服务发现命令行界面

Service Discovery CLI

发现服务有自己的命令行界面（CLI），它用YAML配置文件来对包括证书和私钥路径以及成员服务提供者身份证（MSP ID）在内的属性进行维持。

The discovery service has its own Command Line Interface (CLI) which uses a YAML configuration file to persist properties such as certificate and private key paths, as well as MSP ID.

`discover` 命令有以下子命令：

The `discover` command has the following subcommands:

- saveConfig
- peers
- config
- endorsers
- saveConfig
- peers
- config
- endorsers

And the usage of the command is shown below:

下面展示的是该命令的用法：

```
usage: discover [<flags>] <command> [<args> ...]

~~~~ {.sourceCode .shell}
usage: discover [<flags>] <command> [<args> ...]

Command line client for fabric discovery service

Command line client for fabric discovery service

Flags:
  --help                     Show context-sensitive help (also try --help-long and --help-man).
  --configFile=CONFIGFILE    Specifies the config file to load the configuration from
  --peerTLSCA=PEERTLSCA      Sets the TLS CA certificate file path that verifies the TLS peer's cer
  --tlsCert=TLSCERT          (Optional) Sets the client TLS certificate file path that is used when
  --tlsKey=TLSKEY            (Optional) Sets the client TLS key file path that is used when the pee
  --userKey=USERKEY          Sets the user's key file path that is used to sign messages sent to th
  --userCert=USERCERT        Sets the user's certificate file path that is used to authenticate the
  --MSP=MSP                  Sets the MSP ID of the user, which represents the CA(s) that issued it

Flags:
  --help                     Show context-sensitive help (also try --help-long and --help-man).
  --configFile=CONFIGFILE    Specifies the config file to load the configuration from
  --peerTLSCA=PEERTLSCA      Sets the TLS CA certificate file path that verifies the TLS peer's cer
  --tlsCert=TLSCERT          (Optional) Sets the client TLS certificate file path that is used when
  --tlsKey=TLSKEY            (Optional) Sets the client TLS key file path that is used when the pee
  --userKey=USERKEY          Sets the user's key file path that is used to sign messages sent to th
  --userCert=USERCERT        Sets the user's certificate file path that is used to authenticate the
  --MSP=MSP                  Sets the MSP ID of the user, which represents the CA(s) that issued it
```

```
Commands:
  help [<command>...]
    Show help.

Commands:
  help [<command>...]
    Show help.

  peers [<flags>]
    Discover peers

  peers [<flags>]
    Discover peers

  config [<flags>]
    Discover channel config

  config [<flags>]
    Discover channel config

  endorsers [<flags>]
    Discover chaincode endorsers

  endorsers [<flags>]
    Discover chaincode endorsers

  saveConfig
    Save the config passed by flags into the file specified by --configFile
```

saveConfig Save the config passed by flags into the file specified by –configFile

```


Configuring external endpoints
------------------------------

配置外部端点
------------------------------

Currently, to see peers in service discovery they need to have `EXTERNAL_ENDPOINT`
to be configured for them. Otherwise, Fabric assumes the peer should not be
disclosed.

当前，若想在服务发现中看到节点，需要在其上配置 `EXTERNAL_ENDPOINT`。否则，Fabric假定该节点不应被揭露。

To define these endpoints, you need to specify them in the `core.yaml` of the
peer, replacing the sample endpoint below with the ones of your peer.

