# FISCO BCOS 2.0使用教程： 分布式存储体验

原创 莫楠 FISCO BCOS开源社区 2019-04-08

莫楠

FISCO BCOS 高级架构师

容量的问题都不是问题

AUTHOR | 作者

《分布式存储架构设计》一文发布后，社区成员对技术内核及使用非常关注。团队和社区热心小伙伴、业内专家针对分布式存储，进行了一箩筐地讨论。在此，跟大家分享交流心得，或有助你更好地理解和使用分布式存储：

- FISCO BCOS 2.0的分布式存储采用库表风格，CRUD操作符合业务习惯。
- 不用合约存储变量模式，解构了合约和数据的内嵌式耦合，合约升级更容易。
- 存储访问引擎逻辑和数据结构更直观，容易适配各种存储引擎，扩展空间大。
- 数据本身行列式存储，没有MPT树那般盘根错节的关系，更容易打快照和切割迁移。
- 表加主键的结构索引数据，存取效率高，并发访问更容易。
- 存储开销更少，容量模型和交易数、状态数线性相关，更容易预测业务容量，对海量服务非常有意义。
- 细节方面，弱化状态MPT，但保留了交易和回执MPT，依旧可支持轻客户端，采用过程证明和存在证明，而不依赖易变的状态，不影响实现跨链。
- 状态由增量HASH检验，每块交易产生的状态集会全网严格检验以保证一致性。
- 一开始是面向SQL类型构建的，可支持MySQL和Oracle等引擎，然后适配到NoSQL类型，如LevelDB等。后续还会适配更多高速和海量存储引擎，在【单次io延迟/并发效率/容量扩展】这个三角关系中，探索出最优解。

分布式存储虽说是个大工程（团队几个快枪手撸了小一年才敢拿出来见人），但使用非常简单，本文就讲讲分布式存储的体验流程。初步接触用户，建议先从上篇入手（点标题可直接跳转）→ 分布式存储架构设计

## 配置分布式存储

分布式存储支持多种存储引擎，根据业务需求和部署环境灵活选择，可以配置为不同的存储引擎。

区块链的区块、交易等基础数据采用库表结构保存，状态数据的存储方式可配为库表结构或

MPT，满足不同场景的需求。

分布式存储的配置项位于群组的配置文件中，各个群组可以使用单独的存储策略，群组配置文件位于区块链节点中名为conf/group.[群组号].genesis的路径下，如group.1.genesis，一旦群组启动，该群组的分布式存储的相关配置不能再改变。

**分布式存储配置项示例如下：**

[storage]
type=LevelDB：分布式存储的DB引擎类型，支持”LevelDB”和“External”（rc2版本）
[state]
type=storage：state类型，目前支持storage state和MPT state，默认为storage state

**推荐使用 storage state ，除非必须使用MPT来追溯全局历史状态，不建议使用MPT State。**

## 使用CRUD智能合约开发

分布式存储提供了专用的CRUD接口，支持合约直接访问底层的存储表。

访问CRUD需要引用分布式存储专用的智能合约Table.sol接口，该接口是数据库合约，可以创建表，并对表进行增删改查操作。

**引用Table.sol**
import "./Table.sol";

**Table.sol的接口包括：**

- createTable //创建表
- select(string, Condition) //查询数据
- insert(string, Entry) //插入数据
- update(string, Entry, Condition) //更新数据
- remove(string, Condition) //删除数据

**每个接口的用法如下：**

### 创建表

```
// TableFactory的地址固定为0x1001
TableFactory tf = TableFactory(0x1001);

// 创建t_test表，表的key_field为name，value_field为item_id,item_name
// key_field表示分布式存储主key value_field表示表中的列，可以有多列，以逗号分隔
int count = tf.createTable("t_test", "name", "item_id,item_name");
```

### 查询数据

```
TableFactory tf = TableFactory(0x1001);
Table table = tf.openTable("t_test");

// 条件为空表示不筛选 也可以根据需要使用条件筛选
Condition condition = table.newCondition();

Entries entries = table.select(name, condition);
```

### 插入数据

```
TableFactory tf = TableFactory(0x1001);
Table table = tf.openTable("t_test");

Entry entry = table.newEntry();
entry.set("name", name);
entry.set("item_id", item_id);
entry.set("item_name", item_name);

int count = table.insert(name, entry);
```

### 更新数据

```
TableFactory tf = TableFactory(0x1001);
Table table = tf.openTable("t_test");

Entry entry = table.newEntry();
entry.set("item_name", item_name);

Condition condition = table.newCondition();
condition.EQ("name", name);
condition.EQ("item_id", item_id);

int count = table.update(name, entry, condition);
```

### 删除数据

```
TableFactory tf = TableFactory(0x1001);
Table table = tf.openTable("t_test");

Condition condition = table.newCondition();
condition.EQ("name", name);
condition.EQ("item_id", item_id);

int count = table.remove(name, condition);
```

**PS/**

存储架构的优化是个基础工程，也是个大工程。实现的转变，其实是架构世界观的一种进化，影响会比看到的功能点更深远。此二文所讲述的，仅是分布式存储的冰山一角。更多原理和使用案例，请参考：
https://fisco-bcos-documentation.readthedocs.io/zh_CN/release-2.0/docs/manual/smart_contract.html

**再PS/**

**下篇预告：预编译合约架构设计**

**#系列文章推荐#**


原理解析：群组架构的设计
使用教程：群组架构实操演练

系列文章统一集合到**【公众号菜单栏】>>【知识库】>>【开发教程】**中，便于系统学习和快速查找。

**FISCO BCOS的代码完全开源且免费**

**下载地址↓↓↓**

**https://github.com/fisco-bcos**