Hyperledger Fabric 链码开发实战

1、本篇背景

这里假设您已经基本掌握了链码开发及shim包API的相关知识，这里以一个简单的应用场景为例，进行链码的开发。

假设需要用链码开发简单的员工管理应用，要实现以下几个简单的业务需求：
1、可以添加一个部门，部门字段包括部门ID和部门名称；
2、可以为某个部门添加员工，员工字段包括员工ID、员工姓名、员工所属部门、工作岗位；
3、可以根据员工ID进行查询、修改和删除等操作。

2、链码开发

了解了上面的业务需求后，我们直接进入链码的开发。

先创建一个用于保存本次实验的项目文件夹，这里命名为"my_chaincode01"，在该目录下创建一个与文件夹同名的后缀名为".go"链码文件。

2.1 创建结构体

先创建"部门"、"员工"以及智能合约的结构体

```
// 定义智能合约结构体
type SmartContract struct {
}

/*
 * 定义"部门"结构体
 * 部门字段包括部门ID和部门名称
 */
type Department struct {
  DepartmentID int `json:department_id` // 部门ID
  DepartmentName string `json:department_name`  // 部门名称
}

/*
 * 定义"员工"结构结构体
 * 员工字段包括员工ID、员工姓名、员工所属部门ID、工作岗位
 */
 type Employee struct {
   EmployeeID int `json:employee_id`  // 员工ID
   EmployeeName string `json:employee_name` // 员工姓名
```

```
    DepartmentID int `json:department_id` // 部门ID
    Jobs string `json:jobs`  // 工作岗位
 }
```

2.2 实现Init函数和main函数

这里先使用实现Init函数和main函数，Invoke函数和其他跟业务有关的功能后面一起实现。

```
 // 在链码初始化过程中调用Init来初始化任何数据
 func (t *SmartContract) Init(stub shim.ChaincodeStubInterface) pb.Response
{
    fmt.Println("my_chaincode01 Init")
    return shim.Success(nil)
 }

 // ...后面实现Invoke函数和其他功能
 func (t *SmartContract) Invoke(stub shim.ChaincodeStubInterface) pb.Respons
e {
    // ...
 }

 // Go语言的入口是main函数
 func main() {
   err := shim.Start(new(SmartContract))
   if err != nil {
     fmt.Printf("Error creating new Smart Contract: %", err)
   }
 }
```

2.3 初始化部门

功能是创建一个部门，当然还可以实现删改查等功能，这里暂不处理这些功能。

```
  // 初始化部门
  func (t *SmartContract) initDepartment(stub shim.ChaincodeStubInterface, a
rgs []string) pb.Response {
    if len(args) != 2 {
      return shim.Error("Incorrect number of arguments. Expecting 2 like (de
partmentId, departmentName)")
    }

    departmentIdAsString := args[0]
    // 转换为int类型
    departmentIdAsInt, err := strconv.Atoi(args[0])
    if err != nil {
      return shim.Error("first argument must be a numeric string")
```

```
    }
    departmentName := args[1]
    // 创建"部门"结构体
    department := &Department{departmentIdAsInt, departmentName}

    // 创建联合主键，用多个列组合作为联合主键
    // Fabric是用U+0000来把各个联合主键的字段拼接起来，因为这个字符太特殊，所以很适合，
    departmentIdKey, err := stub.CreateCompositeKey("Department", []string{"
department", departmentIdAsString})
    if err != nil {
           return shim.Error(err.Error())
      }

    // 结构体转换为json字符串
    departmentAsBytes, err := json.Marshal(department)
    if err != nil {
      return shim.Error(err.Error())
    }

    // 新增一条"部门"数据
    err = stub.PutState(departmentIdKey, departmentAsBytes)
    if err != nil {
            return shim.Error(err.Error())
      }
    return shim.Success(departmentAsBytes)
 }
```

