**实验10 使用Java编写链码**

**【实验介绍】**

在实验8中，我们搭建了Fabric环境，运行了e2e案例，实现了账户资本的查询以及转账操作。在实验9中详细讲解了该案例的工作流程和智能合约源代码，相信同学们对Fabric已经有了初步的了解。Fabric的智能合约称为链码（chaincode），要编写链码，开发人员需要实现Fabric预先定义的接口，以可插拔的形式置区块链系统中，来实现业务逻辑。本次实验的内容是使用Java实现一个与e2e案例中功能相同的链码，并进行部署和调用。  
 在进行实验操作之前，我们首先了解链码的相关概念，学习编写一个典型链码的基本流程，并学会使用命令加以管理链码的生命周期。

1. 链码概述

一般而言，链码可以分为用户链码（User Chaincode）和系统链码（System Chaincode）。用户链码为用户开发的链码，用于实现特定的业务逻辑，Fabric使用Docker容器作为用户链码的运行环境，通过gRPC与Fabric进行通信。系统链码负责账本和链的配置管理、背书签名、链码生命周期、账本和链的信息查询功能，验证策略等，运行在Fabric节点（peer）进程中。与用户链码相同，系统链码也需要实现Chaincode接口，程序的开发框架和开发流程基本一致。不同的是，为了让系统链码在节点进程中运行，系统链码需要和节点进程代码一同发布。

Fabric的系统链码主要包括LSCC（Lifecycle system chaincode）生命周期系统链码，负责管理用户链码的生命周期；CSCC（Configuration system chaincode）配置系统链码，负责配置Peer端的Channel；QSCC（Query system chaincode）查询系统链码，提供账本查询 API，如获取区块和交易等信息；ESCC（Endorsement system chaincode）背书系统链码，负责签名过程；VSCC（Validation system chaincode）验证系统链码等，处理交易的验证。具体系统链码的实现在fabric/core/scc文件夹中，感兴趣的读者可以延伸阅读。

1. 链码开发

Fabric支持多种高级语言如Go、Java、Node.js等进行开发，利于开发人员的快速上手。本实验主要内容为Java实现链码，下面展示一个Java链码的基本结构（基于fabric-chaincode-java-1.4.0），如表10-1所示。其中，重要的抽象类ChaincodeBase和接口ChaincodeStub定义在org.hyperledger.fabric.shim包中。

1. ChaincodeBase：抽象类，所有链码的实现都必须继承该类，主要包括init和invoke两个抽象方法。
   1. init：链码在初始化和升级时的主入口，初始化相关数据和应用状态。
   2. invoke：链码业务逻辑的主入口，可以在此编写业务逻辑或调用自定义方法以实现更细致的业务划分。
2. ChaincodeStub：接口，管理交易的上下文，提供对状态变量的访问，并支持对其他链码的实现进行调用。

表10-1 Java链码基本结构

|  |
| --- |
| package org.hyperledger.fabric.example;  import org.hyperledger.fabric.shim.ChaincodeBase;  import org.hyperledger.fabric.shim.ChaincodeStub;  public class SimpleChaincode extends ChaincodeBase {  /\*  \* @Description: 实现父类ChaincodeBase的抽象方法init  \* @Params: [chaincodeStub]  \* @return: org.hyperledger.fabric.shim.Chaincode.Response  \*\*/  @Override  public Response init(ChaincodeStub chaincodeStub) {  // 初始化链码  // ...  // 响应  return newSuccessResponse();  }  /\*  \* @Description: 实现父类ChaincodeBase的抽象方法invoke  \* @Params: [chaincodeStub]  \* @return: org.hyperledger.fabric.shim.Chaincode.Response  \*\*/  @Override  public Response invoke(ChaincodeStub chaincodeStub) {  // 从交易中获取参数或命令  // ...  // 实现业务逻辑或调用其他方法  // ...  // 响应  return newSuccessResponse();  }  /\*  \* @Description: 主方法，由容器调用以启动链码  \* @Params: [args]  \* @return: void  \*\*/  public static void main(String[] args) {  try {  new SimpleChaincode().start(args);  } catch (Exception e) {  e.printStackTrace();  }  }  } |

