

**综合性、设计性实验报告**

姓名\_周鑫文\_\_\_\_\_\_ 学号\_201606050317

专业 软件工程 班级2017级 软件3 班

实验课程名称 前沿软件讲座

指导教师及职称 黎明 副高

开课学期 2019 至 2020 学年 第一学期

上课时间 第5次至第6次实验

湖南科技学院教务处编

1. **实验设计方案**

|  |  |  |
| --- | --- | --- |
| 实验名称： HyperLedgerFabric链码开发实验 | | 实验时间：第5次至第6次实验 |
| 小组合作： 是○ 否○● | 成绩： | 教师评语： |
| 1、实验目的：  (1)、掌握Fabric的基本概念。  (2)、掌握Fabric的设计方法。  (3)、掌握Fabric网络设计方法。  (4)、掌握Fabric启动和链码安装，实例和调用方法 | | |
| 2、实验设备及材料：  MAC一体机，操作系统 MacOSX，开发平台 Remix或者ethreum平台 | | |
| 实验内容：  实现以下功能并完成测试。  1. 创建一个弹珠，弹珠字段包括弹珠ID、颜色、尺寸和拥有者；  2. 通过弹珠ID查询对应的弹珠信息；  3. 通过弹珠ID删除对应的弹珠；  4. 改变弹珠的拥有者；  5. 查询指定ID范围的弹珠信息；  6. 查询某个拥有者的所有弹珠信息；  7. 通过弹珠ID查询所有的交易历史信息。 | | |
| 4、实验方法步骤：  (1) 分析解题思路  完成弹珠的基本设计，确定弹珠的功能清单，确定弹珠的设计，完成弹珠代码设计，测试弹珠代码，部署，实例化弹珠，调用函数，创建，修改，查询弹珠等信息，查询状态数据库，实现功能，测试代码。   1. 编写源程序   创建结构体  type MarblesChaincode struct {  }  type Marble struct {  ObjectType string `json:"objectType"` // CouchDB的字段  ID string `json:"id"` // ID（唯一字符串，将用作键）  Color string `json:"color"` // 颜色（字符串，CSS颜色名称）  Size int `json:"size"` // 尺寸（整型，以毫米为单位）  Owner string `json:"owner"` // 拥有者（字符串）  }  实现Init函数和main函数  func (t \*MarblesChaincode) Init (stub shim.ChaincodeStubInterface) pb.Response {  fmt.Println("Marbles Init Success")  return shim.Success(nil)  }  func (t \*MarblesChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {  fmt.Println("Marbles Invoke")  // ...  return shim.Error("没找到对应方法~")  }  func main() {  err := shim.Start(new(MarblesChaincode))  if err != nil {  fmt.Printf("Error starting Marbles Chaincode - %s", err)  }  }  初始化弹珠  func (t \*MarblesChaincode) initMarble (stub shim.ChaincodeStubInterface, args []string) pb.Response {  if len(args) != 4 {  return shim.Error("Incorrect number of arguments. Expecting 4")  }  marbleObjectType := "marble"  marbleId := args[0]  marbleIdAsBytes, err := stub.GetState(marbleId)  if err != nil {  return shim.Error(err.Error())  }  if marbleIdAsBytes != nil {  return shim.Error("该Marble已存在~")  }  marbleColor := args[1]  marbleSize, err := strconv.Atoi(args[2])  if err != nil {  return shim.Error("size字段为整型~")  }  marbleOwner := args[3]  marble := &Marble{marbleObjectType, marbleId, marbleColor, marbleSize, marbleOwner}  marbleJsonAsBytes, err := json.Marshal(marble)  err = stub.PutState(marbleId, marbleJsonAsBytes)  if err != nil {  return shim.Error(err.Error())  }  return shim.Success(nil)  }  通过弹珠ID查询对应的弹珠信息  func (t \*MarblesChaincode) getMarbleInfoByID (stub shim.ChaincodeStubInterface, args []string) pb.Response {  if len(args) != 1 {  return shim.Error("Incorrect number of arguments. Expecting 1")  }  marbleId := args[0]  // 查询账本中是否已经存在该弹珠ID的信息  marbleIdAsBytes, err := stub.GetState(marbleId)  if err != nil {  return shim.Error(err.Error())  } else if marbleIdAsBytes == nil {  return shim.Error("该Marble不存在~")  }  return shim.Success(marbleIdAsBytes)  }  通过弹珠ID删除对应的弹珠  func (t \*MarblesChaincode) deleteMarbleByID (stub shim.ChaincodeStubInterface, args []string) pb.Response {  if len(args) != 1 {  return shim.Error("Incorrect number of arguments. Expecting 1")  }  marbleId := args[0]  // 查询账本中是否已经存在该弹珠ID的信息  marbleIdAsBytes, err := stub.GetState(marbleId)  if err != nil {  return shim.Error(err.Error())  } else if