要定义这些端点，就得在peer的 `core.yaml `字段指明端点，把下面的样本端点换成你peer上的端点。

```
CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org1.example.com:8051
```

```
CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org1.example.com:8051
```

Persisting configuration

维持配置

To persist the configuration, a config file name should be supplied via
the flag `--configFile`, along with the command `saveConfig`:

要想维持配置，需要通过`—configFile` flag和 `saveConfig`命令来提供一个配置文件名:

```
discover --configFile conf.yaml --peerTLSCA tls/ca.crt --userKey msp/keystore/ea4f6a38ac7057b6fa9
```

~~~~ {.sourceCode .shell}
discover --configFile conf.yaml --peerTLSCA tls/ca.crt --userKey msp/keystore/ea4f6a38ac7057b6fa9
```

By executing the above command, configuration file would be created:

通过执行以上命令可创建配置文件：

```
$ cat conf.yaml
version: 0
tlsconfig:
  certpath: ""
  keypath: ""
  peercacertpath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org
  timeout: 0s
signerconfig:
  mspid: Org1MSP
  identitypath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
  keypath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.examp
```

```
$ cat conf.yaml
version: 0
tlsconfig:
  certpath: ""
  keypath: ""
  peercacertpath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org
  timeout: 0s
signerconfig:
  mspid: Org1MSP
  identitypath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
  keypath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.examp
```

When the peer runs with TLS enabled, the discovery service on the peer requires the client to connect to it with mutual TLS, which means it needs to supply a TLS certificate. The peer is configured by default to request (but not to verify) client TLS certificates, so supplying a TLS certificate isn't needed (unless the peer's `tls.clientAuthRequired` is set to `true`).

当TLS（安全传输层协议）启动时运行peer，peer上的发现服务需要客户端凭借相互的TLS证书来与之相连，这就意味着，发现服务需要提供一个TLS证书。peer的默认配置为请求（但不验证）客户端的TLS证书，因此并不需要提供TLS证书（除非peer的`tls.clientAuthRequired`被设置为`true`）。

When the discovery CLI's config file has a certificate path for `peercacertpath`, but the `certpath` and `keypath` aren't configured as in the above - the discovery CLI generates a self-signed TLS certificate and uses this to connect to the peer.

当发现CLI的配置文件有`peercacertpath`的证书路径，但是`certpath`和`keypath`没有按以上方式进行配置——发现CLI生成一个自签名的TLS证书并用该证书与节点连接。

When the `peercacertpath` isn't configured, the discovery CLI connects without TLS , and this is highly not recommended, as the information is sent over plaintext, un-encrypted.

当未配置`peercacertpath`，发现CLI与节点连接没有用到TLS证书。但由于信息是以纯文本形式进行传送，未经加密，因此极不推荐这种操作。

Querying the discovery service

查询发现服务

The discoveryCLI acts as a discovery client, and it needs to be executed against a peer. This is done via specifying the `--server` flag. In addition, the queries are channel-scoped, so the `--channel` flag must be used.

发现CLI作为一个发现客户端，需要在peer上执行。此过程通过指明 `--server flag`来实现。除此之外，查询是在通道范围内进行的，所以必须使用 `--channel` flag。

The only query that doesn’t require a channel is the local membership peer query, which by default can only be used by administrators of the peer being queried.

唯一不需要通道的的查询是本地成员节点查询，默认情况下，本地成员节点查询只能由被查询节点的管理员使用。

The discover CLI supports all server-side queries:

发现CLI支持所有服务器端的查询:

- Peer membership query
- Configuration query
- Endorsers query

- 节点成员查询

Let’s go over them and see how they should be invoked and parsed:

- 配置查询

Peer membership query:

- 背书者查询

```
$ discover --configFile conf.yaml peers --channel mychannel  --server peer0.org1.example.com:7051
[
    {
        "MSPID": "Org2MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer0.org2.example.com:9051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICKTCCAc+gAwIBAgIRANK4WBck5gKuzTxVQIwhYMUwCgY
        "Chaincodes": [
            "mycc"
        ]
    },
    {
        "MSPID": "Org2MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer1.org2.example.com:10051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRALnNJzplCrYy4Y8CjZtqL7AwCgY
        "Chaincodes": [
            "mycc"
        ]
    },
    {
        "MSPID": "Org1MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer0.org1.example.com:7051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc6gAwIBAgIQP18LeXtEXGoN8pTqzXTHZTAKBgg
        "Chaincodes": [
            "mycc"
        ]
    },
    {
        "MSPID": "Org1MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer1.org1.example.com:8051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQO7zMEHlMfRhnP6Xt65jwtDAKBgg
        "Chaincodes": null
    }
]
```

我们一起来看看这些查询，了解一下它们是如何被调用和语法分析的：

As seen, this command outputs a JSON containing membership information about all the peers in the channel that the peer queried possesses.

