IBM

Open by design™

IBM开源技术微讲堂

区块链和HyperLedger系列

第四讲

Chaincode实战

http://ibm.biz/opentech-ma

讲师介绍

- 李超
- bjlchao@cn.ibm.com
- @CSL@GCG@IBM

IBM

Open by design™

议程

一、Fabric Chaincode概述

- Chaincode是什么
- Fabric开发支持
- Chaincode运行原理

二、Fabric0.6 Chaincode

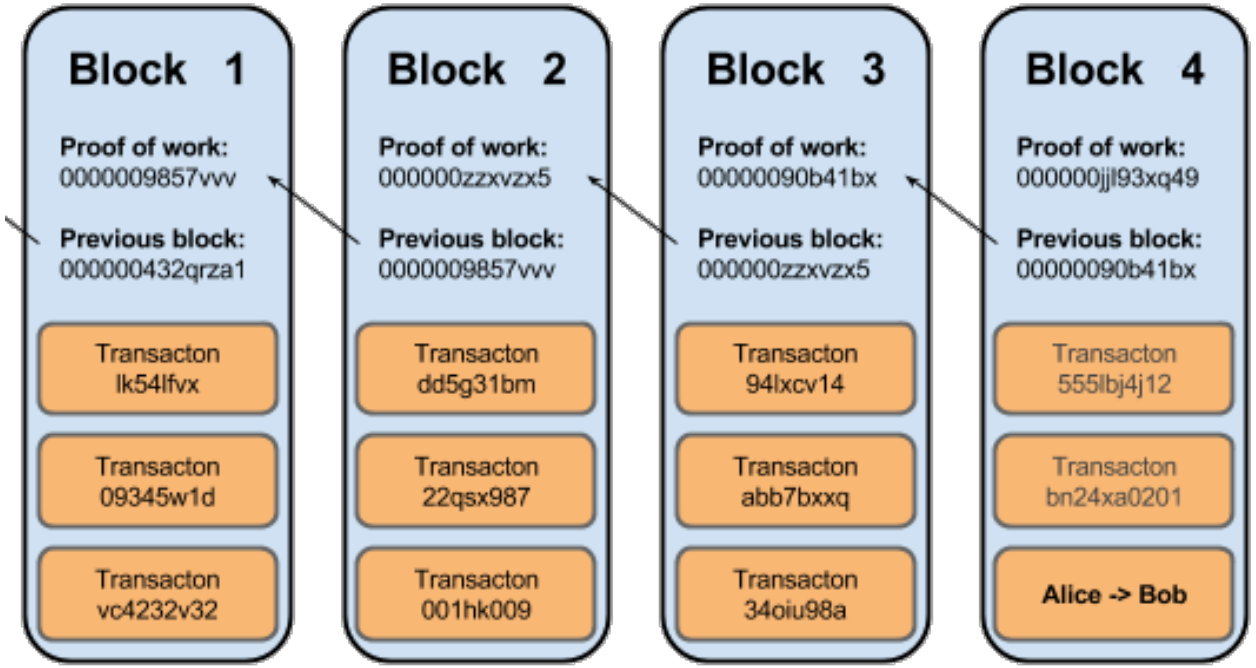
- 相关概念
- 如何编写
- 如何调试

三、Fabric1.0 Chaincode

- 相关概念
- 如何编写
- 如何调试

一、Fabric Chaincode概述 — Chaincode是什么

- 一个接口的实现代码
- 部署在Fabric区块链网络结点上
- 与Fabric区块链交互的唯一渠道
- 生成Transaction的唯一来源
  - Ledger <- Blocks <- Transactions
- 智能合约在Fabric上的实现方式



