006 弹珠资产管理 chaincode 开发

> 孔壹学院：国内区块链职业教育领先品牌
> 官方网址：http://www.kongyixueyuan.com/

主要开发功能

- 创建一个弹珠信息
- 从账本中读取一个弹珠信息
- 删除一个弹珠信息
- 更改一个弹珠的拥有者
- 返回所有名称在指定字典范围内的弹珠信息
- 返回指定拥有者拥有的所有的弹珠的信息
- 返回一个弹珠的所有历史信息

chaincode 代码实现

创建一个弹珠信息

创建弹珠的主要操作：

- 根据marbleName查询弹珠是否被创建
- 如果没有创建，将marble对象转为json对象，将marble写入到账本中

```
func(t *MarblesChaincode) initMarble(stub shim.ChaincodeStubInterface, args
[]string) peer.Response {
    fmt.Println("initMarble start")
    marbleName := args[0]
    color := args[1]
    size, err := strconv.Atoi(args[2])
    if err != nil {
        return shim.Error("size 必须是数字")
    }
    owner := args[3]

    // 首先判断 marbleName 是不是已经存在
    marbleAsBytes , err := stub.GetState(marbleName)
```

```
    if err != nil {
        return shim.Error ("获取marble失败: "+err.Error())
    }else if marbleAsBytes != nil {
        fmt.Printf("%s marble 已经存在! ",marbleName)
        return shim.Error("marble 已经存在! ")
    }


    objectType := "marble"
    marble := &marble{objectType,marbleName,color,size,owner}

    marbleJsonAsBytes,err := json.Marshal(marble)
    if err != nil {
        return shim.Error(err.Error())
    }

    fmt.Printf("marbleJsonAsBytes %v: \n",string(marbleJsonAsBytes))

    err = stub.PutState(marbleName,marbleJsonAsBytes)

    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(nil)
}
```

从账本中读取一个弹珠信息

查询弹珠信息主要操作:

- 通过 GetState 获取相应的弹珠信息

```
func (t *MarblesChaincode) readMarble (stub shim.ChaincodeStubInterface, args []string) peer.Response {
    fmt.Println("readMarble start")

    var name, jsonResp string

    name = args[0]

    marbleAsBytes,err := stub.GetState(name)

    if err != nil {
        jsonResp = "{\"error\": \"获取数据失败! \"}"
        return shim.Error(jsonResp)
```

```
    } else if marbleAsBytes == nil {
        jsonResp = "{\"error\": \"marble 不存在! \"}"
        return shim.Error(jsonResp)
    }

    return shim.Success(marbleAsBytes)

}
```

删除一个弹珠信息

删除一个弹珠信息主要操作:

- 查询弹珠是否存在
- 根据 marbleName 删除相应弹珠

```
func (t *MarblesChaincode) deleteMarble (stub shim.ChaincodeStubInterface, a
rgs []string) peer.Response {

    fmt.Println("deleteMarble start")
    var marblename string

    marblename = args[0]

    marbleAsBytes , err := stub.GetState(marblename)

    if err != nil {
        return shim.Error("获取marble信息失败! "+err.Error())
    } else if marbleAsBytes == nil {
        return shim.Error("marble 不存在! ")
    }

    err = stub.DelState(marblename)

    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(nil)
}
```

更改一个弹珠的拥有者

主要操作:

- 根据 marbleName 查询 marble详情
- 修改 marble的拥有者
- 写入到账本中

```
func (t *MarblesChaincode) transferMarble (stub shim.ChaincodeStubInterface,
 args []string) peer.Response {
    marbleName := args[0]
    newOwner := args[1]
    fmt.Println("start transferMarble ", marbleName, newOwner)

    marbleAsBytes, err := stub.GetState(marbleName)

    if err != nil {
        return shim.Error("获取marble信息失败！")
    } else if marbleAsBytes == nil {
        return shim.Error("marble 不存在！")
    }

    marbleToTransfer := marble{}

    err = json.Unmarshal(marbleAsBytes,&marbleToTransfer)

    if err != nil {
        return shim.Error(err.Error())
    }

    marbleToTransfer.Owner = newOwner

    marbleJsonToTransfer , err := json.Marshal(marbleToTransfer)
    err = stub.PutState(marbleName,marbleJsonToTransfer)
    if err != nil {
        return shim.Error(err.Error())
    }

    fmt.Println("end transferMarble ")

    return shim.Success(nil)
}
```