节点成员查询：

The `Identity` that is returned is the enrollment certificate of the peer, and it can be parsed with a combination of `jq` and `openssl`:

```
$ discover --configFile conf.yaml peers --channel mychannel  --server peer0.org1.example.com:7051
[
    {
        "MSPID": "Org2MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer0.org2.example.com:9051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICKTCCAc+gAwIBAgIRANK4WBck5gKuzTxVQIwhYMUwCgY
        "Chaincodes": [
            "mycc"
        ]
    },
    {
        "MSPID": "Org2MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer1.org2.example.com:10051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRALnNJzplCrYy4Y8CjZtqL7AwCgY
        "Chaincodes": [
            "mycc"
        ]
    },
    {
        "MSPID": "Org1MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer0.org1.example.com:7051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc6gAwIBAgIQP18LeXtEXGoN8pTqzXTHZTAKBgg
        "Chaincodes": [
            "mycc"
        ]
    },
    {
        "MSPID": "Org1MSP",
        "LedgerHeight": 5,
        "Endpoint": "peer1.org1.example.com:8051",
        "Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQO7zMEHlMfRhnP6Xt65jwtDAKBgg
        "Chaincodes": null
    }
]
```

```
$ discover --configFile conf.yaml peers --channel mychannel  --server peer0.org1.example.com:7051
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            55:e9:3f:97:94:d5:74:db:e2:d6:99:3c:01:24:be:bf
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=California, L=San Francisco, O=org2.example.com, CN=ca.org2.example.com
        Validity
            Not Before: Jun  9 11:58:28 2018 GMT
            Not After : Jun  6 11:58:28 2028 GMT
        Subject: C=US, ST=California, L=San Francisco, OU=peer, CN=peer0.org2.example.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:f5:69:7a:11:65:d9:85:96:65:b7:b7:1b:08:77:
                    43:de:cb:ad:3a:79:ec:cc:2a:bc:d7:93:68:ae:92:
                    1c:4b:d8:32:47:d6:3d:72:32:f1:f1:fb:26:e4:69:
                    c2:eb:c9:45:69:99:78:d7:68:a9:77:09:88:c6:53:
                    01:2a:c1:f8:c0
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:8E:58:82:C9:0A:11:10:A9:0B:93:03:EE:A0:54:42:F4:A3:EF:11:4C:82:B6:F9:CE:10:

如上可见，该命令输出了一个JSON，其中包含了被查询节点所在通道上所有节点的成员信息。

    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:29:3f:55:2b:9f:7b:99:b2:cb:06:ca:15:3f:93:
         a1:3d:65:5c:7b:79:a1:7a:d1:94:50:f0:cd:db:ea:61:81:7a:
         02:20:3b:40:5b:60:51:3c:f8:0f:9b:fc:ae:fc:21:fd:c8:36:
         a3:18:39:58:20:72:3d:1a:43:74:30:f3:56:01:aa:26
```