一、Fabric Chaincode概述 — 开发支持

- 开发语言：

  go、java

- SDK：

  $GOPATH/src/github.com/hyperledger/fabric/core/chaincode/shim

  注：按照Fabric的设计，shim包是供Chaincode开发的SDK，理论上应该是可以独立使用的，但目前并不是如此，它仍然依赖于Fabric的其它某些模块。

一、Fabric Chaincode概述 — 运行原理

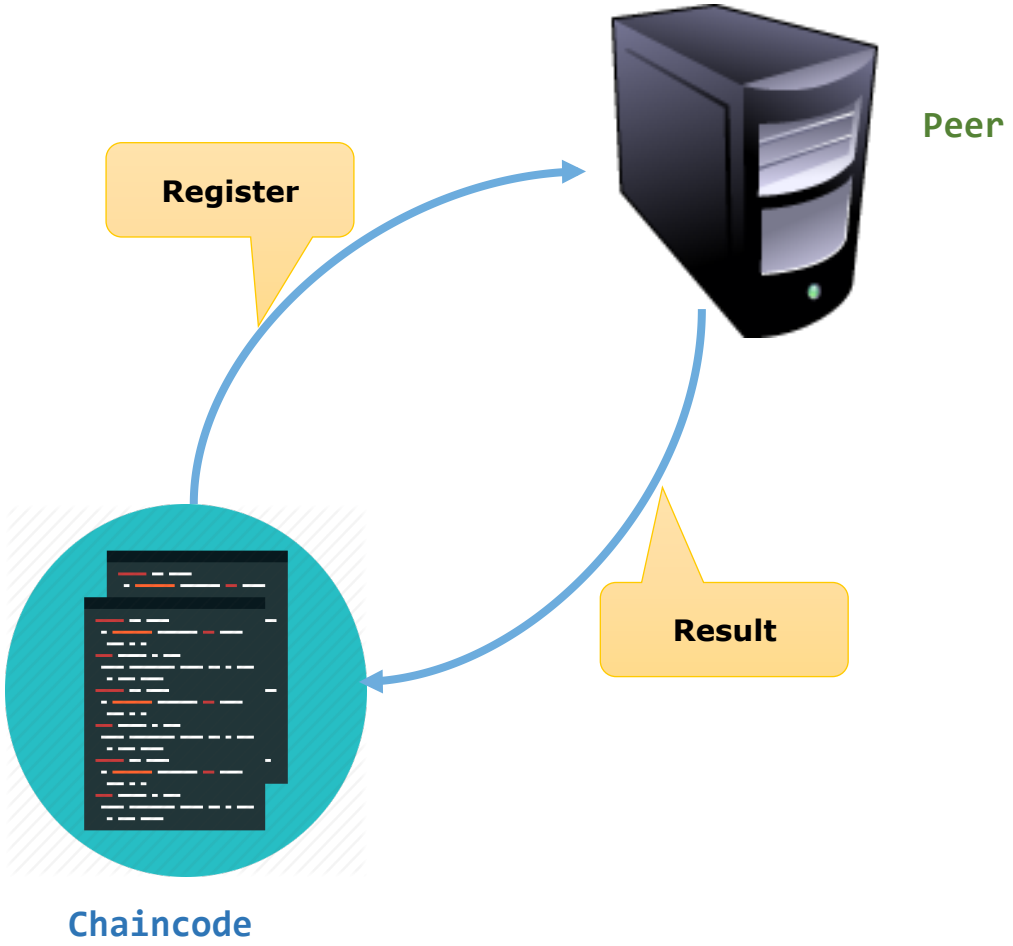
Fabric结点运行模式

- 一般模式
  - Chaincode运行在docker容器中
  - 开发调试过程非常繁杂
    - 部署 -> 调试 -> 修改 -> 创建docker镜像 -> 部署 -> ...
- 开发模式：--peer-chaincodedev
  - Chaincode运行在本地
  - 开发调试相对容易
    - 部署 -> 调试 -> 修改 -> 部署 -> ...

