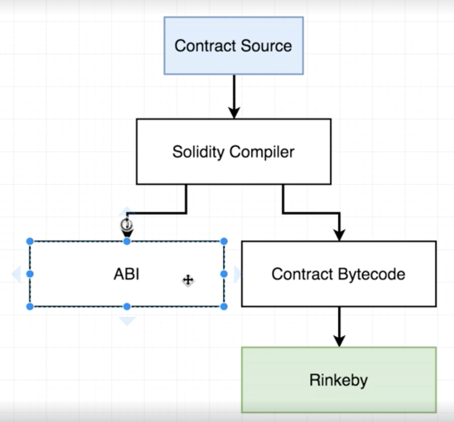
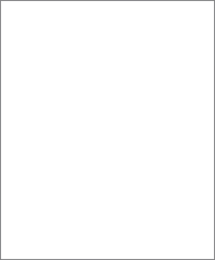
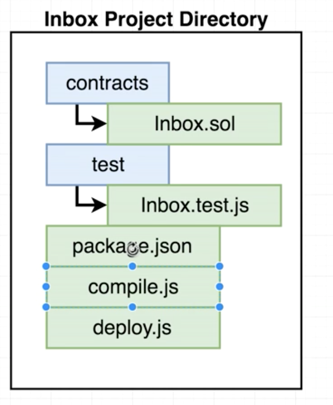
欢迎继续我们的学习。

在上一课的内容中，我们在Atom（或者你喜欢的编辑器）中创建了新的项目。

本课将继续学习如何编译使用solidity语言编写的智能合约。

首先回顾下项目的结构。



接下来要实现的就是其中的compile.js，因为无论是测试，还是最终的部署，都需要基于这个js文件。

那么这个compile.js应该如何实现其功能呢？还是回顾下：

为了实现这个目标，首先需要安装一个node模组。

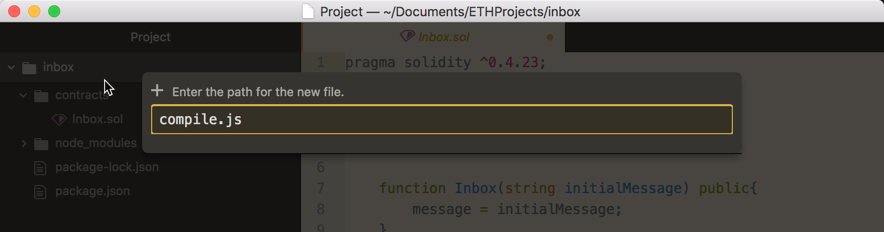
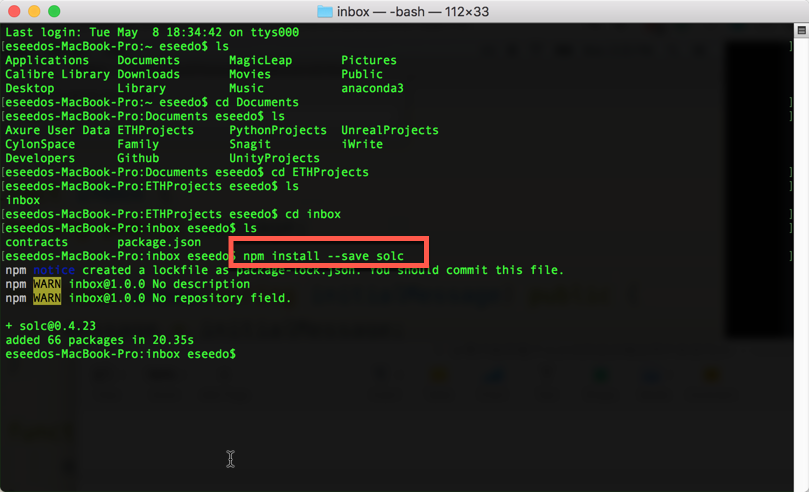
回到Terminal(windows下的cmd）终端，进入项目文件夹所在的路径。

然后输入以下命令：

npm install —save solc

接下来回到Atom，在Inbox目录下右键单击，选择new file，并输入文件名compile.js

要注意的是，在compile.js中，不要直接使用require语句引用sol文件，否则会报错。



require('./contracts/Inbox.sol'); //wrong

这是因为使用require语句，node会将sol当作js文件来直接处理，而显然这样是会出问题的。

这里我们将采用使用文件路径的方式来获取sol文件，因此将用到path 这个package。使用path包的好处是无论在mac系统还是windows系统，都会得到正确的文件路径。