2.4 新增员工

因为员工率属于部门，所以，新增员工的时候，需要判断添加的部门是否已经存在

```
 // 新增员工
 func (t *SmartContract) addEmployee(stub shim.ChaincodeStubInterface, args
[]string) pb.Response {
   // 将字符串数组类型数据转换为"员工"结构体
   employee, err := translateEmployeeFromArgs(args)
     if err != nil {
        return shim.Error(err.Error())
     }
   fmt.Println("employee:", employee)
     employeeIdAsString := strconv.Itoa(employee.EmployeeID)
   // 检查添加的部门ID是否已经存在，返回所有的部门ID
   departmentIds := queryAllDepartmentIDs(stub)
   fmt.Println("departmentIds:", departmentIds)

   // 是否已经存在该员工
   isExist := false
```

```
    if len(departmentIds) > 0 {
      for _, departmentId := range departmentIds {
        // 转换为int类型
        departmentIdAsInt, err := strconv.Atoi(departmentId)
        if err != nil {
              return shim.Error("Department Id argument must be a numeric str
ing")
          }

        if departmentIdAsInt == employee.DepartmentID {
          isExist = true
          break
        }
      }
    }

    if isExist {
      // 读取账本中的数据
      employeeAsBytes, err := stub.GetState(employeeIdAsString)
      if err != nil {
        return shim.Error(err.Error())
      } else if employeeAsBytes != nil {
        fmt.Println("This employee already exists: " + employeeIdAsString)
        return shim.Error("This employee already exists: " + employeeIdAsStri
ng)
      }

      // 结构体转换为json字符串
      employeeAsJsonBytes, err := json.Marshal(employee)
      if err != nil {
        return shim.Error(err.Error())
      }
      // 保存到账本中
      err = stub.PutState(employeeIdAsString, employeeAsJsonBytes)
      if err != nil {
        return shim.Error(err.Error())
      }
      return shim.Success(employeeAsJsonBytes)
    } else {
      fmt.Println("department:" + string(employee.DepartmentID) + " does not
exist")
          return shim.Error("department:" + string(employee.DepartmentID) +
" does not exist")
    }
  }
```

其中涉及到将字符串数组转换为"员工"结构体的功能：

```
// 将字符串数组类型数据转换为"员工"结构体
func translateEmployeeFromArgs(args []string) (*Employee, error) {
  if len(args) != 4 {
    return nil, errors.New("Incorrect number of arguments. Expecting 4 like
(employeeId, employeeName, departmentId, jobs)")
  }

  // 转换为int类型
  employeeId, err := strconv.Atoi(args[0])
  if err != nil {
        return nil, errors.New("first argument must be a numeric string")
    }
  employeeName := args[1]
  departmentId, err := strconv.Atoi(args[2])
  if err != nil {
        return nil, errors.New("third argument must be a numeric string")
    }
  jobs := args[3]

  employee := &Employee{employeeId, employeeName, departmentId, jobs}
  return employee, nil
}
```

而且，涉及到查询所有部门ID的功能：

```
// 将字符串数组类型数据转换为"员工"结构体
func translateEmployeeFromArgs(args []string) (*Employee, error) {
  if len(args) != 4 {
    return nil, errors.New("Incorrect number of arguments. Expecting 4 like
(employeeId, employeeName, departmentId, jobs)")
  }

  // 转换为int类型
  employeeId, err := strconv.Atoi(args[0])
  if err != nil {
        return nil, errors.New("first argument must be a numeric string")
    }
  employeeName := args[1]
  departmentId, err := strconv.Atoi(args[2])
  if err != nil {
        return nil, errors.New("third argument must be a numeric string")
    }
  jobs := args[3]

  employee := &Employee{employeeId, employeeName, departmentId, jobs}
  return employee, nil
```

```
}
```

2.5 删除员工

删除员工和查询员工都比较简单：

```
// 删除员工
func (t *SmartContract) deleteEmployee(stub shim.ChaincodeStubInterface, ar
gs []string) pb.Response {
   if len(args) < 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1 like (
employee_id)")
    }
   employeeIdAsString := args[0]
   employeeAsBytes, err := stub.GetState(employeeIdAsString)
   if err != nil {
     return shim.Error("Failed to get employee info:" + err.Error())
   } else if employeeAsBytes == nil {
     return shim.Error("Employee does not exist")
   }

   err = stub.DelState(employeeIdAsString)
   if err != nil {
     return shim.Error("Failed to delete employee:" + employeeIdAsString + e
rr.Error())
   }
   return shim.Success(nil)
}
```