一、Fabric Chaincode概述 — 运行原理

开发模式下

Chaincode注册过程

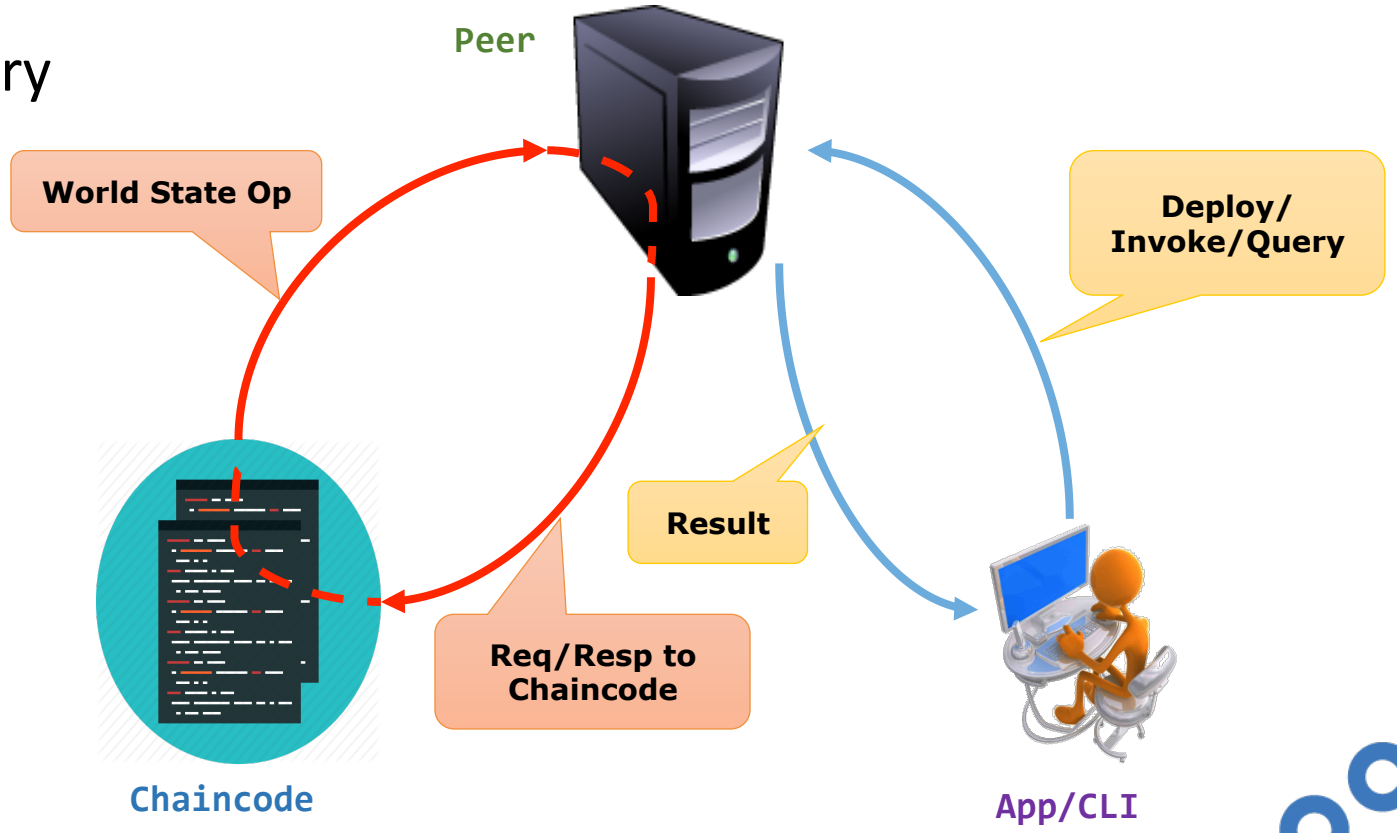


一、Fabric Chaincode概述 — 运行原理

开发模式下

Chaincode Deploy/Invoke/Query

过程



二、Fabric0.6 Chaincode — 相关概念

- Transaction：一次Chaincode函数的运行。
  - 目前有五类，其中一个是Undefined，其余均与chaincode的执行有关。
  - Transaction存储chaincode执行的相关信息，比如chaincodeID、函数名称、参数等，并不包含操作的数据。
- World State：Fabric区块链系统中所有变量的值的集合。
  - Transaction实际操作的是数据，eg. 交易商品的信息。每个chaincode都有自己的数据。
  - Fabric使用Rocksdb存储数据，一个key-value数据库。Key -> 变量，value -> 值。
  - Fabric将每一对key-value叫做一个state，而所有的chaincode的state的合集就是World State。

二、Fabric0.6 Chaincode — 如何编写

必须要实现的接口

```
type Chaincode interface {
        // Deploy transaction中被调用，初始化工作，一般情况下仅被调用一次
        Init(stub ChaincodeStubInterface, function string, args []string) ([]byte, error)

        // Invoke transaction中被调用，更新world state，可多次被调用
        Invoke(stub ChaincodeStubInterface, function string, args []string) ([]byte, error)

        // Query transaction中被调用，查询world state，不能更新world state，可多次被调用
        Query(stub ChaincodeStubInterface, function string, args []string) ([]byte, error)
}
```

二、Fabric0.6 Chaincode — 如何编写

如何实现，eg.

```
package main
import (
        "errors"
        "fmt"
        "github.com/hyperledger/fabric/core/chaincode/shim"
)
type SimpleChaincode struct

func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) { … }

func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) { … }

func (t *SimpleChaincode) Query(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) { … }

func main() {
        err := shim.Start(new(SimpleChaincode))
        if err != nil {
                fmt.Printf("Error starting Simple chaincode: %s", err)
        }
}
```