1. 链码管理

链码是业务逻辑的具体实现，链码的生命周期指链码从上线到下线的一系列过程。开发者可以使用LSCC来管理用户链码的生命周期，目前，Fabric链码的生命周期管理命令分为以下四个：

1. package（打包）：将编写好的链码进行打包；
2. install（安装）：将链码安装在Fabric中；
3. instantiate（实例化）：将已安装的链码实例化为进程；
4. upgrade（升级）：对已存在的链码进行升级。

**【实验要求】**

1. 使用Java编写e2e链码
2. 链码部署与调用
3. 熟悉e2e案例的工作原理

**【实验准备】**

**1. 环境配置**

由于Fabric在1.3版本之后才支持Java链码，所以本次实验使用Fabric 1.3环境。具体的使用方法请根据以下步骤进行：

为了方便大家进行实验和试错，我们将实验所需的修改后的fabric代码上传到了github上，大家简单的通过git clone命令下载即可。

首先我们进入到路径/home/dase/go/src/github.com/hyperledger中，可以看到实验8、9所使用的fabric文件夹，可以直接rm -rf fabric将其删除，也可以使用mv更换文件夹名，如下图10-1所示。

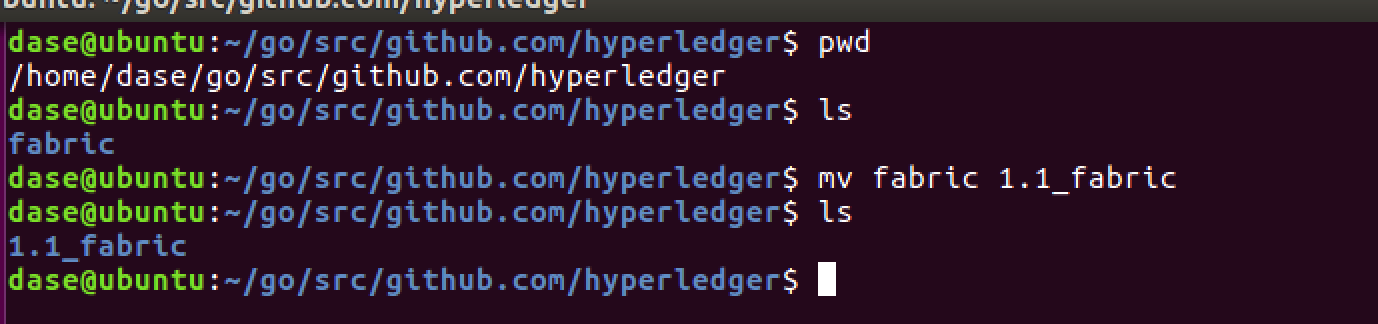


图10-1 更换文件夹名称

**注意：如果大家选择删除原有的fabric的话，一定要记得先把之前启动的docker通过./network.sh down关闭了。另外，之前我们下载了1.1版本的docker镜像，这次实验我们还要下载1.3版本的镜像，可能会导致虚拟机磁盘空间不足。这里建议大家开两个虚拟机或者将之前下载的docker镜像通过docker rmi -f $(docker images -qa)删除。**

在做好相关处理后，我们在/home/dase/go/src/github.com/hyperledger下对实验代码进行克隆，简单输入如下命令即可：

git clone <https://github.com/ZhmYe/Fabric_1.3_Lab_10.git> fabric

**注意：这边一定要克隆到名字叫做fabric的文件夹下，fabric官方脚本里强制要求了目录名称。另外如果git的速度比较慢的话，肯定通过git clone** [**https://github.yuuza.net/ZhmYe/Fabric\_1.3\_Lab\_10.git**](https://github.yuuza.net/ZhmYe/Fabric_1.3_Lab_10.git) **fabric加快一下速度。**

