Blockchain Service

Developer Guide

Issue 01

Date 2023-03-07

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions

Notice

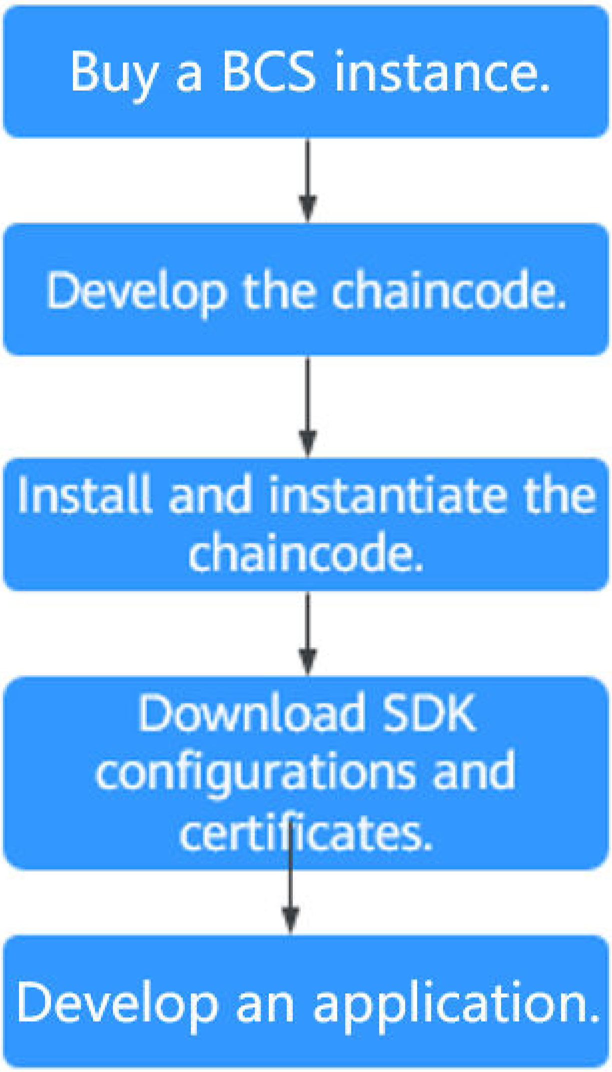
Contents

1 Overview

To use Blockchain Service (BCS), you must develop your own chaincodes and applications. This document describes chaincode development and application configuration for developers with Go or Java development experience.

The process is as follows:

Figure 1-1 Development process



1. Create a BCS instance.

   BCS instances can be deployed in CCE clusters. For details, see **Using a CCE Cluster**.

2. Develop the chaincode.

   A chaincode is a program written in Go for operating the ledger. For details, see **Chaincode Development**.

3. Install and instantiate the chaincode.

   BCS provides a graphical user interface (GUI) for chaincode management, including chaincode installation and instantiation. For details, see **Chaincode Management**.

4. Download SDK configurations and certificates.

   Before developing an application, obtain the required SDK configuration file and certificates. For details, see **BCS Access**.

2 Chaincode Development

2.1 Development Preparation

A chaincode, also called a smart contract, is a program written in Go or Node.js for operating the ledger. It is a code logic that runs on a blockchain and is automatically executed under specific conditions. Chaincodes are an important method for users to implement service logic when using blockchains. Due to the blockchain features, the execution results of smart contracts are reliable and cannot be forged or tampered with.

To use BCS, you must develop your own chaincodes and applications. Applications invoke chaincodes through peers in the blockchain network, and the chaincodes operate ledger data through peers.

Preparing the Development Environment

1. Install the Go development environment. Download the installation package from **https://golang.org/dl/**. (Select a version later than 1.9.2.)

