Ethereum uses levelDB as backend DB at the moment. (2018/03/08) While there exists many way to replace it to another storage, I'll introduce a way using *Apace Thrift* to exchange levelDB to the my own implemented DB called RTI written in c++. (I've implemented it w/ a few colleagues) Followings are slides that are explaining overall process. I added some comments to this pages to provide some more details; I'll write a full document including details in near future so that other people could easily follow and do a practice.

Page 1. Overview (HowTo)

A flow diagram what to do for replacing backend DB.

Page 2. ethdb/interface.go

Ethereum DB interface; you should cover functions in Database interface.

Page 3. Apache Thrift

A brief introduction of a framework, which I used to make go source file w/ c++ database code.

Page 4. Impl. Counterpart of DB func. // Page 5. Impl. RTI.go

In fact, this section is the core part but slides are missing far a lot details. As the "Overview (HowTo)" slide, you need to impl. 1) counter part for DB functions and 2) actual RTIDB.go (the one instead of levelDB). For each, you should do as followings.

1) Impl. Counter part for DB functions

On the page 2. There exists only 4 functions, but to use my own DB, I added set_up_ethdb which makes appropriate tables to store block, transaction, etc.

A. Write down the interface for each functions in ethdb/interface.go on ethdb.thrift (as the apache thrift's syntax)

B. Impl. each functions to work well w/ my own DB. In c++ on interface_server.cc

C. $) thrift --gen go ethdb.thrift => type this on shell auto-generates functions (*written in go!*) in ethdb.thrift. You can use this functions in the next step.

2) Impl. RTI.go

There exists database.go which include actual codes for levelDB. It has several types, functions and we need to impl. our own functions w/ autogenerated codes above. First of all, copy & paste generated source files to go-ethereum/ethdb to utilize them.

A. I replace db-specific functions with my own functions except the meter. I just re-use it in origin database.go.

B. TODO(grhan): I'll fill this section w/ code examples for easy-understanding.

Page 5. Codes should be changed

Now we impl. our own NewDB.go! (in my case, RTIDB.go). So what you should do is just exchange the calling levelDB part to your own one. I listed some key place and put an example how I replace it.