2.6 查询员工

```
// 根据员工ID查询员工信息
func (t *SmartContract) searchEmployeeInfoByID(stub shim.ChaincodeStubInter
face, args []string) pb.Response {
   if len(args) < 1 {
     return shim.Error("Incorrect number of arguments. Expecting 1 like (emp
loyee_id)")
   }
   employeeIdAsString := args[0]
     employeeAsBytes, err := stub.GetState(employeeIdAsString)
     if err != nil {
         return shim.Error("Failed to get employee info:" + err.Error())
     } else if employeeAsBytes == nil {
         return shim.Error("Employee does not exist")
   }
```

```
    fmt.Printf("Search Response:%s\n", string(employeeAsBytes))
      return shim.Success(employeeAsBytes)
  }
```

2.6 更新员工

更新员工信息之前，需要判断传入的部门ID是否存在，存在则更新。不存在的话，这里的做法是直接返回错误信息。

```
  // 更新员工信息
  func (t *SmartContract) updateEmployeeInfo(stub shim.ChaincodeStubInterface
, args []string) pb.Response {
    // 将字符串数组类型数据转换为"员工"结构体
    employee, err := translateEmployeeFromArgs(args)
    if err != nil {
      return shim.Error(err.Error())
    }

    employeeIdAsString := strconv.Itoa(employee.EmployeeID)
    // 检查添加的部门ID是否存在
    departmentIds := queryAllDepartmentIDs(stub)

    isExist := false
    if len(departmentIds) > 0 {
      for _, departmentId := range departmentIds {
        // 转换为int类型
        departmentIdAsInt, err := strconv.Atoi(departmentId)
        if err != nil {
              return shim.Error("Department Id argument must be a numeric str
ing")
          }

        if departmentIdAsInt == employee.DepartmentID {
          isExist = true
          break
        }
      }
    }

    if isExist {
      /*
      * State DB是一个Key-Value数据库，如果我们指定的Key在数据库中已经存在，那么就是修
改操作。
      * 如果Key不存在，那么就是插入操作。
      */
      employeeAsJsonBytes, err := json.Marshal(employee)
      if err != nil {
```

```
        return shim.Error(err.Error())
      }
      // 保存到账本中
      err = stub.PutState(employeeIdAsString, employeeAsJsonBytes)
      if err != nil {
        return shim.Error(err.Error())
      }
      return shim.Success(employeeAsJsonBytes)
    } else {
      fmt.Println("department:" + string(employee.DepartmentID) + " does not
exist")
          return shim.Error("department:" + string(employee.DepartmentID) + "
 does not exist")
    }
  }
```

2.7 实现Invoke函数

```
  // 在链码每个事务中，Invoke会被调用。
  func (t *SmartContract) Invoke(stub shim.ChaincodeStubInterface) pb.Respons
e {
    fmt.Println("my_chaincode01 Invoke")

    function, args := stub.GetFunctionAndParameters()
    if function == "initDepartment" {
      return t.initDepartment(stub, args)
    } else if function == "addEmployee" {
      return t.addEmployee(stub, args)
    } else if function == "deleteEmployee" {
      return t.deleteEmployee(stub, args)
    } else if function == "searchEmployeeInfoByID" {
      return t.searchEmployeeInfoByID(stub, args)
    } else if function == "updateEmployeeInfo" {
      return t.updateEmployeeInfo(stub, args)
    }

    return shim.Error("Invalid Smart Contract function name.")
  }
```