二、Fabric0.6 Chaincode — 如何编写

shim.ChaincodeStubInterface APIs，五类

- State 操作:

GetState(key string) ([]byte, error)
PutState(key string, value []byte) error
DelState(key string) error
RangeQueryState(startKey, endKey string) (StateRangeQueryIteratorInterface, error)

- Table 操作:

CreateTable(name string, columnDefinitions []*ColumnDefinition) error
GetTable(tableName string) (*Table, error)
DeleteTable(tableName string) error
InsertRow(tableName string, row Row) (bool, error)
ReplaceRow(tableName string, row Row) (bool, error)
GetRow(tableName string, key []Column) (Row, error)
GetRows(tableName string, key []Column) (<-chan Row, error)
DeleteRow(tableName string, key []Column) error

- Chaincode相互调用:

InvokeChaincode(chaincodeName string, args [][]byte) ([]byte, error)
QueryChaincode(chaincodeName string, args [][]byte) ([]byte, error)

二、Fabric0.6 Chaincode — 如何编写

shim.ChaincodeStubInterface APIs，五类

- Transaction操作:

GetArgs() [][]byte

GetStringArgs() []string

GetTxID() string

ReadCertAttribute(attributeName string) ([]byte, error)

GetCallerCertificate() ([]byte, error)

GetCallerMetadata() ([]byte, error)

GetBinding() ([]byte, error)

GetPayload() ([]byte, error)

GetTxTimestamp() (*timestamp.Timestamp, error)

VerifyAttribute(attributeName string, attributeValue []byte) (bool, error)

VerifyAttributes(attrs ...*attr.Attribute) (bool, error)

VerifySignature(certificate, signature, message []byte) (bool, error)

- Event 操作:

SetEvent(name string, payload []byte) error

二、Fabric0.6 Chaincode — 如何编写

其它主要的API还有：

NewLogger(name string) *ChaincodeLogger	获得一个自己的log处理器，name必须唯一
Start(cc Chaincode) error	向peer结点注册自定义的chaincode
SetLoggingLevel(level LoggingLevel)	设置shim的log输出等级

