背书策略

每个链码都有背书策略，背书策略指定了通道上的一组 Peer 节点必须执行链码，并且为执行结果进行背书，以此证明交易是有效的。这些背书策略指定了必须为提案进行背书的组织。

> **注解**
>
> 回想一下 **状态**，从区块链数据中分离出来，用键值对表示。更多信息请查看 账本 文档。

作为 Peer 节点进行交易验证的一部分，每个 Peer 节点的检查确保了交易保存了合适 **数量** 的背书，并且是指定背书节点的背书。这些背书结果的检查，同样确保了它们是有效的（比如，从有效的证书得到的有效签名）。

需要背书的多种方式

默认情况下，背书策略在链码定义中指定， 它被通道成员同意然后提交到通道（背书策略包含了所有和链码相关的状态）。

针对私有数据收集策略，你也可以在私有数据收集层面指定一个背书策略，它会为任何私有数据集合的键重写链码级别的背书策略， 因此进一步限制哪个组织可以写进一个私有数据集合。

最后，有案例表明对于一个特定的公有通道状态或者私有数据收集状态（换句话说，一个特定的键值对）用不同状态背书策略很重要。 这个 **基于状态的背书** 允许链码级别或者集合级别的背书策略被不同的策略针对特定的键重写。

为了解释哪些情况下使用这多种背书策略，请考虑在通道上实现汽车交易的情况。创建（也称为“发行”）一辆汽车作为可交易的资产（即把一个键值对存到世界状态）需要满足链码级别的背书策略。下文将详细介绍链码级别的背书策略。

如果代表汽车的键需要特殊的背书策略，该背书策略可以在汽车创建或之后被定义。当下有很多为什么需要特定状态的背书策略的原因，汽车可能具有历史重要性或价值，因此有必要获得持牌估价师的认可。还有，汽车的主人(如果他们是通道的成员)可能还希望确保他们的同伴在交易上签名。这两种情况，**都需要为特殊资产指定，与当前链码默认背书策略不同的背书策略。**

我们将在后面介绍如何定义基于状态的背书策略。但首先，让我们看看如何设置链式代码级的背书策略。

设置链码级背书策略

当通道成员为它们的组织批准一个链码定义时，链码级别的策略被它们（通道成员）同意。 足够数量的通道成员需要批准链码定义来满足 `Channel/Application/LifecycleEndorsement` 策略，在定义能够被提交到通道之前，默认设置为大多数通道成员（译者注：在链码定义能够提交到通道之前需要大多数通道成员批准定义）。 一旦定义被提交，链码可以使用了。任何把数据写进账本的链码调用需要被足够多的通道成员验证以此来满足背书策略。

你可以使用 Fabric SDK 来为链码指定一个背书策略。 例如，参考在 Node.js SDK 文档中的文章 How to install and start your chaincode。 当你通过 Fabric peer 二进制可执行文件并使用 `--signature-policy` 标签来同意提交链码定义时，你也可以通过 CLI 工具来创建一个背书策略。

> **注解**

现在不要担心背书策略语法 (比如 `'Org1.member'`)，我们后面部分会介绍。

例如:

```
peer lifecycle chaincode approveformyorg --channelID mychannel --signature-policy "AND('Org1.membe
```

以上指令通过同意策略为 `AND('Org1.member', 'Org2.member')` 的需要 Org1 和 Org2 都为交易签名的 `mycc` 链码定义。 在足够的通道成员同意 `mycc` 链码定义后，定义和背书策略能使用以下指令被提交到通道上:

```
peer lifecycle chaincode commit -o orderer.example.com:7050 --channelID mychannel --signature-poli
```

注意，如果开启了身份类型 (见 成员服务提供者 (MSP)), 可以使用 `PEER` 角色限定使用 Peer 进行背书。

例如:

```
peer lifecycle chaincode approveformyorg --channelID mychannel --signature-policy "AND('Org1.peer'
```

除了从 CLI 或者 SDK 来指定背书策略，链码也可以使用通道配置中的策略来作为背书策略。 你可以使用 `--channel-config-policy` 标签来选择通道策略，此策略的格式被通道配置和 ACLs 使用。

例如:

```
peer lifecycle chaincode approveformyorg --channelID mychannel --channel-config-policy Channel/App
```

如果你不指定策略，链码定义会默认使用 `Channel/Application/Endorsement` 策略，这需要一笔交易被大多数通道成员验证。 此策略建立在通道上的成员基础上，所以当通道的组织增加或减少时需要自动更新。 使用通道策略的一个优势是它们可以随着通道成员关系变化自动更新。

如果你使用 `--signature-policy` 标签或者 SDK 指定一个背书策略，当组织加入或离开通道时你需要更新策略。 在链码被定义后一个新的组织加入通道能够查询一个链码（倘若 query 具有合适的授权正如通道策略和链码执行的任何应用级别的检查） 但是不能执行或背书链码。只有背书策略语句列举的组织能够为交易签名。

背书策略语法

正如你上面所看到了，策略是使用主体来表达的（主题是跟角色匹配的）。主体可以描述为 `'MSP.ROLE'`， `MSP` 代表了 MSP ID， `ROLE` 是以下4个其中之一： `member`, `admin`, `client`, and `peer`。

以下是几个有效的主体示例:

- `'Org0.admin'`: `Org0` MSP 的任何管理员
- `'Org1.member'`: `Org1` MSP 的任何成员
- `'Org1.client'`: `Org1` MSP 的任何客户端
- `'Org1.peer'`: `Org1` MSP 的任何 Peer