marbleIdAsBytes == nil {  return shim.Error("该Marble不存在~")  }  // 删除账本中的记录  err = stub.DelState(marbleId)  if err != nil {  return shim.Error(err.Error())  }  return shim.Success(nil)  }  改变弹珠的拥有者  func (t \*MarblesChaincode) changeMarbleOwner (stub shim.ChaincodeStubInterface, args []string) pb.Response {  if len(args) != 2 {  return shim.Error("Incorrect number of arguments. Expecting 2")  }  marbleId := args[0]  newOwner := args[1]  // 查询账本中是否已经存在该弹珠ID的信息  marbleIdAsBytes, err := stub.GetState(marbleId)  if err != nil {  return shim.Error(err.Error())  } else if marbleIdAsBytes == nil {  return shim.Error("该Marble不存在~")  }  marble := Marble{}  err = json.Unmarshal(marbleIdAsBytes, &marble)  if err != nil {  return shim.Error(err.Error())  }  marble.Owner = newOwner  marbleJsonAsBytes, err := json.Marshal(marble)  if err != nil {  return shim.Error(err.Error())  }  // 写入到账本中  err = stub.PutState(marbleId, marbleJsonAsBytes)  if err != nil {  return shim.Error(err.Error())  }  return shim.Success(nil)  }  查询指定ID范围的弹珠信息  func (t \*MarblesChaincode) getMarbleByRange (stub shim.ChaincodeStubInterface, args []string) pb.Response {  if len(args) != 2 {  return shim.Error("Incorrect number of arguments. Expecting 2")  }  startKey := args[0]  endKey := args[1]  // 查询指定范围内的键值  resultsIterator, err := stub.GetStateByRange(startKey, endKey)  if err != nil {  return shim.Error(err.Error())  }  defer resultsIterator.Close()  var buffer bytes.Buffer  buffer.WriteString("[")  bArrayMemberAlreadyWritten := false  // 遍历弹珠信息，转换为Json格式的字符串数组并返回  for resultsIterator.HasNext() {  queryResult, err := resultsIterator.Next()  if err != nil {  return shim.Error(err.Error())  }  if bArrayMemberAlreadyWritten == true {  buffer.WriteString(",")  }  buffer.WriteString("{\"Key\":")  buffer.WriteString("\"")  buffer.WriteString(queryResult.Key)  buffer.WriteString("\"")  buffer.WriteString(", \"Record\":")  // Record is a JSON object, so we write as-is  buffer.WriteString(string(queryResult.Value))  buffer.WriteString("}")  bArrayMemberAlreadyWritten = true  }  buffer.WriteString("]")  fmt.Printf("- getMarblesByRange queryResult:\n%s\n", buffer.String())  return shim.Success(buffer.Bytes())  }  查询某个拥有者的所有弹珠信息  func (t \*MarblesChaincode) getMarbleByOwner(stub shim.ChaincodeStubInterface, args []string) pb.Response {  marbleOwner := args[0]  queryStr := fmt.Sprintf("{\"selector\":{\"owner\":\"%s\"}}",marbleOwner)  resultsIterator, err := stub.GetQueryResult(queryStr)  if err != nil {  return shim.Error(err.Error())  }  defer resultsIterator.Close()  var buffer bytes.Buffer  buffer.WriteString("[")  bArrayMemberAlreadyWritten := false  for resultsIterator.HasNext() {  queryResult, err := resultsIterator.Next()  if err != nil {  return shim.Error(err.Error())  }  if bArrayMemberAlreadyWritten == true {  buffer.WriteString(",")  }  buffer.WriteString("{\"Key\":")  buffer.WriteString("\"")  buffer.WriteString(queryResult.Key)  buffer.WriteString("\"")  buffer.WriteString(", \"Record\":")  buffer.WriteString(string(queryResult.Value))  buffer.WriteString("}")  bArrayMemberAlreadyWritten = true  }  buffer.WriteString("]")  fmt.Printf("- getMarbleByOwner