Page 6.   Requisites

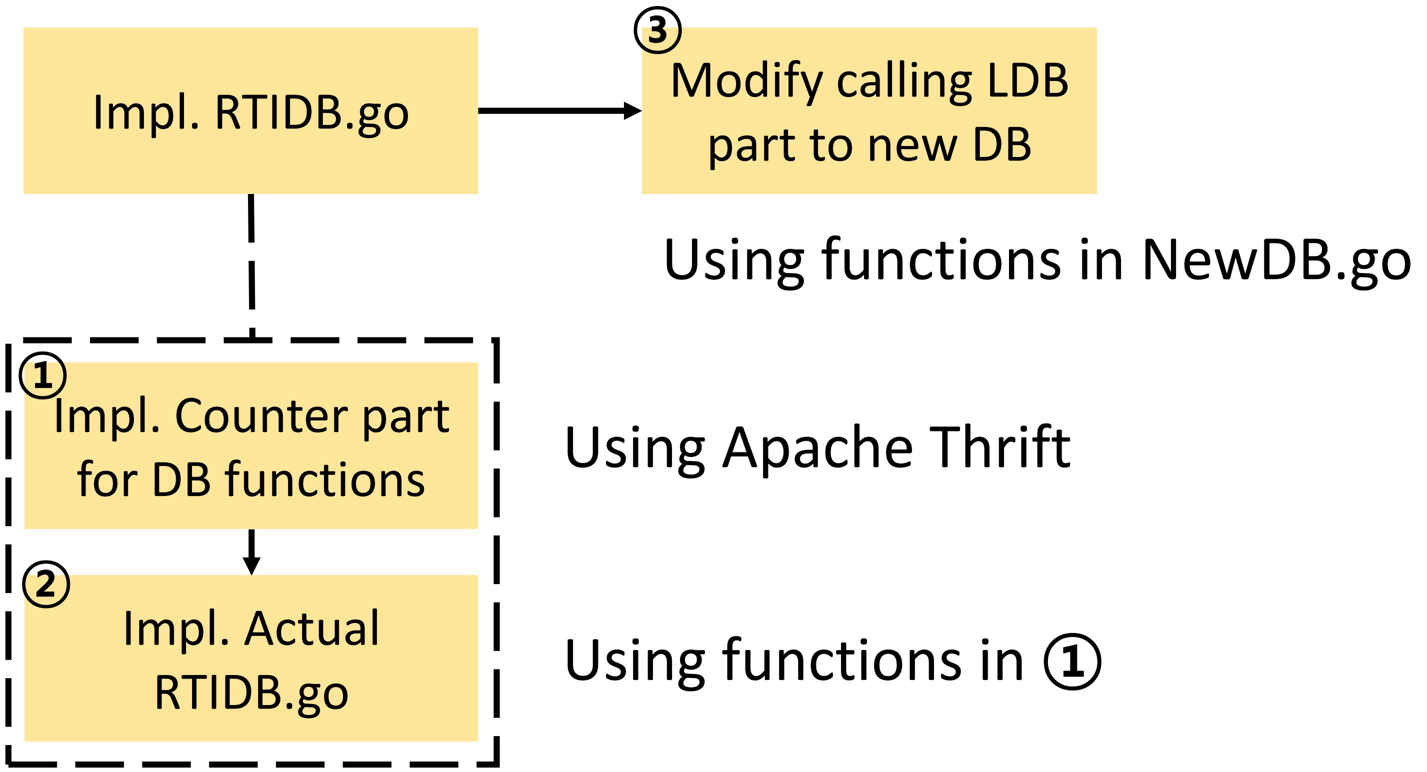
All development and testing is done under linux (centOS) environment. You have to install apache thrift on it, have your own database code (it doesn't have to be written in c++. Thrift supports various language)

Exchange Ethereum Backend DB

한겨레
kr8534@gmail.com

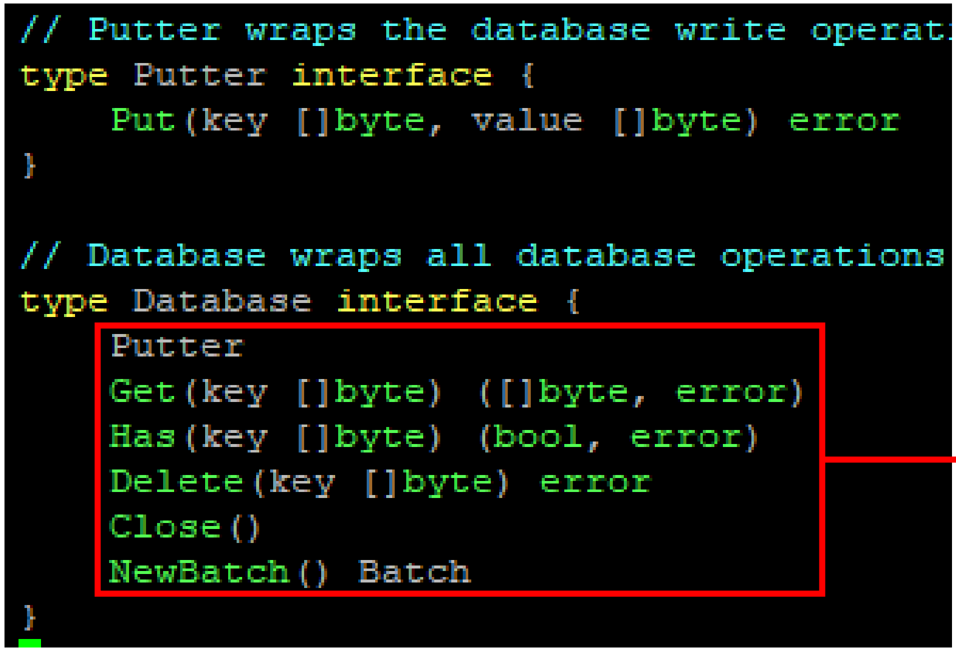
Because this ppt form skip some details, I'll write a doc instead of this ppt form in near future