返回所有名称在指定字典范围内的弹珠信息

- 通过起始key和终止key 获取到指定弹珠信息
- 通过迭代将弹珠信息转为JSON数组

```
func (t *MarblesChaincode) getMarblesByRange( stub shim.ChaincodeStubInterfa
ce, args []string) peer.Response{

    fmt.Println("start getMarblesByRange")

    startKey := args[0]
    endKey := args[1]

    resultIterator, err  := stub.GetStateByRange(startKey,endKey)
    if err != nil {
        return shim.Error(err.Error())
    }

    defer resultIterator.Close()

    //迭代resultIterator
    var buffer bytes.Buffer
    buffer.WriteString("[")

    isWrited := false
    for resultIterator.HasNext() {
        queryResponse, err := resultIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }

        if isWrited == true {
            buffer.WriteString(",")
        }
        buffer.WriteString("{\"key\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResponse.Key)
        buffer.WriteString("\",\"record\":")

        buffer.WriteString(string(queryResponse.Value))
        buffer.WriteString("}")
    }
    buffer.WriteString("]")
    fmt.Printf("getMarblesByRange result: \n%s\n",buffer.String())
    fmt.Println("end getMarblesByRange")
    return shim.Success(buffer.Bytes())
}
```

运行结果：

```
[{"key":"marble2","record":{"docType":"marble","name":"marble2","color":"red
```

```
","size":50,"owner":"jerry"}}{"key":"marble3","record":{"docType":"marble","
name":"marble3","color":"blue","size":70,"owner":"tom"}}]
```

返回指定拥有者拥有的所有的大理石的信息

该查询为富查询，需要支持富查询的数据库（如CouchDB）

```
func (t *MarblesChaincode) queryMarblesByOwner(stub shim.ChaincodeStubInterf
ace,args []string) peer.Response{

    fmt.Println("start queryMarblesByOwner")
    owner := args[0]

    queryString := fmt.Sprintf("{\"selector\":{\"docType\":\"marble\",\"owne
r\":\"%s\"}}",owner)
    fmt.Println(queryString)
    resultIterator, err := stub.GetQueryResult(queryString)
    if err != nil {
        return shim.Error(err.Error())
    }

    defer resultIterator.Close()

    //迭代resultIterator
    var buffer bytes.Buffer
    buffer.WriteString("[")

    isWrited := false
    for resultIterator.HasNext() {
        queryResponse, err := resultIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }

        if isWrited == true {
            buffer.WriteString(",")
        }
        buffer.WriteString("{\"key\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResponse.Key)
        buffer.WriteString("\",\"record\":")

        buffer.WriteString(string(queryResponse.Value))
        buffer.WriteString("}")
    }
    buffer.WriteString("]")
```

```
    fmt.Printf("queryMarblesByOwner result: \n%s\n",buffer.String())
    fmt.Println("end queryMarblesByOwner")
    return shim.Success(buffer.Bytes())

}
```

返回一个大理石的所有历史信息

- GetHistoryForKey 获取相应的历史信息

```
func (t *MarblesChaincode) getHistoryForMarble(stub shim.ChaincodeStubInterf
ace,args []string) peer.Response{

    fmt.Println("start getHistoryForMarble")

    marbleName := args[0]
    fmt.Println("marbleName: ",marbleName)
    resultIterator,err := stub.GetHistoryForKey(marbleName)
    if err != nil {
        return shim.Error(err.Error())
    }

    defer resultIterator.Close()
    //迭代resultIterator
    var buffer bytes.Buffer
    buffer.WriteString("[")

    isWrited := false
    for resultIterator.HasNext() {
        queryResponse, err := resultIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }

        if isWrited == true {
            buffer.WriteString(",")
        }

        buffer.WriteString("{\"TxId\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResponse.TxId)
        buffer.WriteString("\"")

        buffer.WriteString(",\"Value\":")
        if queryResponse.IsDelete {
            buffer.WriteString("null")
```

```
        }else{
            buffer.WriteString(string(queryResponse.Value))
        }

        buffer.WriteString(",\"Timestamp\": ")
        buffer.WriteString("\"")
        buffer.WriteString(time.Unix(queryResponse.Timestamp.Seconds,int64(q
ueryResponse.Timestamp.Nanos)).String())
        buffer.WriteString("\"")

        buffer.WriteString(",\"isDelete\":")
        buffer.WriteString("\"")
        buffer.WriteString(strconv.FormatBool(queryResponse.IsDelete))
        buffer.WriteString("\"")

        buffer.WriteString("}")
        isWrited = true
    }
    buffer.WriteString("]")

    fmt.Printf("getHistoryForMarble result \n%s\n",buffer.String())
    fmt.Println("end getHistoryForMarble")

    return shim.Success(buffer.Bytes())
}
```