被返回的 `Identity` 是节点的成员增加证书，可被 `jq` 和 `openssl` 的组合进行语法分析：

Configuration query:

```
$ discover --configFile conf.yaml peers --channel mychannel  --server peer0.org1.example.com:7051
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            55:e9:3f:97:94:d5:74:db:e2:d6:99:3c:01:24:be:bf
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=California, L=San Francisco, O=org2.example.com, CN=ca.org2.example.com
        Validity
            Not Before: Jun  9 11:58:28 2018 GMT
            Not After : Jun  6 11:58:28 2028 GMT
        Subject: C=US, ST=California, L=San Francisco, OU=peer, CN=peer0.org2.example.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:f5:69:7a:11:65:d9:85:96:65:b7:b7:1b:08:77:
                    43:de:cb:ad:3a:79:ec:cc:2a:bc:d7:93:68:ae:92:
                    1c:4b:d8:32:47:d6:3d:72:32:f1:f1:fb:26:e4:69:
                    c2:eb:c9:45:69:99:78:d7:68:a9:77:09:88:c6:53:
                    01:2a:c1:f8:c0
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:8E:58:82:C9:0A:11:10:A9:0B:93:03:EE:A0:54:42:F4:A3:EF:11:4C:82:B6:F9:CE:10:

The configuration query returns a mapping from MSP IDs to orderer
endpoints, as well as the `FabricMSPConfig` which can be used to verify
```

```
all peer and orderer nodes by the SDK:

    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:29:3f:55:2b:9f:7b:99:b2:cb:06:ca:15:3f:93:
         a1:3d:65:5c:7b:79:a1:7a:d1:94:50:f0:cd:db:ea:61:81:7a:
         02:20:3b:40:5b:60:51:3c:f8:0f:9b:fc:ae:fc:21:fd:c8:36:
         a3:18:39:58:20:72:3d:1a:43:74:30:f3:56:01:aa:26
```

```
$ discover --configFile conf.yaml config --channel mychannel  --server peer0.org1.example.com:705
{
    "msps": {
        "OrdererOrg": {
            "name": "OrdererMSP",
            "root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNMekNDQWRhZ0F3SUJBZ0lSQU1pWkxUb3RmMHR6
            ],
            "admins": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNDVENDQWJDZ0F3SUJBZ0lRR2wzTjhaSzRDekRR
            ],
            "crypto_config": {
                "signature_hash_family": "SHA2",
                "identity_identifier_hash_function": "SHA256"
            },
            "tls_root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNORENDQWR1Z0F3SUJBZ0lRZDdodzFIaHNZTXI2
            ]
        },
        "Org1MSP": {
            "name": "Org1MSP",
            "root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ0lSQU1nN2VETnhwS0t0
            ],
            "admins": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNLakNDQWRDZ0F3SUJBZ0lRRTRFK0tqSHgwdTlz
            ],
            "crypto_config": {
                "signature_hash_family": "SHA2",
                "identity_identifier_hash_function": "SHA256"
            },
            "tls_root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNTVENDQWUrZ0F3SUJBZ0lRZlRWTE9iTENVUjdx
            ],
            "fabric_node_ous": {
                "enable": true,
                "client_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "client"
                },
                "peer_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "peer"
                }
            }
        },
        "Org2MSP": {
            "name": "Org2MSP",
            "root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ0lSQUx2SWV2KzE4Vm9L
            ],
            "admins": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNLVENDQWRDZ0F3SUJBZ0lRU1lpeE1vdmpoM1N2
            ],
            "crypto_config": {
                "signature_hash_family": "SHA2",
                "identity_identifier_hash_function": "SHA256"
            },
            "tls_root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNTakNDQWZDZ0F3SUJBZ0lSQUtoUFFxUGZSYnVp
            ],
            "fabric_node_ous": {
                "enable": true,
                "client_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "client"
                },
                "peer_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "peer"
                }
            }
        },
        "Org3MSP": {
            "name": "Org3MSP",
            "root_certs": [
                "CgJPVQoEUm9sZQoMRW5yb2xsbWVudElEChBSZXZvY2F0aW9uSGFuZGxlEkQKIKoEXcq/psdYnMKCiT79
            ],
            "intermediate_certs": [
                "CtgCCkQKIP0UVivtH8NlnRNrZuuu6jpaj2ZbEB4/secGS57MfbINEiDSJweLUMIQSW12jugBQG81lIQf
            ],
            "admins": [
                "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUhZd0VBWUhLb1pJemowQ0FRWUZLNEVFQUNJRFlnQUVU
            ]
        }
    },
    "orderers": {
        "OrdererOrg": {
            "endpoint": [
                {
                    "host": "orderer.example.com",
                    "port": 7050
                }
            ]
        }
    }
}
```

