使用 CouchDB 作为状态数据库

状态数据库选项

当前Peer状态数据库可以选择LevelDB和CouchDB。LevelDB是peer进程默认内置的key-value状态数据库。CouchDB是一个替代的外部状态数据库。 与LevelDB key-value存储类似，CouchDB可以存储链码中建模的任何二进制数据(对于非JSON格式的数据，Couchdb里面将其视为attachements)。作为一个文档对象存储， CouchDB允许您将数据以JSON格式进行存储，对您的数据进行富查询，并使用索引支持您的查询。

LevelDB和CouchDB都支持核心链码操作，例如获取和设置密钥（资产）以及基于密钥的查询。键可以按范围查询，复合键可以建模，以便对多个参数进行等价查询。例如，``owner,asset_id``的复合键可以用来查询某个实体拥有的所有资产。这些基于关键字的查询可用于对帐本的只读查询，也可用于更新帐本的事务处理中。

用JSON对数据建模允许您针对数据的值发出富查询，而不是只能查询键。这使得应用程序和链码更容易读取存储在区块链账本上的数据。使用CouchDB可以帮助您满足LevelDB不支持的许多用例的审计和报告需求。如果使用CouchDB并用JSON对数据建模，还可以使用链码部署索引。 使用索引使查询更加灵活和高效，并使您能够从链码中查询大型数据集。

CouchDB作为一个独立的数据库进程与peer进程一起运行，因此在设置、管理和操作方面还有其他考虑。您可以考虑从默认的嵌入式LevelDB开始，如果您需要额外的复杂富查询，则可以迁移到CouchDB。 将资产数据建模为JSON是一种很好的做法，这样您就可以选择在将来需要时执行复杂的富查询。

注意

CouchDB JSON 文档只能包含合法的 UTF-8 字符串并且不能以下划线开头（“_”）。无论你使用 CouchDB 还是 LevelDB 都不要在键中使用 U+0000 （空字节）。

CouchDB JSON 文档中不能使用一下值作为顶字段的名字。这些名字为内部保留字段。 - `任何以下划线开头的字段，“_”` - `~version`

从链码中使用 CouchDB

链码查询

链码 API 中大部分方法在 LevelDB 或者 CouchDB 状态数据库中都可用，例如 `GetState`、`PutState`、`GetStateByRange`、`GetStateByPartialCompositeKey`。另外当你使用 CouchDB 作为状态数据库并且在链码中以 JSON 建模资产的时候，你可以使用 `GetQueryResult` 通过向 CouchDB 发送查询字符串的方式使用富查询。查询字符串请参考 CouchDB JSON 查询语法 。

marbles02 示例 演示了如何从链码中使用 CouchDB 查询。它包含了一个 `queryMarblesByOwner()` 方法，通过向链码传递所有者 id 来演示如何通过参数查询。它还使用 JSON 查询语法在状态数据中查询符

合 “docType” 的弹珠的所有者 id：

```
{"selector":{"docType":"marble","owner":<OWNER_ID>}}
```

对富查询的响应有助于理解账本上的数据。但是，不能保证富查询的结果集在链码执行和提交时间之间是稳定的。因此，您不应该在单个事务中使用富查询和更新通道账本。例如，如果您对Alice拥有的所有资产执行富查询并将它们传输给Bob，则链码执行时间和提交时间之间的另一个事务可能会将新资产分配给Alice。

CouchDB 分页

Fabric 支持对富查询和范围查询结果的分页。API 支持范围查询和富查询使用页大小和书签进行分页。要支持高效的分页，必须使用 Fabric 的分页 API。特别地，CouchDB 不支持 `limit` 关键字，分页是由 Fabric 来管理并隐式地按照 pageSize 的设置进行分页。