语法是:

`EXPR(E[, E...])`

`EXPR` 可以是 `AND`，`OR`，或者 `OutOf`，并且 `E` 是一个以上语法的主体或者另外一个 `EXPR`。

比如:

- `AND('Org1.member', 'Org2.member', 'Org3.member')` 要求3个组织的都至少一个成员进行签名。
- `OR('Org1.member', 'Org2.member')` 要求组织1或者组织2的任一成员进行签名。
- `OR('Org1.member', AND('Org2.member', 'Org3.member'))` 要求组织1的任一成员签名，或者组织2和组织3的任一成员，分别进行签名。
- `OutOf(1, 'Org1.member', 'Org2.member')`，等价于``OR('Org1.member', 'Org2.member')``。
- 类似的，`OutOf(2, 'Org1.member', 'Org2.member')` 等价于 `AND('Org1.member', 'Org2.member')`，`OutOf(2, 'Org1.member', 'Org2.member', 'Org3.member')` 等价于 `OR(AND('Org1.member', 'Org2.member'), AND('Org1.member', 'Org3.member'), AND('Org2.member', 'Org3.memb`

设置集合级别的背书策略

和链码级别的背书策略类似的，当你批准提交链码定义，你也可以指定链码的私有数据收集和相关的集合级别的背书策略。 如果一个集合级别的背书策略设置了，写进私有数据集合键的私有数据会要求特定的组织的 peer 节点为交易背书。

你可以使用集合级别的背书策略来限制哪个组织的 peer 节点可以对私有数据集合键域名进行写操作， 例如确保非授权组织不能对集合进行写操作，以及能证明任何在私有数据集合中的状态已经被需要的收集组织背书。

集合级别的背书策略可能限制会更少或者比链码级别的背书策略以及收集的私有数据分发策略限制更少。 例如大多数组织可能需要为一个链码交易背书，但是特定的组织可能需要为一笔在特定集合中包含一个键的交易背书。

集合级别的背书策略语句准确地匹配链码级别的背书策略语句 — 在收集配置中你可以指定 `endorsementPolicy` 和 `signaturePolicy` 或者 `channelConfigPolicy`。更多细节查看 私有数据。

设置键级别的背书策略

设置链码级别或者集合级别的背书策略跟对应的链码生命周期有关。可以在通道实例化或者升级对应链码的时候进行设置。

对比来看, 键级别的背书策略可以在链码内更加细粒度的设置和修改。修改键级别的背书策略是常规交易读写集的一部分。

shim API提供了从常规Key设置和获取背书策略的功能。

注解

下文中的 `ep` 代表背书策略，它可以用上文介绍的语法所描述，或者下文介绍的函数。每种方法都会生成，可以被 shim API 接受的二进制版本的背书策略。

```
SetStateValidationParameter(key string, ep []byte) error
GetStateValidationParameter(key string) ([]byte, error)
```

对于在 Collection 中属于 私有数据 使用以下函数:

```
SetPrivateDataValidationParameter(collection, key string, ep []byte) error
GetPrivateDataValidationParameter(collection, key string) ([]byte, error)
```

为了帮助把背书策略序列化成有效的字节数组，shim提供了便利的函数供链码开发者，从组织 MSP 标示符的角度处理背书策略，详情见 键背书策略:

```
type KeyEndorsementPolicy interface {
    // Policy returns the endorsement policy as bytes
    Policy() ([]byte, error)

    // AddOrgs adds the specified orgs to the list of orgs that are required
    // to endorse
    AddOrgs(roleType RoleType, organizations ...string) error

    // DelOrgs delete the specified channel orgs from the existing key-level endorsement
    // policy for this KVS key. If any org is not present, an error will be returned.
    DelOrgs(organizations ...string) error

    // ListOrgs returns an array of channel orgs that are required to endorse changes
    ListOrgs() ([]string)
}
```

比如，当两个组织要求为键值的改变背书时，需要设置键背书策略，通过把 `MSPIDs` 传递给 `AddOrgs()` 然后调用 `Policy()` 来构建字节数组格式的背书策略，之后传递给 `SetStateValidationParameter()`。

把 shim 作为链码的依赖请参考 管理 Go 链码的扩展依赖。

验证

commit交易时，设置键值的过程和设置键的背书策略的过程是一样的，都会更新键的状态并且使用相同的规则进行验证。

Validation	no validation parameter set	validation parameter set
modify value	check chaincode or collection ep	check key-level ep
modify key-level ep	check chaincode or collection ep	check key-level ep

正如上面讨论的，如果一个键并改变了，并且没有键级别的背书策略，默认会使用链码级别或集合级别的背书策略。设置键级别背书策略的时候，也是使用链码级背书策略，即新的键级别背书策略必须使用已存在的链码背书策略。

如果某个键被修改了，并且键级别的背书策略已经设置，键级别的背书策略就会覆盖链码级别或集合级别背书策略。实际上，键级背书策略可以比链码级别或集合级别背书策略宽松或者严格，因为设置键级背书策略必须满足链码级别或集合级别背书策略，所以没有违反可信的假设。

如果某个键级背书策略被移除（或设为空），链码级别或集合级别背书策略再次变为默认策略。

如果某个交易修改了多个键，并且这些键关联了多个键级背书策略，交易需要满足所有的键级策略才会有效。