进入到路径/home/dase/go/src/github.com/hyperledger/fabric/examples/chaincode/java中，可以看到有一个examplejava的文件夹，和附件中提供的java工程项目名称一样，如图10-2所示。

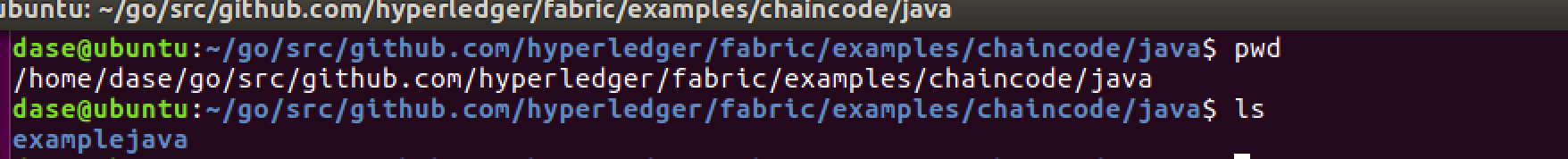


图10-2 实验代码关键路径

由于克隆下来的代码有权限问题，会导致后续.sh文件无法运行，这边我们提前先对一些必要的.sh加上可执行权限。

在fabric根目录下，scripts文件夹下运行chmod +x \*.sh。

在fabric/example/e2e\_cli/scripts文件夹下,chmod +x script.sh。

在fabric/example/e2e\_cli/文件夹下运行chmod +x generateArtifacts.sh network\_setup.sh。

**【实验过程】**

相关的配置修改已经完成，无需像实验八一样进行修改，即使后续大家遇到其它问题需要重装也不用改配置。实验过程中如果还是遇到order或peer无法连接的情况，可能是因为超出了timeout的限制，大家可以把fabric/example/e2e\_cli/scripts/script.sh中的timeout改成120.



补全SimpleChaincode.java中三个方法的逻辑，其中，该链码要实现的逻辑，与上一份文件中提供的代码文件chaincode\_example02\_e2e.go完全一致。读者可以参考chaincode\_example02\_e2e.go中的代码逻辑，进行java版本的编写。

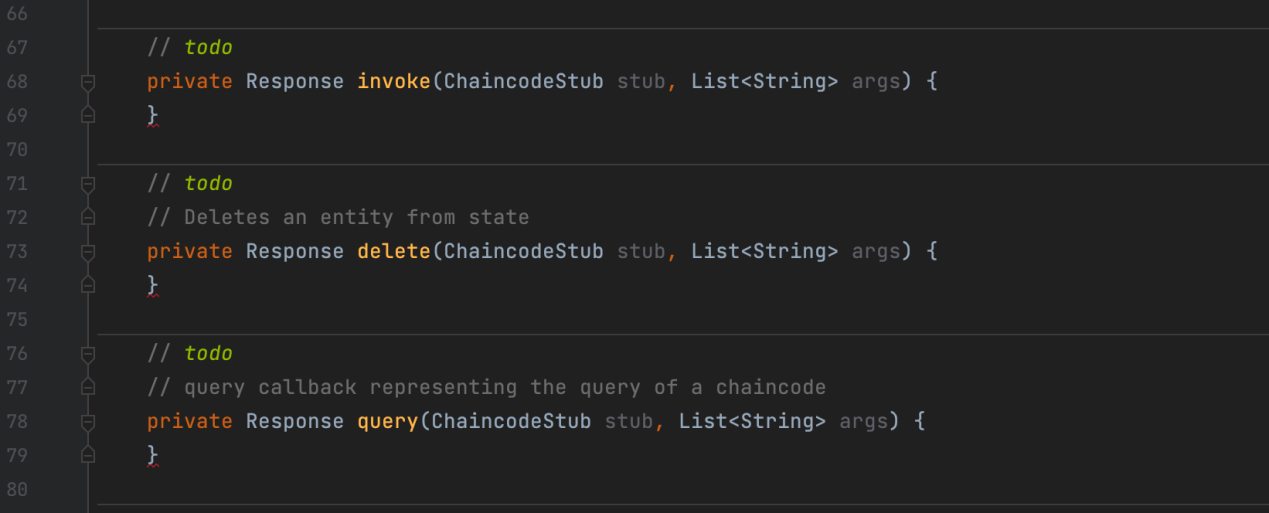
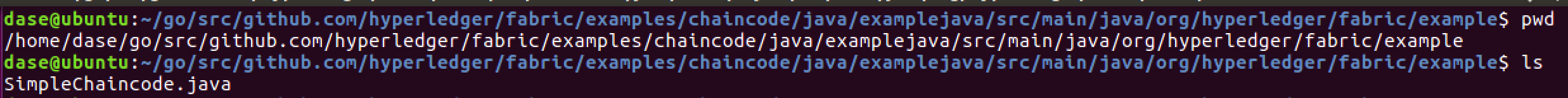