3、链码单元测试

开发完链码后，可以利用shim.MockStub来进行单元测试，从而快速地调试和运行，是提高链码开发效率减少Bug的好方法。

于是，我们可以新建一个单元测试文件，一般命名为"链码文件名_test.go"，比如这里

为"my_chaincode01_test.go"，完整代码如下：

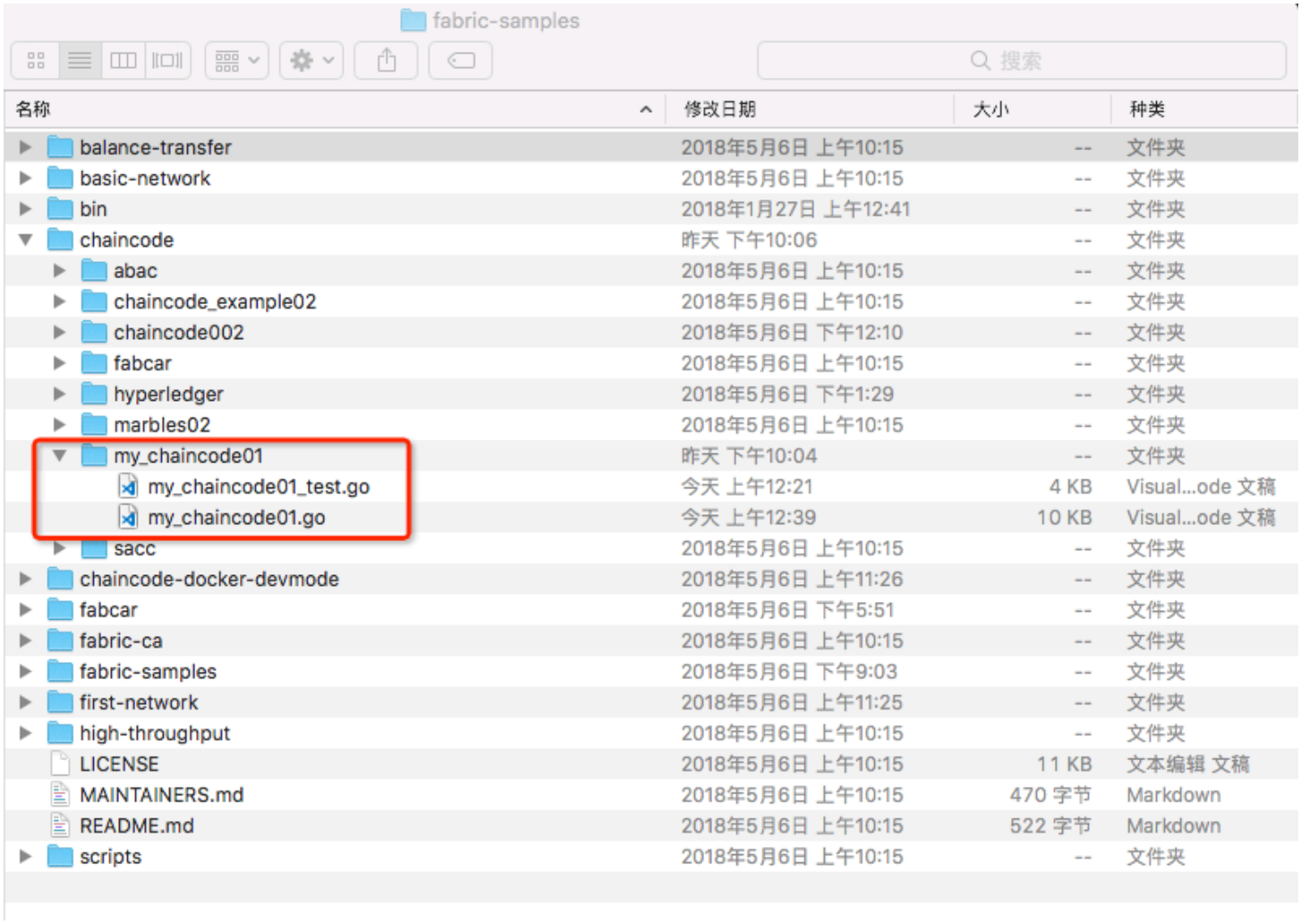
```
package main

/*
*
单元测试
go test -v my_chaincode01_test.go my_chaincode01.go
*/

import (
    "fmt"
    "testing"
    "github.com/hyperledger/fabric/core/chaincode/shim"
)

func mockInit(t *testing.T, stub *shim.MockStub, args [][]byte) {
    res := stub.MockInit("1", args)
    if res.Status != shim.OK {
        fmt.Println("Init failed", string(res.Message))
        t.FailNow()
    }
}

func initDepartment(t *testing.T, stub *shim.MockStub, args []string) {
    res := stub.MockInvoke("1", [][]byte{[]byte("initDepartment"), []byte(ar
gs[0]), []byte(args[1])})

    if res.Status != shim.OK {
        fmt.Println("InitDepartment failed:", args[0], string(res.Message))
        t.FailNow()
    }
}

func addEmployee(t *testing.T, stub *shim.MockStub, args []string) {
    res := stub.MockInvoke("1", [][]byte{[]byte("addEmployee"), []byte(args[
0]), []byte(args[1]), []byte(args[2]), []byte(args[3])})

    if res.Status != shim.OK {
        fmt.Println("AddEmployee failed:", args[0], string(res.Message))
        t.FailNow()
    }
}

func deleteEmployee(t *testing.T, stub *shim.MockStub, employeeId string) {
    res := stub.MockInvoke("1", [][]byte{[]byte("deleteEmployee"), []byte(em
ployeeId)})
```

```
    if res.Status != shim.OK {
        fmt.Println("DeleteEmployee :", employeeId, ", failed :", string(res
.Message))
        t.FailNow()
    }
}

func searchEmployeeInfoByID(t *testing.T, stub *shim.MockStub, employeeId st
ring) {
    res := stub.MockInvoke("1", [][]byte{[]byte("searchEmployeeInfoByID"), [
]byte(employeeId)})
    if res.Status != shim.OK {
        fmt.Println("searchEmployeeInfoByID :", employeeId, ", failed :", st
ring(res.Message))
        t.FailNow()
    }
    if res.Payload == nil {
        fmt.Println("SearchEmployeeInfoByID :" , employeeId, " failed to get
 value")
        t.FailNow()
    }
}

func updateEmployeeInfo(t *testing.T, stub *shim.MockStub, args []string) {
    res := stub.MockInvoke("1", [][]byte{[]byte("updateEmployeeInfo"), []byt
