UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

Paralelni i distribuirani algoritmi i strukture podataka

ms Nebojša Horvat

Zimski semestar 2019/2020.

Studijski program: Računarstvo i automatika

Modul: Računarstvo visokih performansi

Hyperledger Fabric

Chaincode

- Pametni ugovor
  - Biznis logika koja se izvršava u Fabric-u
  - Paket shim
  - Chaincode je zadužen za pisanje podataka u ledger, kao i za čitanje istih
  - Može se pisati u više jezika
    - Golang (podrazumevani)
    - NodeJS
    - Java
    - ...

Struktura chaincode-a

- Chaincode ima funkciju koja ga pokreće
  - Main
- Init
- Invoke
- ~~Query~~
  - Više ne postoji, već se i za upite koristi Invoke funkcija

Init

- Funkcija koja se poziva prilikom instanciranja i upgrade-a chaincode-a
- Koristi se da se inicijalizuje World State početnim vrednostima
- Često se kreira posebna funkcija koja inicijalizuje World State, radi lakšeg testiranja

Init

```
import (
	"fmt"
	"strconv"
	"github.com/hyperledger/fabric/core/chaincode/shim"
	pb "github.com/hyperledger/fabric/protos/peer"
)

type SimpleChaincode struct {
}

func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {

	return shim.Success(nil)
}
```

Invoke-old ways

```
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
        function, args := stub.GetFunctionAndParameters()
        if function == "invoke" {
        // Prebaci x sredstava sa A na B
        return t.invoke(stub, args)
        } else if function == "delete" {
        //Obriši račun
        return t.delete(stub, args)
        } else if function == "query" {
        // Upit
        return t.query(stub, args)
        }

        return shim.Error(“Funkcija ne postoji”)
}
peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n mycc --
peerAddresses peer0.org1.example.com:7051 --peerAddresses
peer0.org2.example.com:7051 -c '{"Args":["invoke","a","b","10"]}'
```

Invoke

```
func (t *SimpleChaincode) invoke(stub shim.ChaincodeStubInterface, args []string) pb.Response {
        var A, B string    // Identifikatori računa
                var Aval, Bval int // Količina sredstava na računima
                var X int          // Količina sredstava za prebacivanje
                var err error
                if len(args) != 3 {
                        return shim.Error("Očekuju se 3 argumenta")
                }
                A = args[0]
                B = args[1]
                Avalbytes, err := stub.GetState(A)
                if err != nil {
                        return shim.Error("Nema računa")
                }
                .....
                Aval = Aval - X
                Bval = Bval + X
                        err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
                if err != nil {
                        return shim.Error(err.Error())
                }
                err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
                if err != nil {
                        return shim.Error(err.Error())
                }

                return shim.Success(nil)
        }
```

Invoke – Primer upita

```
func (t *SimpleChaincode) getHistoryForKey(stub
shim.ChaincodeStubInterface, args []string) pb.Response {


    id:= args[0]


    resultsIterator, err := stub.GetHistoryForKey(id)
    if err != nil {
    return shim.Error(err.Error())
    }
       defer resultsIterator.Close()
....
...
```

Shim paket

- Shim paket se koristi za rad sa ledger-om
  - Pisanje podataka u ledger
  - Čitanje podataka

SHIM paket

- GetState(key) - Vraća vrednost svojine iz Fabric World State-a
- PutState(key,value) - Postavi vrednost svojine u Fabric World State
- GetHistoryForKey(key) - Upit koji vraća sve transakcije nad odgovarajućom svojinom
- DelState(key) // Stavlja “deleted” flag u World State za odgovarajuću svojinu
- GetQueryResult(query) //Vraća iterator na rezultat pretrage specificiran query stringom
- GetStateByRange(from,to) //Vraća iterator na rezultat pretrage specificiran opsegom ključceva (from, to)
- Kompletna dokumentacija
  - https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim

JSON kao vrednost u World State-u

```
func (tx *Transaction) FromJson(txJson []byte) (*Transaction, error) {

    error:= json.Unmarshal(txJson, &tx)

    if error != nil {
        return nil, errors.New(err.Error())
    }

    return tx, nil
}

func (tx *Transaction) ToJson() ([]byte, error) {

    txJson, err or:= json.Marshal(tx)

    if error != nil {
        logger.Error("Error while marshaling")
        return nil, errors.New(err.Error())
    }

    return txJson, nil
}
```

Instaliranje chaincode-a

Lifecycle system chaincode (LSCC) mora biti iskorišćen da bi se instalirao chaincode na peer-ovima

- peer chaincode install -n mycc -v 0 –p path/to/my/chaincode/v0

Instanciranje i upgrade chaincode-a

Takođe, podrazumeva pokretanje sistemskog Chaincode-a

- peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile $ORDERER_CA -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "AND ('Org1MSP.peer','Org2MSP.peer')"
- peer chaincode upgrade -o orderer.example.com:7050 --tls --cafile $ORDERER_CA -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200"]}'

Zadatak

- Izmeniti postojeći chaincode iz asset-transfer-basic foldera, tako da prihvata dodatni argument koji ograničava prebacivanje vlasništva nad asset-om. Ukoliko je dodatni int argument manji od size asset-a, nije moguće odraditi transfer.

Zadatak

- Dodati strukturi Asset dodatno polje koje sadrzi listu svih korisnika tog Asset-a. Korisnik asset-a je nova struktura koju treba napraviti sa proizvoljnim poljima. Implementirati i propratne funkcije koje dozvoljavaju dodavanje/brisanje korisnika iz asset-a.

NodeJS chaincode- primer

```
class SecondaryMarket {


    async Init(stub) {
    }
    async Invoke(stub) {
        let functionAndParams = stub.getFunctionAndParameters();
        let functionName = functionAndParams.fcn;
        let functionParams = functionAndParams.params;


        logger.debug("Invoking " + functionName + "with args: " + functionParams);
        ....
        let fcn = this[functionName];
        return await fcn.call(this, stub, functionParams);
    }
    async createTx(stub, args) {
        let something= args[0]     ...
         // some validation

        let a= args[1];
        let receivedVal = stub.getState(a)
        //use receivedVal
        let val= JSON.parse(something)
        //change val
        await stub.putState(txId, Buffer.from(val));
        return shim.success(Buffer.from("Transaction saved !"));
    }
```