   The package name for each OS is as follows (version 1.11.12 is used as an example):

OS	Package
Windows	go1.11.12.windows-amd64.msi
Linux	go1.11.12.linux-amd64.tar.gz

   – In Windows, you can use the .msi installation package for installation. By default, the .msi file is installed in **C:\Go**. You can add the **C:\Go\bin** directory to the **Path** environment variable. Restart the CLI for the settings to take effect.
   – In Linux, decompress the downloaded binary package to the **/usr/local** directory. Add the **/usr/local/go/bin** directory to the Path environment variable.
   ```
   export PATH=$PATH:/usr/local/go/bin
   ```


After Golang is installed, you can run the **go version** command to view the version information and run the **go env** command to view the path configuration.

2. Install a Go editor of your choice. **GoLand** is recommended.

Downloading the Source Code Package

Download the Fabric source code package as the third-party repository.

Download the required version from the following addresses:

https://github.com/hyperledger/fabric/tree/release-2.2

NOTE

The version of the Fabric package should match that of the blockchain. For example, if a blockchain is v4.x.x, it uses Fabric v2.2, so you need to download the Fabric v2.2 package.

2.2 Development Specifications

Preventing Chaincode Container from Failing After a Panic

NOTE

This section applies only to the Go chaincode development.

When a panic exception occurs, the chaincode container may be suspended and restarted, logs cannot be found, and the problem cannot be located immediately. To prevent this case, add the defer statement at the entry point of the Invoke function. When a panic occurs, the error is returned to the client.

```
// Name the return value in the defer statement to ensure that the client can receive the value if a panic
occurs.
//Use debug.PrintStack() to print the stack trace to the standard output for fault locating.
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) (pr pb.Response) {
   defer func() {
      if err:=recover(); err != nil {
         fmt.Println("recover from invoke:", err)
         debug.PrintStack()
         pr = shim.Error(fmt.Sprintf("invoke painc. err: %v",err))
      }
   }()

   fmt.Println("ex02 Invoke")
   function, args := stub.GetFunctionAndParameters()
   if function == "invoke" {
      // Make payment of X units from A to B
      return t.invoke(stub, args)
   } else if function == "delete" {
      // Deletes an entity from its state
      return t.delete(stub, args)
   } else if function == "query" {
      // the old "Query" is now implemented in invoke
      return t.query(stub, args)
   }

   pr = shim.Error("Invalid invoke function name. Expecting \"invoke\" \"delete\" \"query\"")
   return pr
}
```


Querying Data in Batches

If too many records are returned in one query, too many resources will be occupied and the interface delay will be long. For example, if the interface delay exceeds 30s, the peer task will be interrupted. Therefore, estimate the data volume in advance, and query data in batches if the data volume is large.

To modify or delete the ledger data chaincode, consider performing operations in batches based on the data volume.

Using Indexes with CouchDB

Using indexes with CouchDB accelerates data querying, but slows data writing. Therefore, create indexes only for certain fields based on service requirements.

Verifying Permissions

Verify the permissions of the smart contract executor to prevent unauthorized users from executing chaincodes.

NOTE

If no specified organization is required for the endorsement, select at least two endorsing organizations to ensure that the chaincode data is not maliciously modified (such as installing invalid chaincode or processing data) by any other organizations.

Verifying Parameters

Before parameters (including input parameters and parameters defined in code) are used, the quantity, type, length, and value range of the parameters must be verified to prevent array out-of-range.

Processing Logs

During the development of services that have a complex logic and are prone to error, use **fmt** to print logs to facilitate debugging. **fmt** consumes a lot of time and resources. Delete the logs after the debugging is complete.

Configuring Dependencies

NOTE

This section applies only to the Java chaincode development.

Use Gradle or Maven to manage chaincode projects. If the chaincode project contains non-local dependencies, ensure that all nodes of the BCS instance are bound with EIPs. If the chaincode container runs in a restricted network environment, ensure that all dependencies in the project are configured as local dependencies. To obtain the chaincode used in this section, go to the BCS console and click **Use Cases**. Download **Chaincode_Java_Local_Demo** in the **Java SDK Demo** area.

2.3 Go Chaincode Development

2.3.1 Chaincode Structure

A chaincode is a Go file. After creating a chaincode, you can use it to develop functions.