接下来看看具体的代码：

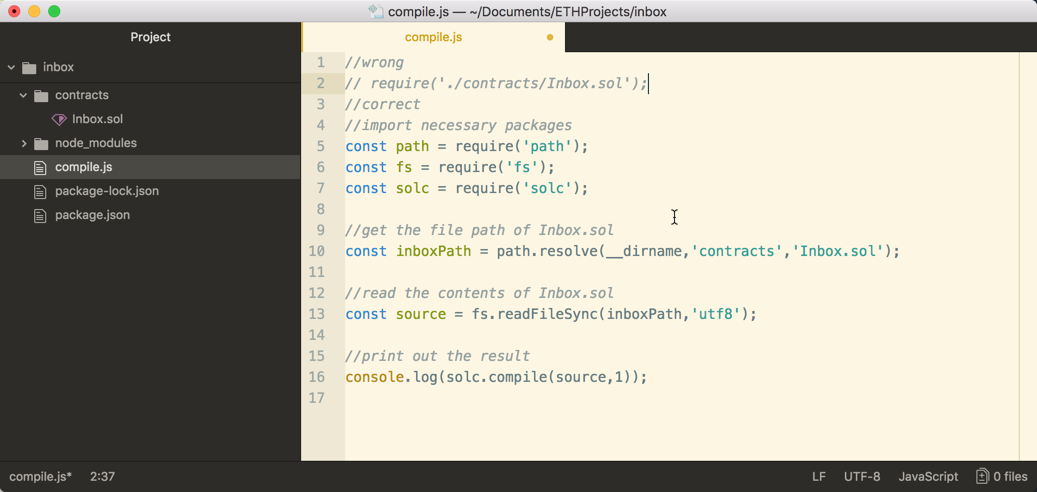
下面按照代码行的数字编号来解释一下具体的含义：

1.第2行代码是错误的导入Inbox.sol的方式，原因刚才已经说明了。

2.第5-7行代码中导入了所需的几个package,分别是用于获取文件路径的path,用于读取文件内容的fs，和用于编译solidity的solc

3.第10行代码的作用是获取Inbox.sol的文件路径

4.第13行代码的作用是读取Inbox.sol的内容，采用utf8编码



5.第16行代码的作用是使用solc编译Inbox.sol的内容，并打印出来。

接下来回到Terminal终端（windows下的cmd)，并尝试运行。

进入项目所在文件夹，并输入以下命令：

node compile.js

等待几秒后，就会看到结果。



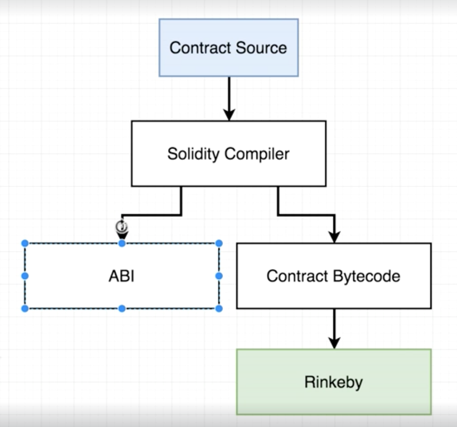
看到这个画面，是不是有点黑客帝国的感觉~

不过虽然里面的内容有点吓人，但是结构还是很清晰的：

1.最外层是contracts

2.然后是Inbox

3.接着是assembly,  
bytecode,  
functionHashes,  
gasEstimates,  
interface，  
metadata,  
opcodes,  
runtimeBytecode,  
srcmap,  
srcmapRuntime,  
errors,  
sourceList,  
sources



其中我们最关心的是byteCode和interface。在之前我们曾经描述过智能合约的编译过程：

其中byteCode就是Contract Bytecode,而interface就是图中的ABI。

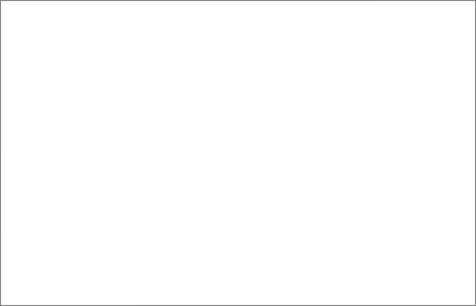
ABI其实就是列出了合约中所有可以被调用的函数，参数等等信息。

好了，现在我们通过console.log已经了解了使用solc编译的结果，接下来将对compile.js进一步的完善。

修改代码如下：

红框部分就是修改的代码，也就是第16行。

跟之前不同的是，我们进一步获取了Inbox这个特定的智能合约的信息，并使用module.exports导出。



好了，本课的内容就到这里了，我们下一课再见~

讨论群-笨猫学编程QQ群：

375143733

答疑论坛：

<http://www.vr910.com/forum.php?mod=forumdisplay&fid=157>

知乎专栏：

<https://zhuanlan.zhihu.com/kidscoding>

新浪博客:

<http://blog.sina.com.cn/eseedo>

Github:

<https://github.com/eseedo>

个人网站：

<http://icode.ai/>