访问控制列表（ACL）

什么是访问控制列表

注意：这个主题在通道管理员级别处理访问控制和策略。学习链码的访问控制，请查看 chaincode for developers tutorial。

Fabric 使用权限控制列表（ACL）通过给资源关联的策略——给身份集合一个是或否的一个规则声明——来管理资源的访问权限。Fabric 包含很多默认的 ACL。在这篇文章中，我们将讨论他们是如何规定和如何覆盖默认值的。

但是在那之前，我们有必要理解一点资源和策略的内容。

资源

Fabric 的用户交互通过用户链码，系统链码，或者 事件流源来实现。因此，这些端点被视为应该在其上执行访问控制的“资源”。

应用开发者应该注意这些资源和与他们关联的默认策略。这些资源的完整列表可以在 `configtx.yaml` 中找到。你可以在这里找到 `configtx.yaml` 示例。

`configtx.yaml` 里边的资源名称详细的罗列了目前 Fabric 里边的资源。这里使用的不严格约定是 `<component>/<resource>`。所以 `cscc/GetConfigBlock` 是 `CSCC` 组件中调用的 `GetConfigBlock` 的资源。

策略

策略是 Fabric 运行的基础，因为它们允许根据与完成请求所需资源相关联的策略来检查与请求 关联的身份（或身份集）。背书策略用来决定一个交易是否被合适地背书。通道配置中定义的策 略被引用为修改策略以及访问控制，并且在通道配置本身中定义。

策略可以采用以下两种方式之一进行构造：作为 `Signature` 策略或者 `ImplicitMeta` 策略。

`Signature` 策略

这些策略标示了要满足策略而必须签名的用户。例如：

```
Policies:
  MyPolicy:
    Type: Signature
    Rule: "OR('Org1.peer', 'Org2.peer')"
```

构造的这个策略可以被解释为： *一个名为 `MyPolicy` 的策略只有被“ Org1 的节点”或着“ Org2 的节点”签名才可以通过。*

签名策略支持 `AND`， `OR` 和 `NOutOf` 的任意组合，能够构造强大的规则，比如：“组 织 A 中的一个管理员和两个其他管理员，或者20个组织管理员中的11个”。

`ImplicitMeta` 策略

`ImplicitMeta` 策略聚合配置层次结构中更深层次的策略结果，这些策略最终由签名策略 定义。他们支持默认规则，比如“组织中大多数管理员”。这些策略使用的语法和 `Signature` 策略不同但是依旧很简单： `<ALL|ANY|MAJORITY> <sub_policy>` 。

比如： `ANY` `Readers` 或者 `MAJORITY` `Admins` 。

注意，在默认策略配置中 `Admins` 有操作员角色。指定只有管理员—或某些管理员子集—可 以访问资源的策略往往是针对网络的敏感或操作方面（例如在通道上实例化链代码）。 `Writers` 表示可以提交账本更新，比如一个交易，但是不能拥有管理权限。 `Reader` 拥有被动角色。他们 可以访问信息但是没有权利提交账本更新和执行管理任务。这些默认策略可以被添加，编辑或者补 充，比如通过新的 `peer` 或者 `client` 角色（如果你拥有 `NodeOU` 支持）

这是一个 `ImplicitMeta` 策略结构的例子：

```
Policies:
  AnotherPolicy:
    Type: ImplicitMeta
    Rule: "MAJORITY Admins"
```

这里， `AnotherPolicy` 策略可以通过 `MAJORITY Admins` （大多数管理员同意）的方式 来满足。这里 `Admins` 是在通过更低级的 `Signature` 策略来满足的。

在哪里定义访问控制权限?

默认的访问控制在 `configtx.yaml` 中，这个文件由 `configtxgen` 用来编译通道配置。

访问控制可以通过两种方式中的一种来更新：编辑 `configtx.yaml` 自身，这会把 ACL 的 改变传递到所有新通道；或者通过特定通道的通道配置来更新访问控制。

如何在 `configtx.yaml` 中格式化 ACL

ACLs 被格式化为资源函数名称字符串的键值对。你可以在这里看到他们的样子示例 configtx.yaml 文件.