queryResult:\n%s\n", buffer.String())  return shim.Success(buffer.Bytes())  }  通过弹珠ID查询所有的交易历史信息  func (t \*MarblesChaincode) getMarbleHistoryForKey (stub shim.ChaincodeStubInterface, args []string) pb.Response {  if len(args) != 1 {  return shim.Error("Incorrect number of arguments. Expecting 1")  }  marbleId := args[0]  // 返回某个键的所有历史值  resultsIterator, err := stub.GetHistoryForKey(marbleId)  if err != nil {  return shim.Error(err.Error())  }  defer resultsIterator.Close()  var buffer bytes.Buffer  buffer.WriteString("[")  bArrayMemberAlreadyWritten := false  for resultsIterator.HasNext() {  queryResult, err := resultsIterator.Next()  if err != nil {  return shim.Error(err.Error())  }  if bArrayMemberAlreadyWritten == true {  buffer.WriteString(",")  }  buffer.WriteString("{\"TxId\":")  buffer.WriteString("\"")  buffer.WriteString(queryResult.TxId)  buffer.WriteString("\"")  buffer.WriteString(", \"Timestamp\":")  buffer.WriteString("\"")  buffer.WriteString(time.Unix(queryResult.Timestamp.Seconds, int64(queryResult.Timestamp.Nanos)).String())  buffer.WriteString("\"")  buffer.WriteString("{\"Value\":")  buffer.WriteString("\"")  buffer.WriteString(string(queryResult.Value))  buffer.WriteString("\"")  buffer.WriteString("{\"IsDelete\":")  buffer.WriteString("\"")  buffer.WriteString(strconv.FormatBool(queryResult.IsDelete))  buffer.WriteString("\"")  bArrayMemberAlreadyWritten = true  }  buffer.WriteString("]")  fmt.Printf("- getMarblesByRange queryResult:\n%s\n", buffer.String())  return shim.Success(buffer.Bytes())  }  实现Invoke函数  func (t \*MarblesChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {  fmt.Println("Marbles Invoke")  function, args := stub.GetFunctionAndParameters()  if function == "initMarble" {  return t.initMarble(stub, args)  } else if function == "getMarbleInfoByID" {  return t.getMarbleInfoByID(stub, args)  } else if function == "deleteMarbleByID" {  return t.deleteMarbleByID(stub, args)  } else if function == "changeMarbleOwner" {  return t.changeMarbleOwner(stub, args)  } else if function == "getMarbleByRange" {  return t.getMarbleByRange(stub, args)  } else if function == "getMarbleByOwner" {  return t.getMarbleByOwner(stub, args)  } else if function == "getMarbleHistoryForKey" {  return t.getMarbleHistoryForKey(stub, args)  }  return shim.Error("没找到对应方法~")  }  services:  peer:  environment:  ...  - CORE\_PEER\_MSPCONFIGPATH=/etc/hyperledger/msp  ###以下是添加的内容  - CORE\_PEER\_MSPCONFIGPATH=/etc/hyperledger/msp  - CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB  - CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=couchdb:5984  - CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_USERNAME=  - CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_PASSWORD=    depends\_on:  - orderer  ###以下是添加的内容  - couchdb    couchdb:  container\_name: couchdb  image: hyperledger/fabric-couchdb  # Populate the COUCHDB\_USER and COUCHDB\_PASSWORD to set an admin user and password  # for CouchDB. This will prevent CouchDB from operating in an "Admin Party" mode.  environment:  - COUCHDB\_USER=  - COUCHDB\_PASSWORD=  ports:  - 5984:5984  富查询配置CouchDB   1. 结果截图          1. 结果分析   **额，弹珠设置成功。** | | |