本课程文字讲义中涉及到知识点和概念的使用参见视频教程，您可通过以下方式联系并加入开发者引擎社区获取：

第四课 本地部署与标准讲解

本节课分为两个部分：

1、Flow 合约本地模拟部署

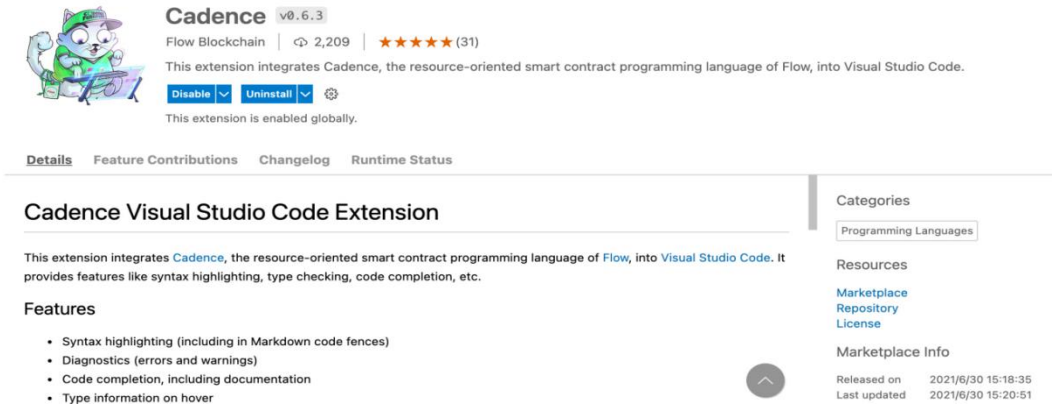
2、FT，NFT，NFT-Metadata 标准解析

（本节课大部分内容需结合视频教程）

一、Flow 合约本地模拟部署

1.1、环境准备

1、推荐 visual studio 编辑器，安装 Cadence 插件。



2、Flow-Cli 命令行安装

不同的系统有不同的安装方式，参见官方文档 Flow-Cli 安装

1.2、项目配置初始化

使用命令 flow init [--global]，其中 global 表示设置为全局配置。通常情况，一个项目一个配置，互不干扰是最优的处理方式。初始化完成之后，生成名为 flow.json 的配置文件，每个配置项的含义与功能如下：

```
{
    "emulators": {
        "default": {
            "port": 3569,
            "serviceAccount": "emulator-account"
        }
    },
    "contracts": {},
    "networks": {
        "emulator": "127.0.0.1:3569",
        "mainnet": "access.mainnet.nodes.onflow.org:9000",
        "testnet": "access.devnet.nodes.onflow.org:9000"
    },
    "accounts": {
        "emulator-account": {
            "address": "f8d6e0586b0a20c7",
            "key": "ea41c9ff...xxxxxxxxxxxxxxxx"
        }
    },
    "deployments": {}
}
```

1、emulators: 模拟器列表，保持默认不动即可。

2、networks: 区块链网络列表，在部署的时候可以指定使用那个网络。Flow 自动生成了默认的节点配置，如需用到其他的节点，后续添加即可。对于本节课的演示，保持不变即可。emulator 代表本地模拟器，mainnet 代表主网，而 testnet 代表测试网。

3、accounts: 所有的账户列表。这一部分需要指出两点：1）这里列出的账户不仅仅是某一个网的用户，而是可能用到的所有网的账户，都在这一项下面一字码排开，不需要加任何的层级来区分我这一些账户是属于模拟器的，然后这些账户是属于主网的，这些账户是属于测试网的。2）配置初始化完成之后，在这里生成了一个默认账户，emulator-account。那么问题就是 flow 为什么要生成一个默认账户呢？这个问题我们稍后再说。

4、contracts: 顾名思义，这一项就是声明所有要部署的合约。

5、deployments: 这一项配置是对前面三者的一个粘合，就是他告诉 Flow，要把哪些合约部署到哪个网的哪个账户下。

1.3、项目配置操作

具体的，初始化完成的一个项目该如何配置呢？emulator 和 networks 配置就保持不变，我主要看其余三个配置的使用。

1、accounts 配置有三种形式：

1）简单形式，如上图 emulator-account 的配置形式

```
"network-acctname" {
        "address" : "1234567890",
        "key" : "12345...67890"
}
```