这个示例的两个摘录：

```
# ACL policy for invoking chaincodes on peer
peer/Propose: /Channel/Application/Writers
```

```
# ACL policy for sending block events
event/Block: /Channel/Application/Readers
```

这些 ACL 定义为对资源 `peer/Propose` 和 `event/Block` 的访问分别被限定为满足路径 `/Channel/Application/Writers` 和 `/Channel/Application/Readers` 中定义的策略的身份。

更新 `configtx.yaml` 中的默认 ACL

如果在引导网络时需要覆盖 ACL 默认值，或者在引导通道之前更改 ACL，最佳做法是更 新 `configtx.yaml`。

假如你想修改 `peer/Propose` 的默认 ACL — 为在节点上执行链码指定策略 — 从 `/Channel/Application/Writers` 到一个叫 `MyPolicy` 的策略。

这可以通过添加一个叫 `MyPolicy`（它可以是任何名字，但是在这个例子中我们称它为 `MyPolicy`）的策略来完成。这个策略定义在 `configtx.yaml` 中的 `Application.Policies` 部分，指定了一个用来检查允许或者拒绝一个用户的规则。在这个例子中，我们将创建 一个标示为 `SampleOrg.admin` 的 `Signature` 策略。

```
Policies: &ApplicationDefaultPolicies
    Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
    Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
    Admins:
        Type: ImplicitMeta
        Rule: "MAJORITY Admins"
    MyPolicy:
        Type: Signature
        Rule: "OR('SampleOrg.admin')"
```

然后，编辑 `configtx.yaml` 中的 `Application: ACLs` 部分来将 `peer/Propose` 从：

```
peer/Propose: /Channel/Application/Writers
```

改变为：

```
peer/Propose: /Channel/Application/MyPolicy
```

一旦 `configtx.yaml` 中的这些内容被改变了，`configtxgen` 工具就可以在创建交易的 时候使用这些策略和定义的 ACLs。当交易以合适的方式被联盟中的管理员签名和确认之后， 被定义了 ACLs 和策略的新通道就被创建了。

一旦 `MyPolicy` 被引导进通道配置，它就还可以被引用来覆盖其他默认的 ACL。例如：

```
SampleSingleMSPChannel:
    Consortium: SampleConsortium
    Application:
        <<: *ApplicationDefaults
        ACLs:
            <<: *ACLsDefault
            event/Block: /Channel/Application/MyPolicy
```

这将限制订阅区块事件到 `SampleOrg.admin` 的能力。

如果已经被创建的通道想使用这个 ACL，他们必须使用如下流程每次更新一个通道配置：

在通道配置中更新默认 ACL

如果已经创建的通道想使用 `MyPolicy` 来显示访问 `peer/Propose`——或者他们想创 建一个不想让其他通道知道的 ACL——他们将不得不通过配置更新交易来每次更新一个 通道。

注意：通道配置交易的过程我们在这里就不深究了。如果你想了解更多，请参考这篇文 章 channel configuration updates 和 "Adding an Org to a Channel" tutorial.

下边添加 `MyPolicy`，在这里 `Admins`， `Writers`， 和 `Readers` 都已经存在了。

```
"MyPolicy": {
  "mod_policy": "Admins",
  "policy": {
    "type": 1,
    "value": {
      "identities": [
        {
          "principal": {
            "msp_identifier": "SampleOrg",
            "role": "ADMIN"
          },
          "principal_classification": "ROLE"
        }
      ],
      "rule": {
        "n_out_of": {
          "n": 1,
          "rules": [
            {
              "signed_by": 0
            }
          ]
        }
      },
      "version": 0
    }
  },
  "version": "0"
},
```

特别注意这里的 `msp_identifer` 和 `role`。

然后，在配置中的 ACL 部分，将 `peer/Propose` 的 ACL 从：

```
"peer/Propose": {
  "policy_ref": "/Channel/Application/Writers"
```

改为：

```
"peer/Propose": {
  "policy_ref": "/Channel/Application/MyPolicy"
```

注意：如果你不想在你的通道配置中定义 ACL，你就要添加完整的 ACL 结构。

一旦配置被更新了，它就需要通过常规的通道更新过程来提交。

满足需要访问多个资源的 ACL

如果一个成员生成了一个访问多个系统链码的请求，必须满足所有系统链码的

例如，`peer/Propose` 引用通道上的任何提案请求。如果特定提案请求访问需要满足 `Writers` 身份的两个系统链码和一个需要满足 `MyPolicy` 身份的系统链码，那提交这个提案的成员就必 须拥有 `Writers` 和 `MyPolicy` 都评估为 “true” 的身份。

在默认配置中，`Writers` 是一个 `rule` 为 `SampleOrg.membe` 的签名策略。换句话说就是，“组织中的任何成员”。上边列出的 `MyPolicy`，拥有 `SampleOrg.admin` 或者 “组织中的任何 管理员”。为了满足这些 ACL，成员必须同时是一个管理员和 `SampleOrg` 中的成员。默认地， 所有管理员都是成员（尽管并非所有管理员都是成员），但可以将这些策略覆盖为你希望的任何 成员。因此，跟踪这些策略非常重要，以确保节点提案的 ACL 不是不可能满足的（除非是这样）。