Chaincode Interface

- To start a chaincode, you must call the Start function in the shim package (1.4 style). The input parameter is the Chaincode interface type defined in the shim package. During chaincode development, define a structure to implement the Chaincode interface.
```
type Chaincode interface {
   Init(stub ChaincodeStubInterface) pb.Response
   Invoke(stub ChaincodeStubInterface) pb.Response
}
```
- When developing chaincodes of the 2.2 style (using fabric-contract-api-go), define a structure to implement the Chaincode interface.
```
type Chaincode interface {
    Init(ctx contractapi.TransactionContextInterface, args...) error
    Invoke(ctx contractapi.TransactionContextInterface, args...) error
}
```

Chaincode Structure

- The Go chaincode structure (1.4 style) is as follows:
```
package main

//Import the required package.
import (
   "github.com/hyperledger/fabric/core/chaincode/shim"
   pb "github.com/hyperledger/fabric/protos/peer"
)

//Declare a structure.
type SimpleChaincode struct {}

//Add the Init method to the structure.
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
        //Implement the processing logic for chaincode initialization or update in this method.
    //stub APIs can be flexibly used during compilation.
}

//Add the Invoke method to the structure.
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
        //Implement the processing logic for responding to the call or query in this method.
    //stub APIs can be flexibly used during compilation.
}

//Main function. The shim.Start() method needs to be invoked.
func main() {
   err := shim.Start(new(SimpleChaincode))
   if err != nil {
      fmt.Printf("Error starting Simple chaincode: %s", err)
   }
}
```
- The Go chaincode structure (2.2 style) is as follows:
```
package main

//Import the required package.
import (
 "github.com/hyperledger/fabric/plugins/fabric-contract-api-go/contractapi")
```



```
//Declare a structure.
type Chaincode struct {
   contractapi.Contract
}

//Add the Init method to the structure.
func (ch * Chaincode) Init(ctx contractapi.TransactionContextInterface, args...) error {
      //Implement the processing logic for chaincode initialization or update in this method.
}

//Add the Invoke method to the structure.
func (ch * Chaincode) Invoke(ctx contractapi.TransactionContextInterface, args...) error {
      //Implement the processing logic for responding to the call or query in this method.
}

//Main function
func main() {
   cc, err := contractapi.NewChaincode(new(ABstore))
   if err != nil {
      panic(err.Error())
   }
   if err := cc.Start(); err != nil {
      fmt.Printf("Error starting ABstore chaincode: %s", err)
   }
}
```

2.3.2 Chaincode APIs

The shim package in the Fabric source code package provides the following types of APIs:

- Parameter parsing APIs: used to parse parameters transferred to the invoked function or method during chaincode invocation
- Ledger data operation APIs: used to provide methods for performing operations on ledger data, including status data query and transaction processing
- Transaction obtaining APIs: used to obtain information about transaction proposals
- APIs for private data operations: used to perform operations on private data (the APIs are available since Hyperledger Fabric v1.2.0)
- Other APIs: used to set events and invoke other chaincodes