其中，network-acctname 为用户自定义账户名称，建议带上该账户所属的网络名称前缀，以下相同。address 即为账户地址，key 为账户的私钥。

2）高级形式

```
"network-acctname" {
    "address" : "1234567890",
    "key" : {
        "type" : "hex",
        "index" : 0,
        "signatureAlgorithm" : "ECDSA_P256",
        "hashAlgorithm" : "SHA3_256",
        "privateKey" : "12345...67890"
    }
```

其中，账户类型 type 为 hex 保持不变即可。index 表示该账户下公钥的索引。signatureAlgorithm 和 hashAlgorithm 代表账户使用的签名和哈希算法。privateKey 即对应 index 指定公钥的私钥

3）google KMS 形式，参见官方文档。

2、contracts 配置有两种形式

1）简单形式，合约名称以及项目中的文件位置

```
"contractname" : "./contractsDir/contract1File.cdc"
```

2）高级形式，用于引入已部署存在的合约，alias 的合约不写入 deployments

```
"contractname" : {
        "source" : "./contractsDir/contract1File.cdc",
        "aliases" : {
                "network1name" : "deployed contract address1",
                "network2name" : "deployed contract address2"
        }
}
```

aliases 配置就告诉 Flow 从那个个地址获取项目中依赖的链上已经存在的合约。source 为该依赖合约的源码位置，因为如果要在本地模拟器部署，而本地模拟器上是没有部署该合约的。

3、deployments 配置

此配置形式的含义为：告诉 Flow 在什么网络的哪个个账户下部署哪些合约。

```
"network1name" : {
        "network-acctname" : [
                "contract1name",
                "contract2name"
        ]
}
```

1.4、模拟器启动及账户创建

在 Flow 上，创建一个账户有两个前提：第一个在 flow 上创建账户与以太坊上是不同的，它必须要由一个现有的账户来发起一笔交易，来创建一个账户。所以这实际上也解释了我们刚才在讲配置的时候来提出的问题，为什么我们初始化完了之后，他必须要为我们默认生成一个账户，如果他不生成账户，我们也没法去创建其他的账户了。第二个是在创建账户之前，要先创建一个密钥对，在用此生成一个账户。

接下演示账户创建，参见视频教程演示。

1、密钥对生成命令：> flow keys generate [--sig-algo = ECDSA_P256/ECDSA_secp256k1]

--sig-algo 指定该账户生成使用的加密算法，如果不指定，默认为 ECDSA_P256。

2、开启模拟器命令：> flow emulator start [--persist]

--persist 代表将模拟器上状态的变更持久化，重新启动后还保持上次关闭时的状态。如果不指定此选项，则每次启动模拟器都是全新的状态

3、创建一包含单公钥的账户：

```
chentenglong@localhost hello % flow accounts create \
--key 818fdf40e85a55d8f04b674d21560ad646c111066ee35b0c3b74ad5731c7f0a07a5465f70634d4012b656dddfad10703
\
--sig-algo ECDSA_secp256k1 --signer emulator-account
```

其中，--key 即为生成密钥对的公钥；--sig-algo 为密钥对生成时使用的加密算法，当使用默认的 P256 时，可省略；--signer 指定为创建账户这笔交易签名的账户

4、创建一个包含多公钥的账户：

```
chentenglong@localhost hello % flow accounts create \
> --key 9b0b26541f0fc54b9c9720622bca696246079b8b6a65c962d779622db770427db6f4d8c1c2b08d760262567fca6c1264a9b923a86703
e --sig-algo ECDSA_P256 --hash-algo SHA3_256 \
--key 818fdf40e85a55d8f04b674d21560ad646c111066ee35b0c3b74ad5731c7f0a07a5465f70634d4012b656dddfad10703c031fe13ac5737
--sig-algo ECDSA_secp256k1 --hash-algo SHA3_256 \
 --signer emulator-account
```

