1. 发布代币合约

1.1

[这里]

(<https://github.com/ONT-Avocados/python-template/blob/master/OEP4Sample/OEP4Sample_compiler2.0.py>)

是OEP4的合约源码，也就是ERC20也就是同质化资产代币。

1.2

注意

[这里]

(<https://github.com/ONT-Avocados/python-template/blob/master/OEP4Sample/OEP4Sample_compiler2.0.py#L20>)

把你自己的地址填上去，目的是把所有的初始代币都转到你填写的地址中。

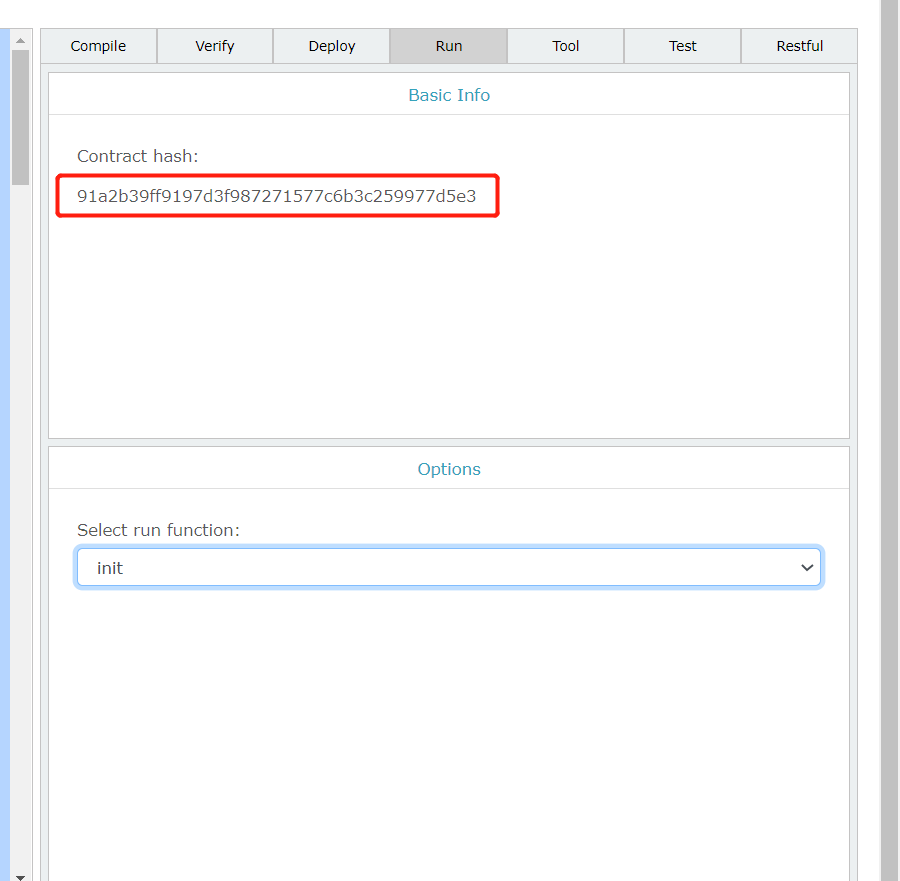
1.3

关于如何使用smartx可以参考

[这里](https://github.com/ontio/ontology-smartcontract/blob/master/README\_cn.md)