2.3.3 Sample Chaincode (1.4)

The following is an example of installing and instantiating the account transfer chaincode (1.4). For details about how to debug this chaincode, see **the official Fabric examples**.

```
package main

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

type SimpleChaincode struct {
}

//Automatically invoked during chaincode instantiation or update to initialize the data status.
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
   //The output information of the println function is displayed in the logs of the chaincode container.
```

```
    fmt.Println("ex02 Init")

    //Obtain the parameters transferred by the user to invoke the chaincode.
    _, args := stub.GetFunctionAndParameters()

    var A, B string //Two accounts
    var Aval, Bval int //Balances of the two accounts
    var err error

    //Check whether the number of parameters is 4. If not, an error message is returned.
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments. Expecting 4")
    }

    A = args[0] //Username of account A
     Aval, err = strconv.Atoi(args[1]) //Balance in account A
    if err != nil {
        return shim.Error("Expecting integer value for asset holding")
    }

    B = args[2] //Username of account B
    Bval, err = strconv.Atoi(args[3])  //Balance in account B
    if err != nil {
        return shim.Error("Expecting integer value for asset holding")
    }

    fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

    //Write the status of account A to the ledger.
    err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
    if err != nil {
        return shim.Error(err.Error())
    }

    //Write the status of account B to the ledger.
    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(nil)
}

//The function is automatically invoked to query or invoke the ledger data.
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    fmt.Println("ex02 Invoke")
    //Obtain the function name and parameters transferred by the user to invoke the chaincode.
    function, args := stub.GetFunctionAndParameters()

     //Check the obtained function name.
    if function == "invoke" {
        //Invoke the invoke function to implement the money transfer operation.
        return t.invoke(stub, args)
    } else if function == "delete" {
        //Invoke the delete function to deregister the account.
        return t.delete(stub, args)
    } else if function == "query" {
        //Invoke the query interface to query the account.
        return t.query(stub, args)
    }
    //If the transferred function name is incorrect, shim.Error() is returned.
    return shim.Error("Invalid invoke function name. Expecting \"invoke\" \"delete\" \"query\"")
}

//Transfer money between accounts.
func (t *SimpleChaincode) invoke(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    var A, B string //Accounts A and B
    var Aval, Bval int //Account balance
    var X int //Transfer amount
```

```
    var err error

    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }

    A = args[0]       //Username of account A
    B = args[1]       //Username of account B

    //Obtain the balance of account A from the ledger.
    Avalbytes, err := stub.GetState(A)
    if err != nil {
        return shim.Error("Failed to get state")
    }
    if Avalbytes == nil {
        return shim.Error("Entity not found")
    }
    Aval, _ = strconv.Atoi(string(Avalbytes))

    //Obtain the balance of account B from the ledger.
    Bvalbytes, err := stub.GetState(B)
    if err != nil {
        return shim.Error("Failed to get state")
    }
    if Bvalbytes == nil {
        return shim.Error("Entity not found")
    }
    Bval, _ = strconv.Atoi(string(Bvalbytes))

    //X indicates the transfer amount.
    X, err = strconv.Atoi(args[2])
    if err != nil {
        return shim.Error("Invalid transaction amount, expecting a integer value")
    }
    //Transfer
    Aval = Aval - X
    Bval = Bval + X
    fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

    //Update the balance of account A after the transfer.
    err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
    if err != nil {
        return shim.Error(err.Error())
    }

    //Update the balance of account B after the transfer.
    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(nil)
}

//Account deregistration
func (t *SimpleChaincode) delete(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }

    A = args[0] //Username

    //Delete the account status from the ledger.
    err := stub.DelState(A)
    if err != nil {
        return shim.Error("Failed to delete state")
    }

    return shim.Success(nil)
```

```
}

//Account query
func (t *SimpleChaincode) query(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    var A string
    var err error

    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting name of the person to query")
    }

    A = args[0]   //Username

    //Obtain the balance of the account from the ledger.
    Avalbytes, err := stub.GetState(A)
    if err != nil {
        jsonResp := "{\"Error\":\"Failed to get state for " + A + "\"}"
        return shim.Error(jsonResp)
    }

    if Avalbytes == nil {
        jsonResp := "{\"Error\":\"Nil amount for " + A + "\"}"
        return shim.Error(jsonResp)
    }

    jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + string(Avalbytes) + "\"}"
    fmt.Printf("Query Response:%s\n", jsonResp)
    //Return the transfer amount.
    return shim.Success(Avalbytes)
}

func main() {
    err := shim.Start(new(SimpleChaincode))
    if err != nil {
        fmt.Printf("Error starting Simple chaincode: %s", err)
    }
}
```