图10-5 实验任务

链码是有状态的，随着Fabric的运行和运行，无法直接对链码进行测试。读者可以在完成代码后，直接将代码放入e2e案例中运行，看运行结果是否与Go版本的链码相同。

读者补全java代码后，将代码文件SimpleChaincode.java，放至以下路径中：

图10-6 代码路径

请勿自行修改文件名，也不要删除、移动examplejava文件夹下的内容，也请严格按照下图的路径进行放置，否则无法成功运行。

根据上述步骤放置SimpleChaincode.java文件后，就可以进行e2e案例的运行了，但是在运行之前，需要完成几个步骤，首先，如果读者之前有用1.1版本运行过./network\_setup.sh up，请务必确认运行./network\_setup.sh down，以停止Fabric服务，并删除相关的容器，然后使用以下命令删除原先的镜像：

|  |
| --- |
| # 删除旧镜像  docker rmi -f $(docker images -qa)  # 获取1.3镜像  source download-dockerimages.sh  # 初始化证书  ./generateArtifacts.sh |

最后，进入到e2e文件夹中，执行./network\_setup.sh up，进行案例的运行，若没有明显的报错，那么说明你的链码实现是正确的。

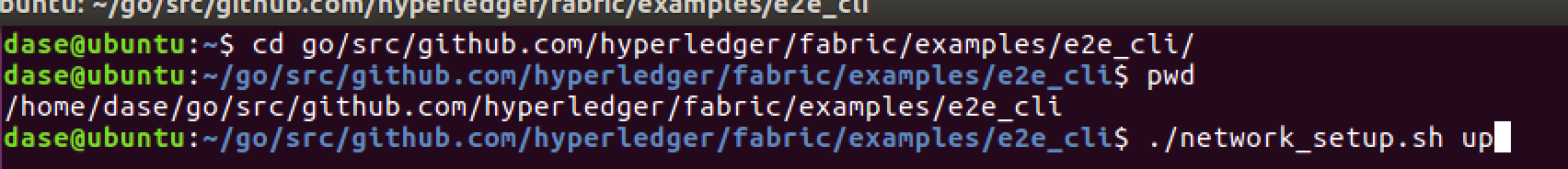


图10-7 运行e2e案例

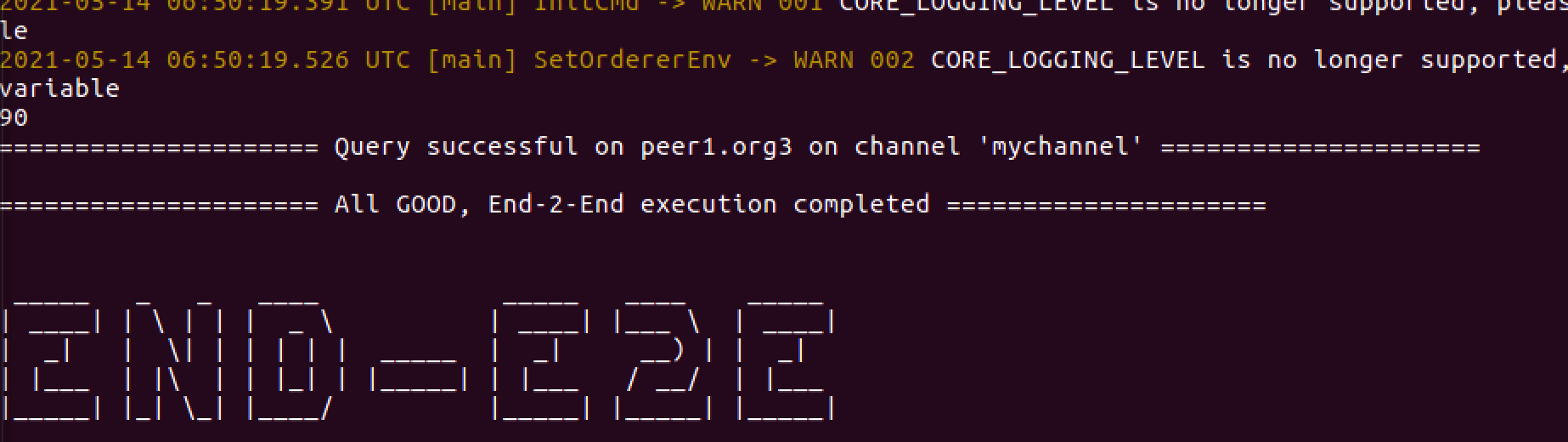


图10-8 成功样例