1.4 compile, deploy之后，调用`init`方法，不需要传参数，所有代币都会转入你1.2步的地址中。

1.5 注意这里的合约hash：91a2b39ff9197d3f987271577c6b3c259977d5e3

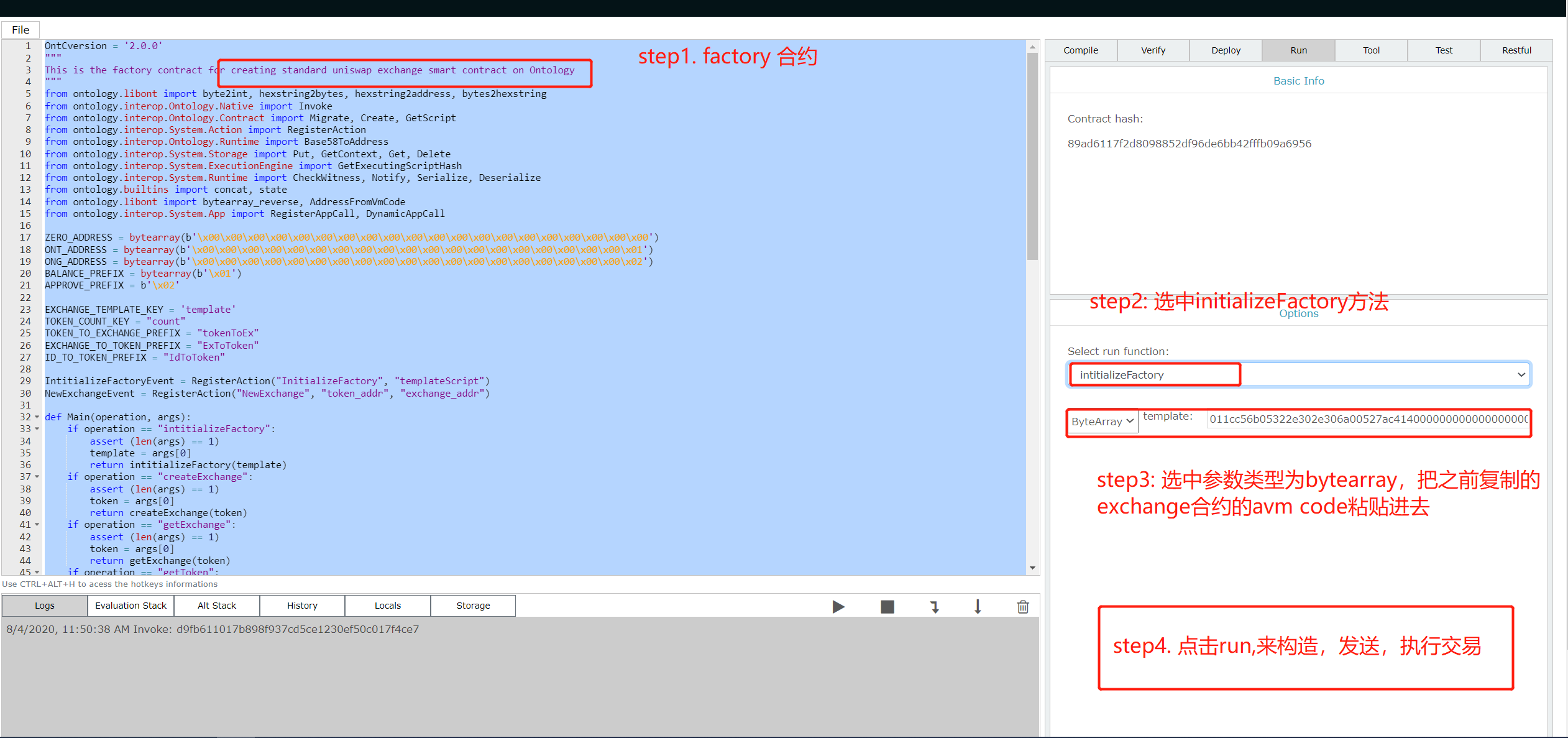


2. 发布exchange合约，并完成与factory合约的绑定

2.1 首先需要参考1来编译、布署factory合约，然后编译exchange合约，注意exchange合约不需要布署，编译exchange合约之后可以看到这个界面