以上之灵创建了一个包含两个公钥的账户。当指定多公钥时，要分别指定使用的加密算法和哈希算法。创建成功之后，输出如下信息：

```
Transaction ID: 5d07230ae461d170c55b368b88709625e39d184888dee9b72ccdf5143d381936

Address  0x179b6b1cb6755e31
Balance  0.00100000
Keys     2

Key 0   Public Key              9b0b26541f0fc54b9c9720622bca696246079b8b6a65c962d779622db7704
a867037965b913f5a4c49e25ee
        Weight                  1000
        Signature Algorithm     ECDSA_P256
        Hash Algorithm          SHA3_256
        Revoked                 false
        Sequence Number         0
        Index                   0


Key 1   Public Key              818fdf40e85a55d8f04b674d21560ad646c111066ee35b0c3b74ad5731c7f
```

其中，Address 即为新生成的账户地址。Balance 为账户余额。Keys 只是此账户下包含多少个公钥，随后紧跟公钥的详细信息。公钥信息中有个 Weight 量表示公钥的权重值，默认 1000。

1.5、账户与公钥的关系

在初步认识了账户与公钥的关系后，有以下几点需注意和了解：
1、账户地址的生成不是由公钥经过某种算法生成的，是由链上的一个函数确定性的分配。
2、创建账户时可以不指定公钥；使用同一个公钥生成的两个账户地址不同。
3、由上，账户与公钥是多对多关系。
4、目前的 Flow-Cli 中命令的实现不支持后期给账户添加公钥，需要使用 GoSDK。

1.6、模拟器合约部署与交互

视频教程演示了完整的从空文件夹直到部署合约并与之交互的完整过程，使用的示例代码可以添加开发者引擎社区获取。合约及其 flow.json 配置文件示例如下：


```
"contracts": {
    "HelloWorld": "./contracts/HelloWorld.cdc"
},
"accounts": {
    "emulator-account": {
        "address": "f8d6e0586b0a20c7",
        "key": "bf4fc31ede0xxxxxxxxxx5b72f39d7b6974d627b"
    },
    "emulator-creator": {
        "address": "01cf0e2f2f715450",
        "key": {
            "type": "hex",
            "index": 0,
            "signatureAlgorithm": "ECDSA_secp256k1",
            "hashAlgorithm": "SHA3_256",
            "privateKey": "f3873884e24exxxxxxxxabfcb6f2f748"
        }
    }
},
"deployments": {
    "emulator": {
        "emulator-creator": [
            "HelloWorld"
        ]
    }
}
```

关于 accounts 配置下 emulator-creator，我使用了高级形式的原因为：创建该账户使用的密钥对不是通过默认的算法生成的。如果使用默认算法的密钥对则可以使用简单形式。

1、合约的部署与更新

> flow project deploy [--network=emulator]

其中，--network 指定部署那个网络，默认本地模拟器，测试网为 testnet，主网 mainnet

> flow accounts update-contract　Hello ./hello/contract.cdc --signer=the-creator

2、执行脚本 Script

> flow scripts execute hello/sayHi.script.cdc "Hello" 123

> flow scripts execute hello/sayHi.script.cdc \

--args-json='[{"type": "String", "value": "Cadence"},{"type": "Int", "value": "123"}]'

以上命令展示了如何给脚本传参的两种方式。

3、执行交易 transaction

Flow 中交易的执行需要涉及到如下三种角色：

1）发起人(proposer)：更新交易记录序列号；

2）付费者(payer)：支付交易费者；

3）授权者(authorizers): prepare 阶段的参数。

Flow 要求签名的公钥 key 的权重值必须大于等于 1000，可以据此实现多公钥签名，如设置两个 key 的权重 500。

发送交易有两种方式，一种是分步进行，这种需要详细指定三种角色都是哪个账户，虽复杂一些但更灵活；另一种适用于当三种角色是同一个账户时则三合一完成，如下所示：

1、交易签名与发送：

构建交易：

```
flow transactions build ./hello/sayHi.transaction.cdc \
--authorizer the-creator \
--proposer the-creator \
--payer the-creator \
--filter payload \
 --save transaction.build.rlp
```