配置查询：

It's important to note that the certificates here are base64 encoded, and thus should decoded in a manner similar to the following:

配置查询返回了从MSP（成员服务提供者）ID到orderer端点的映射，还返回了`FabricMSPConfig`，它可被用来通过SDK验证所有peer和orderer：

```
$ discover --configFile conf.yaml config --channel mychannel  --server peer0.org1.example.com:705
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c8:99:2d:3a:2d:7f:4b:73:53:8b:39:18:7b:c3:e1:1e
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=California, L=San Francisco, O=example.com, CN=ca.example.com
        Validity
            Not Before: Jun  9 11:58:28 2018 GMT
            Not After : Jun  6 11:58:28 2028 GMT
        Subject: C=US, ST=California, L=San Francisco, O=example.com, CN=ca.example.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:28:ac:9e:51:8d:a4:80:15:0a:ff:ae:c9:61:d6:
                    08:67:b0:15:c3:c7:99:46:61:63:0a:10:a6:42:6a:
                    b0:af:14:0c:c0:e2:5b:b4:a1:c3:f0:07:7e:5b:7c:
                    c4:b2:95:13:95:81:4b:6a:b9:e3:87:a4:f3:2c:7c:
                    ae:00:91:9e:32
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Certificate Sign, CRL Sign
            X509v3 Extended Key Usage:
                Any Extended Key Usage
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                60:9D:F2:30:26:CE:8F:65:81:41:AD:96:15:0E:24:8D:A0:9D:C5:79:C1:17:BF:FE:E5:1B:FB:
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:3d:e1:a7:6c:99:3f:87:2a:36:44:51:98:37:11:
```

```
        d8:a0:47:7a:33:ff:30:c1:09:a6:05:ec:b0:53:53:39:c1:0e:
        02:20:6b:f4:1d:48:e0:72:e4:c2:ef:b0:84:79:d4:2e:c2:c5:
        1b:6f:e4:2f:56:35:51:18:7d:93:51:86:05:84:ce:1f
```

```
$ discover --configFile conf.yaml config --channel mychannel  --server peer0.org1.example.com:705
{
    "msps": {
        "OrdererOrg": {
            "name": "OrdererMSP",
            "root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNMekNDQWRhZ0F3SUJBZ0lSQU1pWkxUb3RmMHR6
            ],
            "admins": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNDVENDQWJDZ0F3SUJBZ0lRR2wzTjhaSzRDekRR
            ],
            "crypto_config": {
                "signature_hash_family": "SHA2",
                "identity_identifier_hash_function": "SHA256"
            },
            "tls_root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNORENDQWR1Z0F3SUJBZ0lRZDdodzFIaHNZTXI2
            ]
        },
        "Org1MSP": {
            "name": "Org1MSP",
            "root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ0lSQU1nN2VETnhwS0t0
            ],
            "admins": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNLakNDQWRDZ0F3SUJBZ0lRRTRFK0tqSHgwdTlz
            ],
            "crypto_config": {
                "signature_hash_family": "SHA2",
                "identity_identifier_hash_function": "SHA256"
            },
            "tls_root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNTVENDQWUrZ0F3SUJBZ0lRZlRWTE9iTENVUjdx
            ],
            "fabric_node_ous": {
                "enable": true,
                "client_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "client"
                },
                "peer_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "peer"
                }
            }
        },
        "Org2MSP": {
            "name": "Org2MSP",
            "root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ0lSQUx2SWV2KzE4Vm9L
            ],
            "admins": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNLVENDQWRDZ0F3SUJBZ0lRU1lpeE1vdmpoM1N2
            ],
            "crypto_config": {
                "signature_hash_family": "SHA2",
                "identity_identifier_hash_function": "SHA256"
            },
            "tls_root_certs": [
                "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNTakNDQWZDZ0F3SUJBZ0lSQUtoUFFxUGZSYnVp
            ],
            "fabric_node_ous": {
                "enable": true,
                "client_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "client"
                },
                "peer_ou_identifier": {
                    "certificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNSRENDQWVxZ0F3SUJBZ
                    "organizational_unit_identifier": "peer"
                }
            }
        },
        "Org3MSP": {
            "name": "Org3MSP",
            "root_certs": [
                "CgJPVQoEUm9sZQoMRW5yb2xsbWVudElEChBSZXZvY2F0aW9uSGFuZGxlEkQKIKoEXcq/psdYnMKCiT79
            ],
            "intermediate_certs": [
                "CtgCCkQKIP0UVivtH8NlnRNrZuuu6jpaj2ZbEB4/secGS57MfbINEiDSJweLUMIQSW12jugBQG81lIQf
            ],
            "admins": [
                "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUhZd0VBWUhLb1pJemowQ0FRWUZLNEVFQUNJRFlnQUVU
            ]
        }
    },
    "orderers": {
        "OrdererOrg": {
            "endpoint": [
                {
                    "host": "orderer.example.com",
                    "port": 7050
                }
            ]
        }
    }
}
```