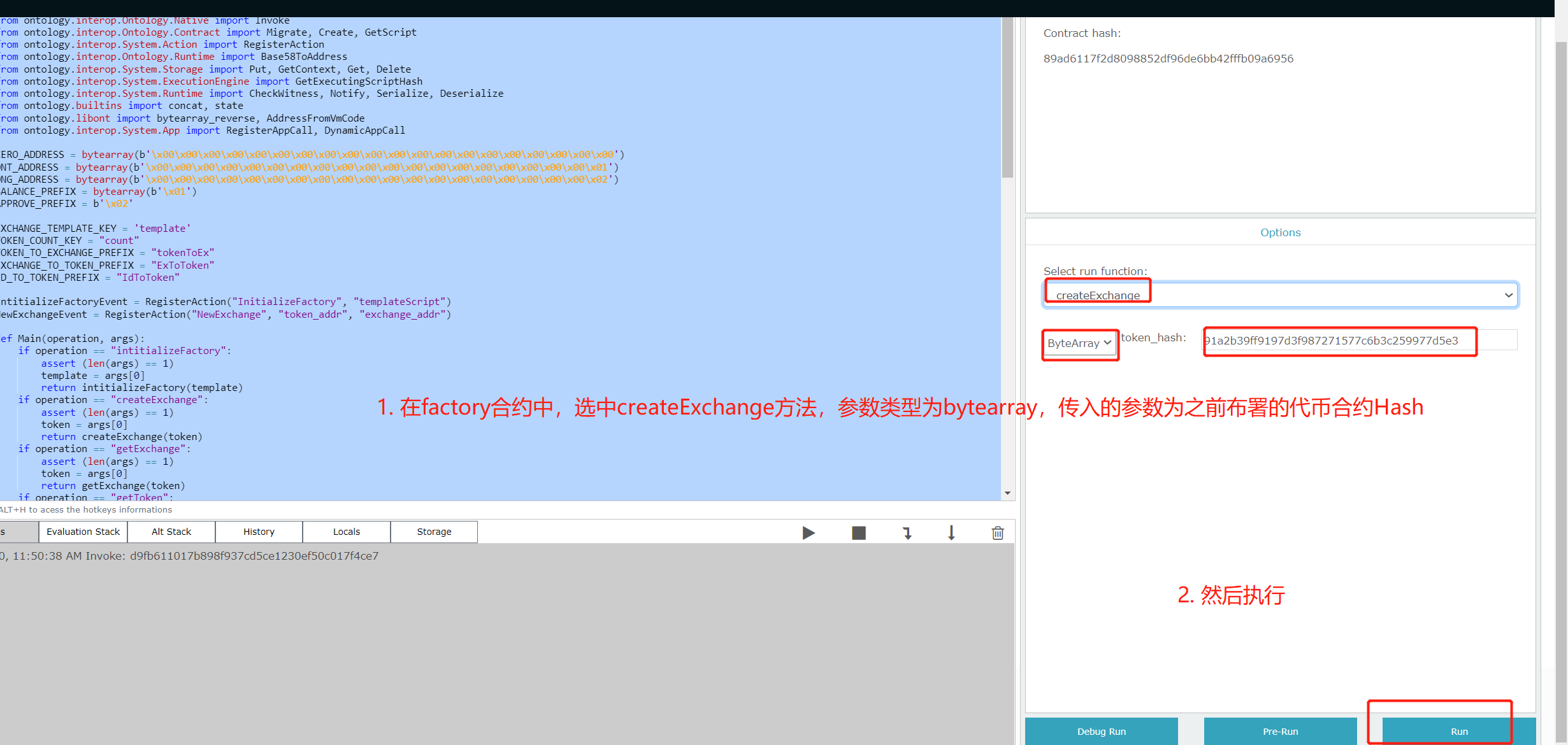


2.2 复制avm code，打开factory合约布署后的界面，

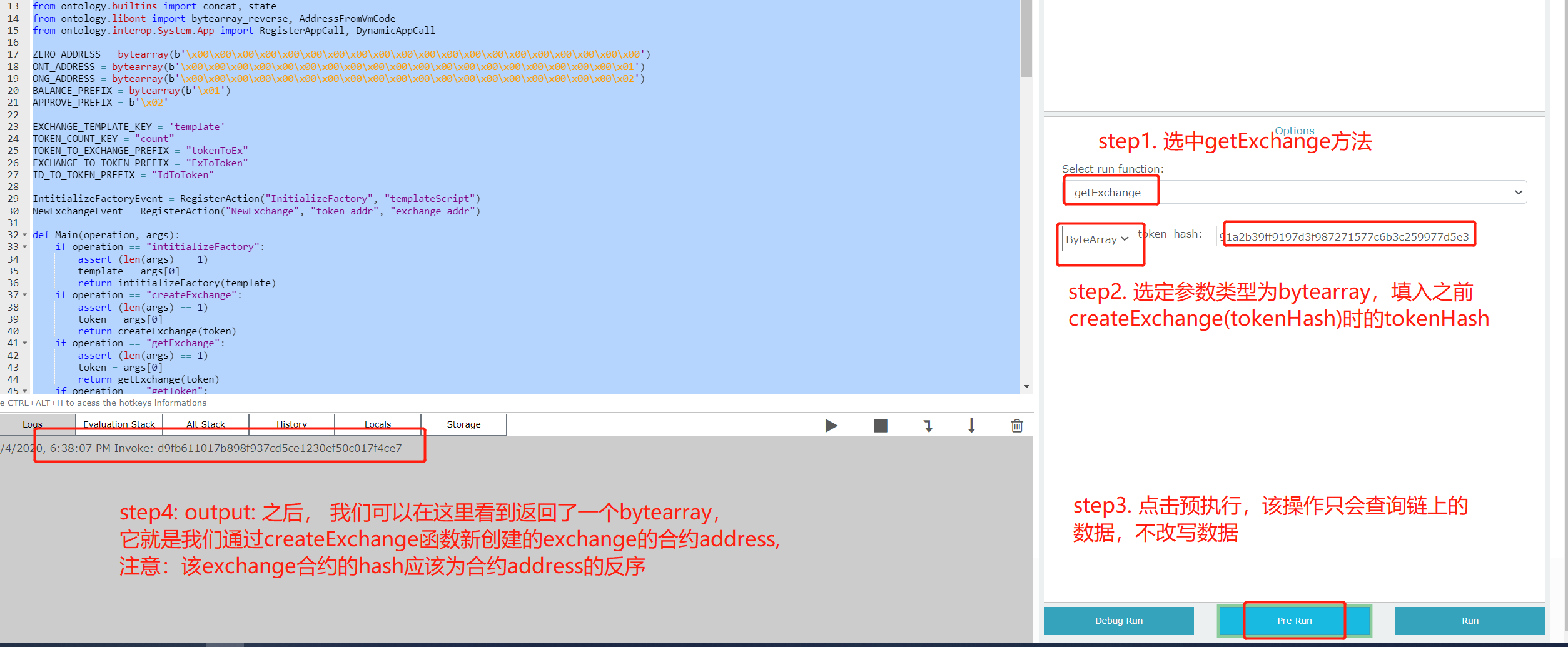


2.3 上一步调用成功后，exchange合约的byte code就已经存储在factory合约里面了，任何人可以通过调用 createExchange方法来创建一个新的exchange合约，

createExchange(91a2b39ff9197d3f987271577c6b3c259977d5e3)



2.4 可以通过getExchange()接口进行预执行，得到exchange address，注意exchange address为exchange hash的反序(以字节为单位)，



解释：

在这一步中，我们调用createExchange(tokenHash1)时，该函数内部逻辑为  
L1: Factory合约取出我们之前initializeFactory时传入的byte code（它是通过编译exchange合约得到的），然后布署一个新的exchange,该合约exchangeHash1

L2: Factory合约调用新布署的exchangeHash1的setup()方法,来让新布署的exchangeHash1合约识别并记录该exchange的Factory是哪个。

L3: Factory合约接着把映射写入存储

tokenHash1 -> exchangeAddress1(即reversed exchangeHash1)

exchangeHash1-> tokenAddress1(即reversed tokenHash1)

L4：setup函数的参数有两个：tokenAddress1, factoryAddress，

Setup内部逻辑为：

确保factoryAddress是一本合约而不是一个普通帐户

将tokenAddress1和factoryAddress保存下来

至此，我们完成了通过factory 对exchange的创建工作，且两本合约都记录着对方。

关于ont与ontd的转换

测试网 ONT-Decimal 合约hash: 2e0de81023ea6d32460244f29c57c84ce569e7b7

Ont->ontD接口：

函数名ont2ontd

参数fromAcct: 类型为address,含义为谁想把他ont转换成为ontd

参数ontAmount: 类型为interger,含义为谁把他多少个ont转换为ontd

Ontd->ont接口：

函数名：ontd2ont

参数fromAcct: 类型为address,含义为谁想把他ont转换成为ontd

参数ontdAmount: 类型为interger,含义为谁把他多少个ontd转换为ont

其他所有的接口服从该文档，<https://github.com/ontio/OEPs/blob/master/OEPS/OEP-4.mediawiki>