2.3.4 Sample Chaincode (2.0)

The following is an example of installing and instantiating the account transfer chaincode (2.0). For details about how to debug this chaincode, see **the official Fabric examples**.

```
package main

import (
        "errors"
        "fmt"
        "strconv"

        "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// Chaincode implementation
type ABstore struct {
        contractapi.Contract
}

// Automatically invoked during chaincode instantiation or update to initialize the chaincode data.
func (t *ABstore) Init(ctx contractapi.TransactionContextInterface, A string, Aval int, B string, Bval int) error {
        // The output information of the println function is recorded in the logs of the chaincode container.
        fmt.Println("ABstore Init")
        var err error

        fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)
        // Write the status data to the ledger.
        err = ctx.GetStub().PutState(A, []byte(strconv.Itoa(Aval)))
        if err != nil {
```

```
                return err
        }

        err = ctx.GetStub().PutState(B, []byte(strconv.Itoa(Bval)))
        if err != nil {
                return err
        }

        return nil
}

// A transfers X to B.
func (t *ABstore) Invoke(ctx contractapi.TransactionContextInterface, A, B string, X int) error {
        var err error
        var Aval int
        var Bval int

        // Obtain the status data from the ledger.
        Avalbytes, err := ctx.GetStub().GetState(A)
        if err != nil {
                return fmt.Errorf("Failed to get state")
        }
        if Avalbytes == nil {
                return fmt.Errorf("Entity not found")
        }
        Aval, _ = strconv.Atoi(string(Avalbytes))

        Bvalbytes, err := ctx.GetStub().GetState(B)
        if err != nil {
                return fmt.Errorf("Failed to get state")
        }
        if Bvalbytes == nil {
                return fmt.Errorf("Entity not found")
        }
        Bval, _ = strconv.Atoi(string(Bvalbytes))

        // Transfer
        Aval = Aval - X
        Bval = Bval + X
        fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

        // Write the status data back to the ledger.
        err = ctx.GetStub().PutState(A, []byte(strconv.Itoa(Aval)))
        if err != nil {
                return err
        }

        err = ctx.GetStub().PutState(B, []byte(strconv.Itoa(Bval)))
        if err != nil {
                return err
        }

        return nil
}

// Account deregistration
func (t *ABstore) Delete(ctx contractapi.TransactionContextInterface, A string) error {

    // Delete the account status from the ledger.
        err := ctx.GetStub().DelState(A)
        if err != nil {
                return fmt.Errorf("Failed to delete state")
        }

        return nil
}

// Account query
func (t *ABstore) Query(ctx contractapi.TransactionContextInterface, A string) (string, error) {
```

```
        var err error
        // Obtain the status data from the ledger.
        Avalbytes, err := ctx.GetStub().GetState(A)
        if err != nil {
                jsonResp := "{\"Error\":\"Failed to get state for " + A + "\"}"
                return "", errors.New(jsonResp)
        }

        if Avalbytes == nil {
                jsonResp := "{\"Error\":\"Nil amount for " + A + "\"}"
                return "", errors.New(jsonResp)
        }

        jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + string(Avalbytes) + "\"}"
        fmt.Printf("Query Response:%s\n", jsonResp)
        return string(Avalbytes), nil
}

func main() {
        cc, err := contractapi.NewChaincode(new(ABstore))
        if err != nil {
                panic(err.Error())
        }
        if err := cc.Start(); err != nil {
                fmt.Printf("Error starting ABstore chaincode: %s", err)
        }
}
```

2.3.5 Chaincode Debugging

To debug a chaincode, you can use MockStub to perform unit tests on the chaincode. Before chaincode debugging, import the shim package to the test code. You can refer to the sample test code provided below or the **test code provided by Fabric**.

Writing Test Code

The test code of **Sample Chaincode (1.4)** is as follows:

```
package main

import (
    "fmt"
    "testing"

    "github.com/hyperledger/fabric/core/chaincode/shim"
)

func checkInit(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 checkState(t *testing.T, stub *shim.MockStub, name string, value string) {
    bytes := stub.State[name]
    if bytes == nil {
        fmt.Println("State", name, "failed to get value")
        t.FailNow()
    }
    if string(bytes) != value {
        fmt.Println("State value", name, "was not", value, "as expected")
        t.FailNow()
    }
}
```