签名交易：

```
flow transactions sign ./transaction.build.rlp \
--signer the-creator \
 --filter payload \
--save transaction.signed.rlp
```

发送交易：

```
flow transactions send-signed ./transaction.signed.rlp
```

2、三步合一：

```
flow transactions send ./hello/sayHi.transaction.cdc  --signer the-creator
```

详细演示参见视频教程。

二、FT，NFT，NFT-Metadata 标准解析

Flow 依赖一组核心的合约以保证正常有序运行。本节课主要简单分析一下 Fungible Token(对标以太坊的 ERC20)，Non-Fungible Token(对标以太坊的 ERC721)，以及 NFT-MetaData 补充协议。这三个协议的官网介绍以及源码码参见：

FT：https://docs.onflow.org/core-contracts/flow-token/

NFT：https://docs.onflow.org/core-contracts/non-fungible-token/

NFT-MetaData: https://docs.onflow.org/core-contracts/nft-metadata/

三个合约在 Flow 各个网部署的公开账户地址如下，若我们在项目中用到时，可以采用合约配置的高级形式直接引用。

Source: FlowToken.cdc

NETWORK	CONTRACT ADDRESS
Emulator	0x0ae53cb6e3f42a79
Testnet	0x7e60df042a9c0868
Mainnet	0x1654653399040a61

Source: NonFungibleToken.cdc

NETWORK	CONTRACT ADDRESS
Testnet	0x631e88ae7f1d7c20
Mainnet	0x1d7e57aa55817448

Source: MetadataViews.cdc

NETWORK	CONTRACT ADDRESS
Testnet	0x631e88ae7f1d7c20
Mainnet	0x1d7e57aa55817448

各合约结合源码的详细讲解参见视频教程

2.1、FT 标准

这是一个合约接口。核心是一个 Vault 的资源，带有一个 balance 属性，账户之间发生转账时，转入转出的均为 Vault 资源，而不简简单单数字上的扣减和增加。Vault 资源实现了三个分权的接口：Provider 转出具有一定数额的 Vault，Reciver 接收一定数额的 Vault, Balance 获得余额。

此外在实现示例上定义了 administrator 可以创建 minter 和 burner 对 FT 进行管理。

2.2、NFT 标准

这是一个合约接口。除了 NFT 及与之相关的操作外，核心为一个 Collection 的资源，相当于存储 NFT 的仓库，实现了三个分权的接口：Provider 提供转出 NFT 操作，Receiver 接收一个 NFT，CollectionPublic 定义了其他一些获取 NFT 相关的接口。特别地，borrowNFT()的定义返回一个对 NFT 的引用，而示例实现 exampleNFT 中，除实现了这个方法，还额外定义了一个 borrowExampleNFT()，也返回了个引用。两者的返回值略有不同，前者是标准中 NFT 的引用，后者是 exampleNFT 中实现的引用。

2.3、NFT-Metadata 补充标准

与以上不同，这是一个合约，虽是合约，但内也只是定义了一些需要 NFT 去实现或展示使用的接口或类型等相关信息。这个补充标准的主要用途在于统一 NFT 的展示等类似场景。现在的 NFT-Metadata 标准包含两大部分功能解析 NFT 资源的信息和 NFT 版税标准的确立。NFT 资源本身的信息核心由 Resolver 和 ResolveCollection 两个接口定义，由具体的 NFT 资源实现。而版税信息由 Royalty 结构体类型描述，主要字段为某账户下具有 FungibleToken.Receiver 权限的 Capability。此账户一般为 minter 的账户，当 NFT 每次被交易，都会有一定的版税支付，目的是激励。同时 NFT-Metaview 支持多人受益，需要在 mint NFT 时指定，可以参数的形式传入该交易。

本节课结束~

至此第一部分，围绕 Cadence ***语言的学习结束了，从下节课进入第二部分，分布式应用*** DApp ***开发实战.***