Overview (HowTo)


Impl. RTIDB.go
③
Modify calling LDB part to new DB
Using functions in NewDB.go
①
Impl. Counter part for DB functions
Using Apache Thrift
②
Impl. Actual RTIDB.go
Using functions in ①

ethdb/Interface.go

- Go interface ≈ C++ pure abstract class
- All methods in interface should be treated to use it



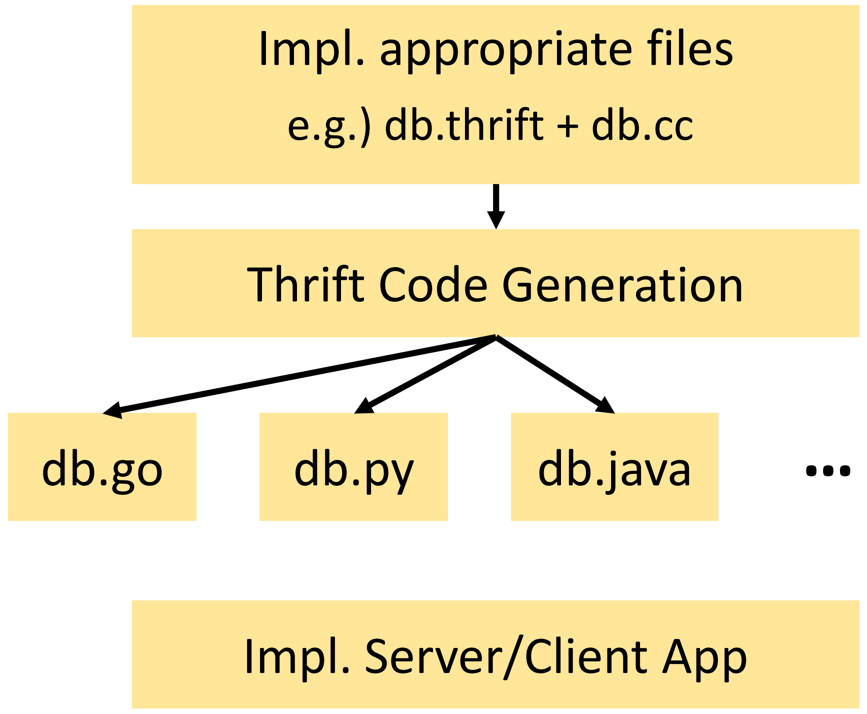
```
// Putter wraps the database write operat
type Putter interface {
	Put(key []byte, value []byte) error
}

// Database wraps all database operations
type Database interface {
	Putter
	Get(key []byte) ([]byte, error)
	Has(key []byte) (bool, error)
	Delete(key []byte) error
	Close()
	NewBatch() Batch
}
```

Just need to cover this for using new “database” type

Apache Thrift

- Software framework, for scalable cross-language services development, combines a software stack with a code generation engine

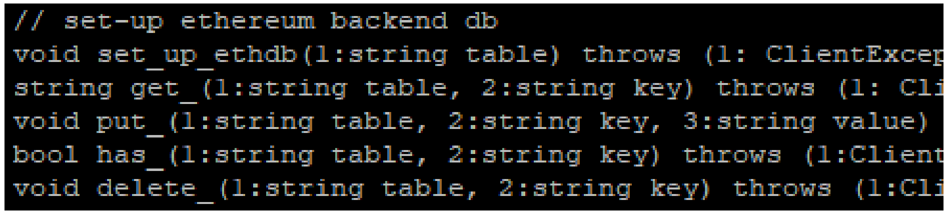


Define data types and interfaces

Support 18 Languages (2018.01.17)
C/C++, C#, Go, Java, Python, Ruby, PHP, Haskell, JavaScript, node.js, Perl, …