e(args[0]), []byte(args[1]), []byte(args[2]), []byte(args[3])})

    if res.Status != shim.OK {
        fmt.Println("UpdateEmployeeInfo failed:", args[0], string(res.Messag
e))
        t.FailNow()
    }
}


// 测试"初始化部门"
func TestInitDepartment(t *testing.T) {
    smartContract := new(SmartContract)
    stub := shim.NewMockStub("SmartContract", smartContract)
    mockInit(t, stub, nil)
    initDepartment(t, stub, []string{"1", "department_software"})
    initDepartment(t, stub, []string{"2", "department_test"})
}

// 测试"新增员工", 部门ID不存在时创建会失败
func TestAddEmployee(t *testing.T) {
    smartContract := new(SmartContract)
```

```
    stub := shim.NewMockStub("SmartContract", smartContract)
    mockInit(t, stub, nil)
    initDepartment(t, stub, []string{"1", "department_software"})
    addEmployee(t, stub, []string{"1", "Wenzil", "1", "Software Engineer"})
    // ID为"2"的部门没有创建，返回错误，所以先注释掉这一行
    // addEmployee(t, stub, []string{"2", "Test", "2", "Test Engineer"})
}

// 测试"查询员工信息"
func TestSearchEmployeeInfoByID(t *testing.T) {
    smartContract := new(SmartContract)
    stub := shim.NewMockStub("SmartContract", smartContract)
    mockInit(t, stub, nil)
    initDepartment(t, stub, []string{"2", "department_test"})
    addEmployee(t, stub, []string{"2", "Test", "2", "Test Engineer"})
    searchEmployeeInfoByID(t, stub, "2")
}

// 测试"删除员工"
func TestDeleteEmployee(t *testing.T) {
    smartContract := new(SmartContract)
    stub := shim.NewMockStub("SmartContract", smartContract)
    mockInit(t, stub, nil)
    initDepartment(t, stub, []string{"3", "department_ui"})
    addEmployee(t, stub, []string{"3", "Han Meimei", "3", "UI Designer"})
    deleteEmployee(t, stub, "3")
}

// 测试更新"员工信息"
func TestUpdateEmployeeInfo(t *testing.T) {
    smartContract := new(SmartContract)
    stub := shim.NewMockStub("SmartContract", smartContract)
    mockInit(t, stub, nil)
    initDepartment(t, stub, []string{"4", "department_blockchain"})
    addEmployee(t, stub, []string{"7", "Li Lei", "4", "Blockchain Designer"}
)
    updateEmployeeInfo(t, stub, []string{"7", "Li Lei", "4", "Blockchain Sen
ior Designer"})
    searchEmployeeInfoByID(t, stub, "7")
}
```