如果是通过查询 API（`GetStateByRangeWithPagination()`、`GetStateByPartialCompositeKeyWithPagination()`、和 `GetQueryResultWithPagination()`）来指定 pageSize 的，返回给链码的结果（以 pageSize 为范围）会带有一个书签。该书签会返回给调用链码的客户端，客户端可以根据这个书签来查询结果的下一“页”。

分页 API 只能用于只读交易中，查询结果旨在支持客户端分页的需求。对于需要读和写的交易，请使用不带分页的链码查询 API。在链码中，您通过迭代的方式来获取你想要的深度。

无论是否使用了分页 API，所有链码查询都受限于 `core.yaml` 中的 `totalQueryLimit`（默认 100000）。这是链码将要迭代并返回给客户端最多的结果数量，以防意外或者恶意地长时间查询。

注解

无论链码中是否使用了分页，节点都会根据 `core.yaml` 中的 ``internalQueryLimit``（默认 1000）来查询 CouchDB。 这样就保证了在执行链码的时候有合理大小的结果在节点和 CouchDB 之间，以及链码和客户端之间传播。

在 使用 CouchDB 教程中有一个使用分页的示例。

CouchDB 索引

CouchDB 中的索引用来提升 JSON 查询的效率以及按顺序的 JSON 查询。索引可以让你在账本中有大量数据时进行查询。 索引可以在 `/META-INF/statedb/couchdb/indexes` 文件夹中和链码打包在一起。每一个索引文件必须定义在一个扩展名为 `*.json` 的文本文件中，文件内容符合 CouchDB 索引 JSON 语法 。例如，要想支持上边提到的弹珠查询，提供了一个 `docType` 和 `owner` 字段的简单索引文件：

```
{"index":{"fields":["docType","owner"]},"ddoc":"indexOwnerDoc", "name":"indexOwner","type":"json"
```

索引文件可以在 这里 找到。

链码的``META-INF/statedb/couchdb/indexes``目录中的任何索引将与链码打包以进行部署。当链码包安装在peer上并且链码定义提交到通道时，索引将部署到peer通道和特定于链码的数据库。如果您先安装链码，然后将链码定义提交到通道，那么将在提交时部署索引。如果已经在通道上定义了链码，并且链码包随后安装在加入通道的peer上，则将在链码**安装**时部署索引。

部署之后，调用链码查询的时候会自动使用索引。CouchDB 会根据查询的字段选择使用哪个索引。或者，在查询选择器中通过 `use_index` 关键字指定要使用的索引。

安装的不同版本的链码可能会有相同版本的索引。要更改索引，需要使用相同的索引名称但是不同的索引定义。在安装或者实例化完成的时候，索引就会重新被部署到 Peer 节点的状态数据库了。

如果你已经有了大量的数据，然后才安装或者初始化链码，在安装或初始化的过程中索引的创建可能会花费一些时间。 同样，如果你已经有了大量的数据，然后提交后续版本的链码定义，也会花费一些时间创建索引。. 在索引创建的过程中请不要调用来嘛查询状态数据库。在交易的过程中，区块提交到账本后索引会自动更新。如果安装链码的过程中 Peer 节点崩溃了，couchdb 的索引可能就没有创建成功。这种情况下，你需要重新安装链码来创建索引。

CouchDB 配置

通过在 `stateDatabase` 状态选项中将 goleveldb 切换为 CouchDB 可以启用 CouchDB 状态数据库。另外配置 `couchDBAddress` 来指向 Peer 节点所使用的 CouchDB。如果 CouchDB 设置了用户名和密码，也需要在配置中指定。其他的配置选项在 `couchDBConfig` 部分也都有相关说明。重启 Peer 节点就可以使 *core.yaml* 文件立马生效。

你也可以使用环境变量来覆盖 core.yaml 中的值，例如 `CORE_LEDGER_STATE_STATEDATABASE` 和 `CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS` 。