Server/Client impl
w/ auto-generated functions

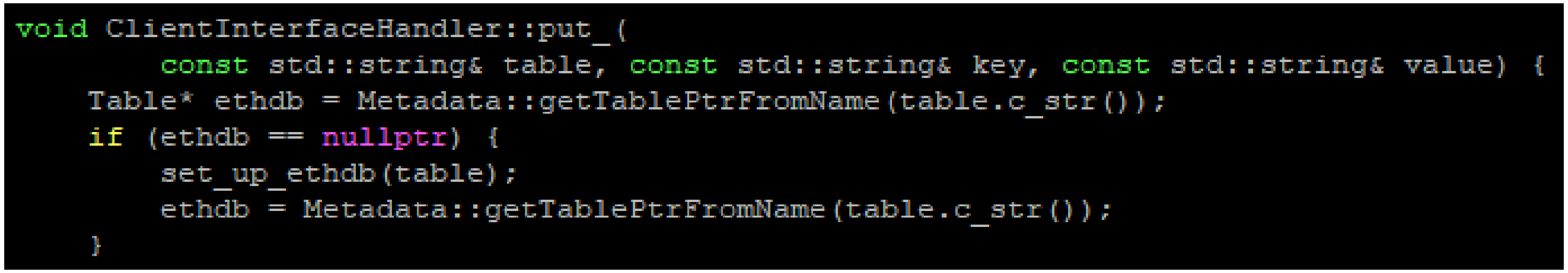
Impl. Counterpart of DB func.



```
// set-up ethereum backend db
void set_up_ethdb(1:string table) throws (1: ClientExcep
string get_(1:string table, 2:string key) throws (1: Cli
void put_(1:string table, 2:string key, 3:string value)
bool has_(1:string table, 2:string key) throws (1:Client
void delete_(1:string table, 2:string key) throws (1:Cli
```

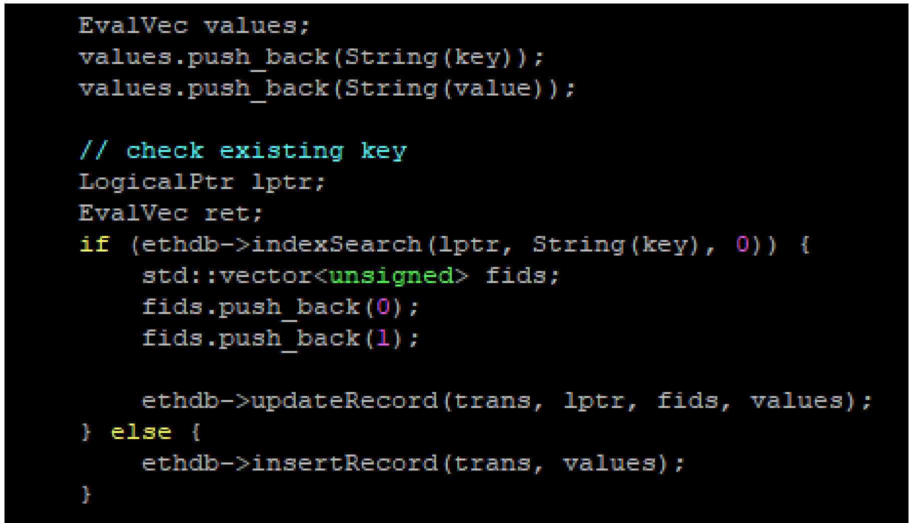
ethdb.thrift

Interface_server.cc (part of it)



```
void ClientInterfaceHandler::put_(
        const std::string& table, const std::string& key, const std::string& value) {
    Table* ethdb = Metadata::getTablePtrFromName(table.c_str());
    if (ethdb == nullptr) {
        set_up_ethdb(table);
        ethdb = Metadata::getTablePtrFromName(table.c_str());
    }
```

