组和***动态***数组:

```
// 固定长度为2的静态数组:
uint[2] fixedArray;
// 固定长度为5的string类型的静态数组:
string[5] stringArray;
// 动态数组，长度不固定，可以动态添加元素:
uint[] dynamicArray;
```

你也可以建立一个 **结构体**类型的数组 例如，比如之前提到的 `Person`:

```
Person[] people; // 这是动态数组，我们可以不断添加元素
```

*记住：状态变量被永久保存在区块链中。所以在你的合约中创建**动态数组**来保存成结构的数据是非常有意义的。*

公共数组

你可以定义 `public` 数组，语法如下:

```
Person[] public people;
```

`public`条目意味着其它的合约可以从这个数组读取数据（但不能写入数据），所以这在合约中是一个有用的保存公共数据的模式。

数组中插入元素

可以定义一个新的 `Person` 结构，然后把它加入到名为 `people` 的数组中.

现在我们学习创建新的 `Person` 结构，然后把它加入到名为 `people` 的数组中.

```
Person satoshi = Person(172, "Satoshi");
people.push(satoshi);
```

你也可以两步并一步，用一行代码更简洁:

```
people.push(Person(16, "Vitalik"));
```

> 注：`array.push()` 在数组的 **尾部** 加入新元素，所以元素在数组中的顺序就是我们添加的顺序，如:

```
uint[] numbers;
numbers.push(5);
numbers.push(10);
numbers.push(15);
// numbers is now equal to [5, 10, 15]
```

> `array.push()` 在完成加入之后，同时会返回数组的长度，类型是`uint`

4.7 函数

在 Solidity 中函数定义的句法如下:

```
function eatHamburgers(string _name, uint _amount) public returns (string) {
}
```

这是一个名为 `eatHamburgers` 的函数，它接受两个参数：一个 `string` 类型的 和 一个 `uint` 类型的，返回一个 `string` 类型。

> 注：习惯上函数里的变量都是以(`_`)开头 (但不是硬性规定) 以区别全局变量。本实验会沿用这个习惯。

和其他语言一样，函数调用方式如下:

```
string result = eatHamburgers("vitalik", 100);
```

公开、私有函数

Solidity 定义的函数的属性默认为 `public`。 这就意味着任何一方 (或其它合约) 都可以调用你合约里的函数。

显然，不是什么时候都需要这样，而且这样的合约易于受到攻击。 所以将自己的函数定义为 `private` 是一个好的编程习惯，只有当你需要外部世界调用它时才将它设置为 `public`。

定义一个私有函数很简单，在其后添加 `private` 关键字即可。和函数的参数类似，私有函数的名字习惯用(`_`)起始。

```
uint[] numbers;

function _addToArray(uint _number) private {
  numbers.push(_number);
}
```

这意味着只有合约内部才能够调用这个函数，给 `numbers` 数组添加新成员。

Ps：当然显式指定函数为 `public` 也是可以的，通常为了可读性，总是会标记函数属性。

更多的函数修饰符

参考这样的代码：

```
string greeting = "What's up dog";

function sayHello() public returns (string) {
    return greeting;
}
```

上面的函数实际上没有改变 Solidity 里的状态，即，它没有改变任何值或者写任何东西。

这种情况下我们可以把函数定义为 **view**, 意味着它只能读取数据不能更改数据:

```
function sayHello() public view returns (string) {}
```

Solidity 还支持 **pure** 函数, 表明这个函数甚至都不访问应用里的数据，例如：

```
function _multiply(uint a, uint b) private pure returns (uint) {
    return a * b;
}
```

这个函数甚至都不读取应用里的状态——它的返回值完全取决于它的输入参数，在这种情况下我们把函数定义为 **pure**.