辅助类

StateRangeQueryIterator

ColumnDefinition

Table

Column

二、Fabric0.6 Chaincode — 如何调试

开发调试过程，以MyChaincode为例，完全本地

- 本地启动一个VP结点，开发模式

  peer node start --peer-chaincodedev --logging-level=debug

- 编写自己的Chaincode程序
- 生成可执行程序MyChaincode

  go build

- 运行可执行程序，向VP注册：

  CORE_CHAINCODE_ID_NAME=mycc CORE_PEER_ADDRESS=0.0.0.0:7051 ./MyChaincode

二、Fabric0.6 Chaincode — 如何调试

开发调试过程，以MyChaincode为例，完全本地

- 部署MyChaincode

  peer chaincode deploy -n mycc –c ‘{“Args”:[…]}’

- 提交Invoke transaction

  peer chaincode invoke -n mycc –c ‘{“Args”:[…]}’

- 提交Query transaction

  peer chaincode query -n mycc –c ‘{“Args”:[…]}’

三、Fabric1.0 Chaincode — 相关概念

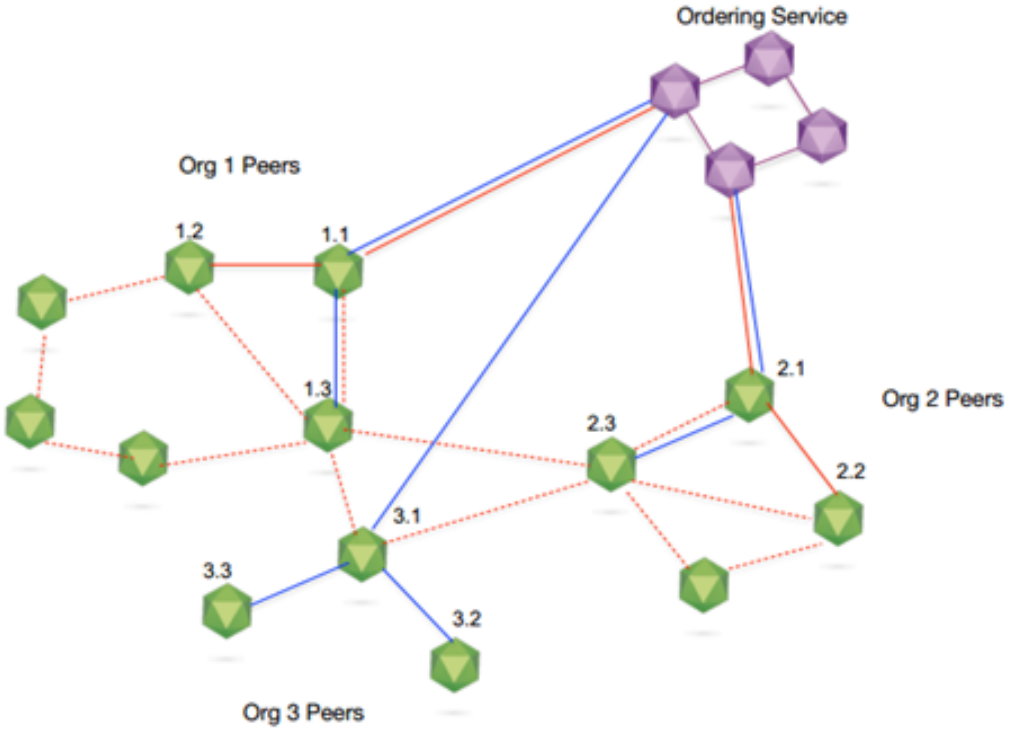
相关概念

Channel — 通道，子链。同一peer可加入不同channel。Chaincode的操作基于channel进行。同一channel上的peer结点同步其上chaincode执行的结果。