…



```
    EvalVec values;
    values.push_back(String(key));
    values.push_back(String(value));

    // check existing key
    LogicalPtr lptr;
    EvalVec ret;
    if (ethdb->indexSearch(lptr, String(key), 0)) {
        std::vector<unsigned> fids;
        fids.push_back(0);
        fids.push_back(1);

        ethdb->updateRecord(trans, lptr, fids, values);
    } else {
        ethdb->insertRecord(trans, values);
    }
```

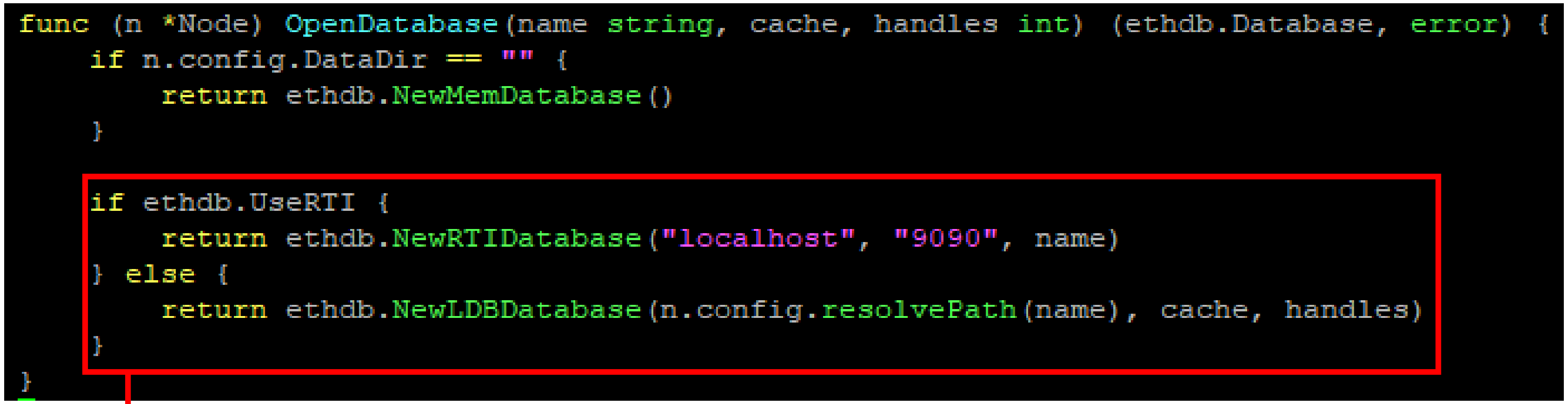
Impl. RTI.go

- AS-IS: database.go (leveldb)
- For basic operations, need followings.

RTIDB.go	Database.go	Note
Type RTIDatabaseInterface struct {…}	Type LDBDatabase struct {…}	
NewRTIDatabase	NewLDBDatabase	constructor
Put, Get, Has, Delete, …	Put, Get, Has, Delete, …	Basic funcs.
Meter	Meter	stats

Codes should be changed

- node/node.go
- node/service.go
- cmd/geth/chaincmd.go
- eth/database.go
- swarm/storage/ => should add NewDB.go in this
- …



```
func (n *Node) OpenDatabase(name string, cache, handles int) (ethdb.Database, error) {
	if n.config.DataDir == "" {
		return ethdb.NewMemDatabase()
	}

	if ethdb.UseRTI {
		return ethdb.NewRTIDatabase("localhost", "9090", name)
	} else {
		return ethdb.NewLDBDatabase(n.config.resolvePath(name), cache, handles)
	}
}
```

example that modify LDB to RTI(NewDB) at the open database part

Requisites

- thrift version (currently unstable)
- Your own DB (open source or at least provide api)