**【实验小结】**

本实验阐述了Fabric中的智能合约——链码的一些基本概念，包括链码的分类、主要系统链码、链码的生命周期以及链码管理命令等，并以Java链码的基本结构为例，着重讨论了用户链码的开发流程。与以太坊智能合约不同，Fabric预先定义了一套开发接口，支持多种高级语言来实现链码，并在docker容器中运行。通过本实验的理论学习与工程实践，读者应该能够独立编写、部署并调用Java链码。

**【习题】**

1. Fabric链码的种类该怎么划分？

2. 链码的生命周期有哪些？

3. Fabric chaincode和以太坊智能合约有什么不同？

**【参考文献】**

1. 范吉立, 李晓华, 聂铁铮, 等. 区块链系统中智能合约技术综述[J]. 计算机科学, 2019, 46(11): 1-10.
2. 林诗意,张磊,刘德胜.基于区块链智能合约的应用研究综述[J].计算机应用研究, 2021, 38(09):2570-2581.
3. Hyperledger Community. Hyperledger Fabric(OL). (2020-06-01) [2020-07-12]. <https://github.com/hyperledger/fabric>.
4. Hyperledger Community. Smart Contracts and Chaincode(OL). (2020-08-20) [2020-12-11]. <https://github.com/hyperledger/fabric-docs-i18n/blob/release-2.2/docs/locale/es/source/smartcontract/smartcontract.md>.

[5] Hyperledger Fabric Chaincode编程接口.