Endorsers query:

值得注意的是，这里的证书是base64编码的，所以应该用类似以下的方法对其进行解码。

To query for the endorsers of a chaincode call, additional flags need to be supplied:

```
$ discover --configFile conf.yaml config --channel mychannel  --server peer0.org1.example.com:705
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c8:99:2d:3a:2d:7f:4b:73:53:8b:39:18:7b:c3:e1:1e
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=California, L=San Francisco, O=example.com, CN=ca.example.com
        Validity
            Not Before: Jun  9 11:58:28 2018 GMT
            Not After : Jun  6 11:58:28 2028 GMT
        Subject: C=US, ST=California, L=San Francisco, O=example.com, CN=ca.example.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:28:ac:9e:51:8d:a4:80:15:0a:ff:ae:c9:61:d6:
                    08:67:b0:15:c3:c7:99:46:61:63:0a:10:a6:42:6a:
                    b0:af:14:0c:c0:e2:5b:b4:a1:c3:f0:07:7e:5b:7c:
                    c4:b2:95:13:95:81:4b:6a:b9:e3:87:a4:f3:2c:7c:
                    ae:00:91:9e:32
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Certificate Sign, CRL Sign
            X509v3 Extended Key Usage:
                Any Extended Key Usage
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                60:9D:F2:30:26:CE:8F:65:81:41:AD:96:15:0E:24:8D:A0:9D:C5:79:C1:17:BF:FE:E5:1B:FB:
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:3d:e1:a7:6c:99:3f:87:2a:36:44:51:98:37:11:
         d8:a0:47:7a:33:ff:30:c1:09:a6:05:ec:b0:53:53:39:c1:0e:
         02:20:6b:f4:1d:48:e0:72:e4:c2:ef:b0:84:79:d4:2e:c2:c5:
         1b:6f:e4:2f:56:35:51:18:7d:93:51:86:05:84:ce:1f
```

- The `--chaincode` flag is mandatory and it provides the chaincode name(s). To query for a chaincode-to-chaincode invocation, one needs to repeat the `--chaincode` flag with all the chaincodes.
- The `--collection` is used to specify private data collections that are expected to used by the chaincode(s). To map from thechaincodes passed via `--chaincode` to the collections, the following syntax should be used: `collection=CC:Collection1,Collection2,...`.
- The `--noPrivateReads` is used to indicate that the transaction is not expected to read private data for a certain chaincode. This is useful for private data “blind writes”, among other things.