Endorser — （模拟）执行Chaincode。分离计算任务，减轻consensus节点负担，增加吞吐量。支持endorsement policy，更加灵活。

Orderer — 对chaincode执行结果consensus。支持solo/kafka/sBFT不同的ordering策略。

Committer — 将chaincode执行结果写进ledger。



三、Fabric1.0 Chaincode — 如何编写

必须要实现的接口

```
type Chaincode interface {
        // 初始化工作，一般情况下仅被调用一次
        Init(stub ChaincodeStubInterface) pb.Response

        // 查询或更新world state，可多次被调用
        Invoke(stub ChaincodeStubInterface) pb.Response
}
```

注：查询操作不会产生transaction。

三、Fabric1.0 Chaincode — 如何编写

如何实现，eg.

```
package main
import (
        "errors"
        "fmt"
        "github.com/hyperledger/fabric/core/chaincode/shim"
)
type SimpleChaincode struct
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) { … }
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) { … }
func main() {
        err := shim.Start(new(SimpleChaincode))
        if err != nil {
                fmt.Printf("Error starting Simple chaincode: %s", err)
        }
}
```

三、Fabric1.0 Chaincode — 如何编写

shim.ChaincodeStubInterface APIs，五类

- State 读写操作

GetState(key string) ([]byte, error)

PutState(key string, value []byte) error

DelState(key string) error

GetStateByRange(startKey, endKey string) (StateQueryIteratorInterface, error)

GetStateByPartialCompositeKey(objectType string, keys []string) (StateQueryIteratorInterface, error)

GetQueryResult(query string) (StateQueryIteratorInterface, error)

GetHistoryForKey(key string) (StateQueryIteratorInterface, error)

CreateCompositeKey(objectType string, attributes []string) (string, error)

SplitCompositeKey(compositeKey string) (string, []string, error)

- Args 读操作

GetArgs() [][]byte

GetStringArgs() []string

GetFunctionAndParameters() (string, []string)

GetArgsSlice() ([]byte, error)

三、Fabric1.0 Chaincode — 如何编写

shim.ChaincodeStubInterface APIs，五类

- Transaction读操作

GetCreator() ([]byte, error)

GetTransient() (map[string][]byte, error)

GetBinding() ([]byte, error)

GetTxTimestamp() (*timestamp.Timestamp, error)

GetTxID() string

- Event 设置

SetEvent(name string, payload []byte) error

- Chaincode相互调用

InvokeChaincode(chaincodeName string, args [][]byte, channel string) pb.Response

三、Fabric1.0 Chaincode — 如何编写

其它主要的API还有：

NewLogger(name string) *ChaincodeLogger　　获得一个自己的log处理器，name必须唯一

Start(cc Chaincode) error　　向peer结点注册自定义的chaincode

SetLoggingLevel(level LoggingLevel)　　设置shim的log输出等级

辅助类

StateRangeQueryIterator　　区间查询

注：API详细说明： https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim

三、Fabric1.0 Chaincode — 如何调试

一般模式下开发调试过程

以fabric chaincode_example02为例，完全本地，使用fabric默认配置

- 本地启动一个Orderer结点，solo模式

  orderer

- 本地启动一个peer结点

  CORE_PEER_ID=peer0 peer node start

- Install Chaincode程序

  peer chaincode install -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02/ -n mycc -v 1.0

三、Fabric1.0 Chaincode — 如何调试

开发调试过程

- 部署Chaincode程序

```
peer chaincode instantiate -n mycc -v 1.0 -c '{"Args": ["init", "A", "100", "B", "100"]}'
```

- 提交Invoke transaction

```
peer chaincode invoke -n mycc -c '{"Args":["invoke", "A", "B", "10"]}‘
```

- 提交Query transaction

```
peer chaincode query -n mycc -c '{"Args":["query", "A"]}'
```

IBM开源技术微讲堂

区块链和HyperLedger系列

第四讲完

http://ibm.biz/opentech-ma