4、部署和测试链码

确保您搭建并配置好了Hyperledger Fabric的开发环境，我们把上面创建的链码文件夹复制到"fabric-samples"目录下。



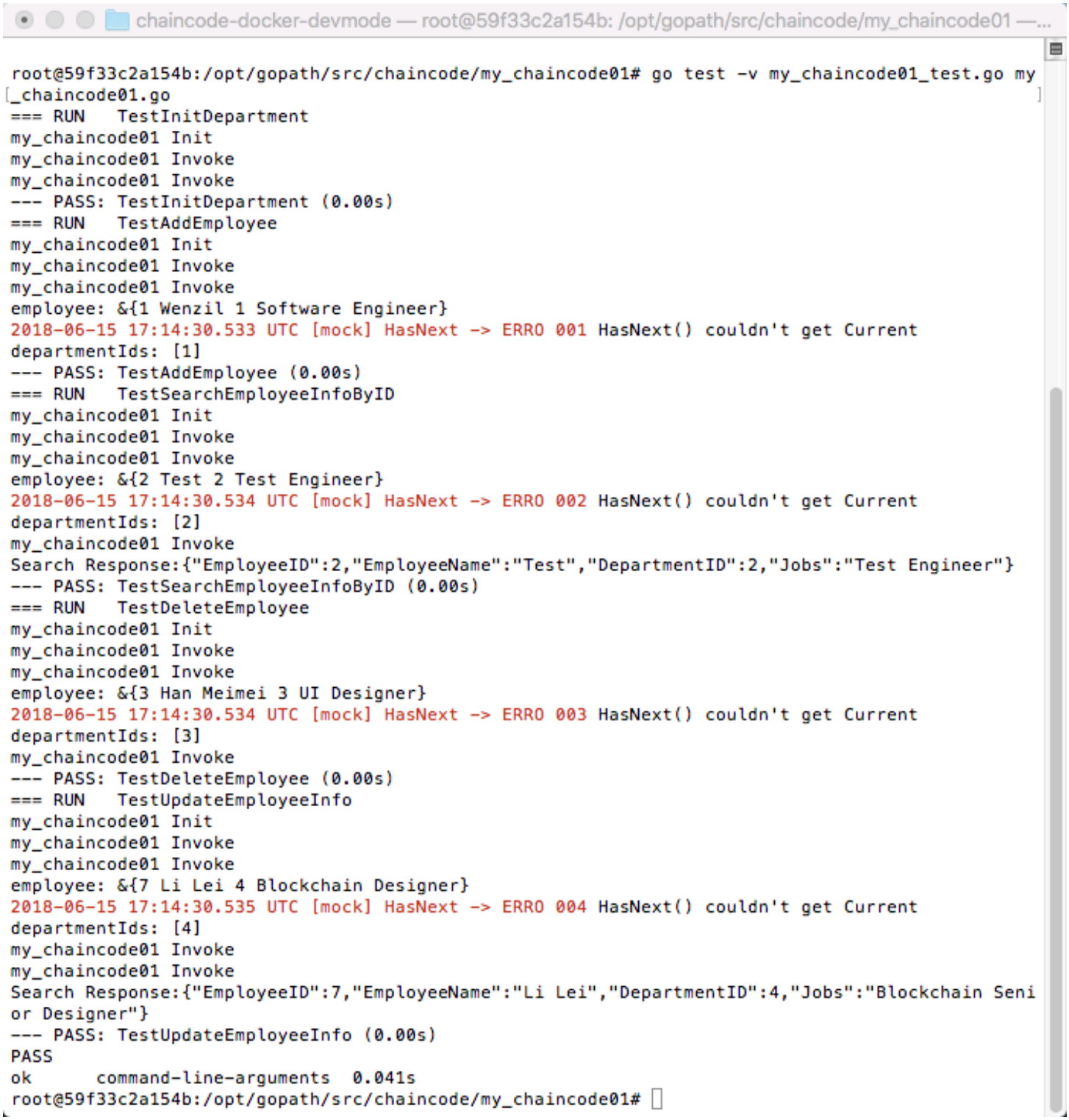
同时，开启三个终端，确保终端进入到"fabric-samples/chaincode-docker-devmode"目录下。