背书者查询:

For example, to query for a chaincode invocation that results in both cc1 and cc2 to be invoked, as well as writes to private data collection col1 by cc2, one needs to specify: `--chaincode=cc1 --chaincode=cc2 --collection=cc2:col1`

要想查询一个链码调用的背书者，必须提供额外的flag：

If chaincode cc2 is not expected to read from collection `col1` then `--noPrivateReads=cc2` should be used.

- `--chaincode` flag是必需的，它提供了链码名。要查询多链码的调用，必须对所有相关链码重复提供 `-chaincode` flag。

Below is the output of an endorsers query for chaincode **mycc** when the endorsement policy is `AND('Org1.peer', 'Org2.peer')`:

- `--collection` 被用来指明链码预计将使用的私有数据集合。若要把 `--chaincode` 通过的链码映射到数据集合中，应使用以下语法：`collection=CC:Collection1,Collection2,...` 。

```
$ discover --configFile conf.yaml endorsers --channel mychannel  --server peer0.org1.example.com:
[
    {
        "Chaincode": "mycc",
        "EndorsersByGroups": {
            "G0": [
                {
                    "MSPID": "Org1MSP",
                    "LedgerHeight": 5,
                    "Endpoint": "peer0.org1.example.com:7051",
                    "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRANTiKfUVHVGnrYV
                }
            ],
            "G1": [
                {
                    "MSPID": "Org2MSP",
                    "LedgerHeight": 5,
                    "Endpoint": "peer1.org2.example.com:10051",
                    "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRAIs6fFxk4Y5cJxS
                },
                {
                    "MSPID": "Org2MSP",
                    "LedgerHeight": 5,
                    "Endpoint": "peer0.org2.example.com:9051",
                    "Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQVek/l5TVdNvi1pk
                }
            ]
        },
        "Layouts": [
            {
                "quantities_by_group": {
                    "G0": 1,
                    "G1": 1
                }
            }
        ]
    }
]
```

例如，某项链码调用导致了cc1和cc2被调用，同时cc2将该链码调用写入私有数据集合cc1，要想查询该项链码调用，必须指明：`--chaincode=cc1 --chaincode=cc2 --collection=cc2:col1`

Not using a configuration file

以下显示的是当背书策略为 `AND('Org1.peer', 'Org2.peer')` 时，链码**mycc**的背书者查询的输出：

It is possible to execute the discovery CLI without having a configuration file, and just passing all needed configuration as commandline flags. The following is an example of a local peer membership query which loads administrator credentials:

```
$ discover --configFile conf.yaml endorsers --channel mychannel  --server peer0.org1.example.com:
[
    {
        "Chaincode": "mycc",
        "EndorsersByGroups": {
            "G0": [
                {
                    "MSPID": "Org1MSP",
                    "LedgerHeight": 5,
                    "Endpoint": "peer0.org1.example.com:7051",
                    "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRANTiKfUVHVGnrYV
                }
            ],
            "G1": [
                {
                    "MSPID": "Org2MSP",
                    "LedgerHeight": 5,
                    "Endpoint": "peer1.org2.example.com:10051",
                    "Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRAIs6fFxk4Y5cJxS
                },
                {
                    "MSPID": "Org2MSP",
                    "LedgerHeight": 5,
                    "Endpoint": "peer0.org2.example.com:9051",
                    "Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQVek/l5TVdNvi1pk
                }
            ]
        },
        "Layouts": [
            {
                "quantities_by_group": {
                    "G0": 1,
                    "G1": 1
                }
            }
        ]
    }
]
```

未使用配置文件

在没有配置文件的情况下也可以执行发现CLI，仅将所有需要的配置通过为命令行flag。以下是一个有关载入了管理员证书的本地节点成员查询的例子：