https://hyperledger-fabric.readthedocs.io/zh\_CN/latest/chaincode4ade.html

**【附录】**

附表1 SimpleChainCode.java

|  |
| --- |
| package org.hyperledger.fabric.example;  import java.util.List;  import com.google.protobuf.ByteString;  import io.netty.handler.ssl.OpenSsl;  import org.apache.commons.logging.Log;  import org.apache.commons.logging.LogFactory;  import org.hyperledger.fabric.shim.ChaincodeBase;  import org.hyperledger.fabric.shim.ChaincodeStub;  import static java.nio.charset.StandardCharsets.UTF\_8;  public class SimpleChaincode extends ChaincodeBase {  private static Log \_logger = LogFactory.getLog(SimpleChaincode.class);  @Override  public Response init(ChaincodeStub stub) {  try {  \_logger.info("Init java simple chaincode");  String func = stub.getFunction();  if (!func.equals("init")) {  return newErrorResponse("function other than init is not supported");  }  List<String> args = stub.getParameters();  if (args.size() != 4) {  newErrorResponse("Incorrect number of arguments. Expecting 4");  }  // Initialize the chaincode  String account1Key = args.get(0);  int account1Value = Integer.parseInt(args.get(1));  String account2Key = args.get(2);  int account2Value = Integer.parseInt(args.get(3));  \_logger.info(String.format("account %s, value = %s; account %s, value %s", account1Key, account1Value, account2Key, account2Value));  stub.putStringState(account1Key, args.get(1));  stub.putStringState(account2Key, args.get(3));  return newSuccessResponse();  } catch (Throwable e) {  return newErrorResponse(e);  }  }  @Override  public Response invoke(ChaincodeStub stub) {  try {  \_logger.info("Invoke java simple chaincode");  String func = stub.getFunction();  List<String> params = stub.getParameters();  if (func.equals("invoke")) {  return invoke(stub, params);  }  if (func.equals("delete")) {  return delete(stub, params);  }  if (func.equals("query")) {  return query(stub, params);  }  return newErrorResponse("Invalid invoke function name. Expecting one of: [\"invoke\", \"delete\", \"query\"]");  } catch (Throwable e) {  return newErrorResponse(e);  }  }  private Response invoke(ChaincodeStub stub, List<String> args) {  if (args.size() != 3) {  return newErrorResponse("Incorrect number of arguments. Expecting 3");  }  String accountFromKey = args.get(0);  String accountToKey = args.get(1);  String accountFromValueStr = stub.getStringState(accountFromKey);  if (accountFromValueStr == null) {  return newErrorResponse(String.format("Entity %s not found", accountFromKey));  }  int accountFromValue = Integer.parseInt(accountFromValueStr);  String accountToValueStr = stub.getStringState(accountToKey);  if (accountToValueStr == null) {  return newErrorResponse(String.format("Entity %s not found", accountToKey));  }  int accountToValue = Integer.parseInt(accountToValueStr);  int amount = Integer.parseInt(args.get(2));  if (amount > accountFromValue) {  return newErrorResponse(String.format("not enough money in account %s", accountFromKey));  }  accountFromValue -= amount;  accountToValue += amount;  \_logger.info(String.format("new value of A: %s", accountFromValue));  \_logger.info(String.format("new value of B: %s", accountToValue));  stub.putStringState(accountFromKey, Integer.toString(accountFromValue));  stub.putStringState(accountToKey, Integer.toString(accountToValue));  \_logger.info("Transfer complete");  return newSuccessResponse("invoke finished successfully", ByteString.copyFrom(accountFromKey + ": " + accountFromValue + " " + accountToKey + ": " + accountToValue, UTF\_8).toByteArray());  }  // Deletes an entity from state  private Response delete(ChaincodeStub stub, List<String> args) {  if (args.size() != 1) {  return newErrorResponse("Incorrect number of arguments. Expecting 1");  }  String key = args.get(0);  // Delete the key from the state in ledger  stub.delState(key);  return newSuccessResponse();  }  // query callback representing the query of a chaincode  private Response query(ChaincodeStub stub, List<String> args) {  if (args.size() != 1) {  return newErrorResponse("Incorrect number of arguments. Expecting name of the person to query");  }  String key = args.get(0);  //byte[] stateBytes  String val = stub.getStringState(key);  if (val == null) {  return newErrorResponse(String.format("Error: state for %s is null", key));  }  \_logger.info(String.format("Query Response:\nName: %s, Amount: %s\n", key, val));  return newSuccessResponse(val, ByteString.copyFrom(val, UTF\_8).toByteArray());  }  public static void main(String[] args) {  System.out.println("OpenSSL avaliable: " + OpenSsl.isAvailable());  new SimpleChaincode().start(args);  }  } |