4.1 终端1 - 开启网络

```
###删除所有活跃的容器###
docker rm -f $(docker ps -aq)
###清理网络缓存###
docker network prune
###开启网络###
docker-compose -f docker-compose-simple.yaml up
```

4.2 终端2 - 编译和运行链码

```
###进入Docker容器cli###
docker exec -it chaincode bash
###进入到链码对应目录###
cd my_chaincode01
###执行单元测试命令###
go test -v my_chaincode01_test.go my_chaincode01.go
```



单元测试通过后，继续执行如下命令：

```
###编译链码###
go build
###启动节点###
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./my_chaincode01
###如果失败，把"7052"改为"7051"试试看
```

4.3 终端3 - 调用链码

1、启动Docker cli容器：

```
docker exec -it chaincode bash
```

2、安装和实例化链码：

```
peer chaincode install -p chaincodedev/chaincode/my_chaincode01 -n mycc -v 0
peer chaincode instantiate -n mycc -v 0 -c '{"Args":[]}' -C myc
```

3、初始化部门：

```
peer chaincode invoke -n mycc -c '{"Args":["initDepartment","1","department_
software"]}' -C myc
2018-06-15 17:31:32.191 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200 payload:"{\"DepartmentID\"
:1,\"DepartmentName\":\"department_software\"}"
```

3、新增员工：

```
peer chaincode invoke -n mycc -c '{"Args":["addEmployee","1","Wenzil","1","S
oftware Engineer"]}' -C myc
2018-06-15 17:33:21.046 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200 payload:"{\"EmployeeID\":1
,\"EmployeeName\":\"Wenzil\",\"DepartmentID\":1,\"Jobs\":\"Software Engineer
\"}"
peer chaincode invoke -n mycc -c '{"Args":["addEmployee","2","Li Lei","1","A
I Engineer"]}' -C myc
2018-06-15 17:35:12.030 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200 payload:"{\"EmployeeID\":2
,\"EmployeeName\":\"Li Lei\",\"DepartmentID\":1,\"Jobs\":\"AI Engineer\"}"
```

4、更新员工：

```
peer chaincode invoke -n mycc -c '{"Args":["updateEmployeeInfo","2","Li Lei"
,"1","Blockchain Engineer"]}' -C myc
2018-06-15 17:39:25.819 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200 payload:"{\"EmployeeID\":2
,\"EmployeeName\":\"Li Lei\",\"DepartmentID\":1,\"Jobs\":\"Blockchain Engine
er\"}"
```

5、查询员工：

```
peer chaincode invoke -n mycc -c '{"Args":["searchEmployeeInfoByID","2"]}' -
C myc
2018-06-15 17:40:58.217 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200 payload:"{\"EmployeeID\":2
,\"EmployeeName\":\"Li Lei\",\"DepartmentID\":1,\"Jobs\":\"Blockchain Engine
er\"}"
```

6、删除员工：

```
peer chaincode invoke -n mycc -c '{"Args":["deleteEmployee","2"]}' -C myc
2018-06-15 17:41:51.422 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200
```

7、再次查询：

```
peer chaincode invoke -n mycc -c '{"Args":["searchEmployeeInfoByID","2"]}' -
C myc
###直接报错###
Error: Error endorsing invoke: rpc error: code = Unknown desc = chaincode er
ror (status: 500, message: Employee does not exist) - <nil>
......

###改为查询ID为"1"的员工###
2018-06-15 17:43:56.039 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 06
3 Chaincode invoke successful. result: status:200 payload:"{\"EmployeeID\":1
,\"EmployeeName\":\"Wenzil\",\"DepartmentID\":1,\"Jobs\":\"Software Engineer
\"}"
```