下边是 *core.yaml* 中的 `stateDatabase` 部分：

```
state:
  # stateDatabase - options are "goleveldb", "CouchDB"
  # goleveldb - default state database stored in goleveldb.
  # CouchDB - store state database in CouchDB
  stateDatabase: goleveldb
  # Limit on the number of records to return per query
  totalQueryLimit: 10000
  couchDBConfig:
     # It is recommended to run CouchDB on the same server as the peer, and
     # not map the CouchDB container port to a server port in docker-compose.
     # Otherwise proper security must be provided on the connection between
     # CouchDB client (on the peer) and server.
     couchDBAddress: couchdb:5984
     # This username must have read and write authority on CouchDB
     username:
     # The password is recommended to pass as an environment variable
     # during start up (e.g. LEDGER_COUCHDBCONFIG_PASSWORD).
     # If it is stored here, the file must be access control protected
     # to prevent unintended users from discovering the password.
     password:
     # Number of retries for CouchDB errors
     maxRetries: 3
     # Number of retries for CouchDB errors during peer startup
     maxRetriesOnStartup: 10
     # CouchDB request timeout (unit: duration, e.g. 20s)
     requestTimeout: 35s
     # Limit on the number of records per each CouchDB query
     # Note that chaincode queries are only bound by totalQueryLimit.
```

```
    # Internally the chaincode may execute multiple CouchDB queries,
    # each of size internalQueryLimit.
    internalQueryLimit: 1000
    # Limit on the number of records per CouchDB bulk update batch
    maxBatchUpdateSize: 1000
    # Warm indexes after every N blocks.
    # This option warms any indexes that have been
    # deployed to CouchDB after every N blocks.
    # A value of 1 will warm indexes after every block commit,
    # to ensure fast selector queries.
    # Increasing the value may improve write efficiency of peer and CouchDB,
    # but may degrade query response time.
    warmIndexesAfterNBlocks: 1
```

Hyperledger Fabric 提供的 CouchDB docker 镜像可以通过 Docker Compose 脚本来定义 `COUCHDB_USER` 和 `COUCHDB_PASSWORD` 环境变量，从而设置 CouchDB 管理员的用户名和密码。

如果没有使用 Fabric 提供的 docker 镜像安装 CouchDB，必须编辑 local.ini 文件 来设置管理员的用户名和密码。

Docker Compose 脚本只能在创建容器的时候设置用户名和密码。在容器创建之后，必须使用 *local.ini* 文件来修改用户名和密码。

如果您选择将fabric-couchdb容器端口映射到主机端口，请确保您知道安全隐患。在开发环境中映射CouchDB容器端口将公开CouchDB REST API，并允许您通过CouchDB web界面（Fauxton）可视化数据库。在生产环境中，应该避免映射主机端口以限制对CouchDB容器的访问。只有peer才能访问CouchDB容器。

注意

每次 Peer 节点启动的时候都会读取 CouchDB 节点的选项。

查询练习

避免对将导致扫描整个 CouchDB 数据库的；链码查询。全长数据库扫描将导致较长的响应时间，并将降低您的网络性能。您可以采取以下一些步骤来避免长时间查询：

- 使用 JSON 查询：
    - 确保在链码包中创建了索引。
    - 不要使用 `$or`、`$in` 和 `$regex` 之类会扫描整个数据库的操作。
- 对于范围查询、复合键查询和 JSON 查询：
    - 使用分页查询，不要使用一个大的查询结果。
- 如果在您的应用中想创建一个仪表盘（dashboard）或者聚合数据，您可以将区块链数据复制到链下的数据库中，通过链下数据库来查询或分析区块链数据，以此来优化数据存储，并防止网络性能的降低或交易的终端。要实现这个功能，可以通过区块或链码事件将交易数据写入链下数据库或者分析引擎。对于每一个接收到的区块，区块监听应用将遍历区块中的每一个交易并根据每一个有效交易的 `读写集` 中的键值对构建一个数据存储。文档 基于通道的 Peer 节点事件服务 提供了可重放事件，以确保下游数据存储的完整性。