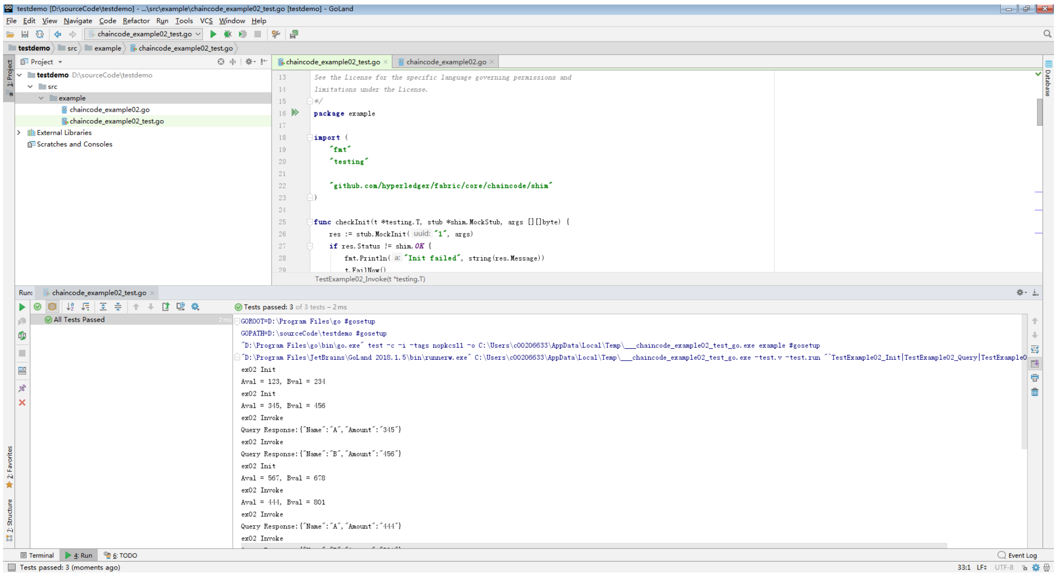


```
func checkQuery(t *testing.T, stub *shim.MockStub, name string, value string) {
    res := stub.MockInvoke("1", [][]byte{[]byte("query"), []byte(name)})
    if res.Status != shim.OK {
        fmt.Println("Query", name, "failed", string(res.Message))
        t.FailNow()
    }
    if res.Payload == nil {
        fmt.Println("Query", name, "failed to get value")
        t.FailNow()
    }
    if string(res.Payload) != value {
        fmt.Println("Query value", name, "was not", value, "as expected")
        t.FailNow()
    }
}

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

func TestExample02_Init(t *testing.T) {
    scc := new(SimpleChaincode)
    stub := shim.NewMockStub("ex02", scc)

    // Init A=123 B=234
    checkInit(t, stub, [][]byte{[]byte("init"), []byte("A"), []byte("123"), []byte("B"), []byte("234")})

    checkState(t, stub, "A", "123")
    checkState(t, stub, "B", "234")
}

func TestExample02_Query(t *testing.T) {
    scc := new(SimpleChaincode)
    stub := shim.NewMockStub("ex02", scc)

    // Init A=345 B=456
    checkInit(t, stub, [][]byte{[]byte("init"), []byte("A"), []byte("345"), []byte("B"), []byte("456")})

    // Query A
    checkQuery(t, stub, "A", "345")

    // Query B
    checkQuery(t, stub, "B", "456")
}

func TestExample02_Invoke(t *testing.T) {
    scc := new(SimpleChaincode)
    stub := shim.NewMockStub("ex02", scc)

    // Init A=567 B=678
    checkInit(t, stub, [][]byte{[]byte("init"), []byte("A"), []byte("567"), []byte("B"), []byte("678")})

    // Invoke A->B for 123
    checkInvoke(t, stub, [][]byte{[]byte("invoke"), []byte("A"), []byte("B"), []byte("123")})
    checkQuery(t, stub, "A", "444")
    checkQuery(t, stub, "B", "801")

    // Invoke B->A for 234
    checkInvoke(t, stub, [][]byte{[]byte("invoke"), []byte("B"), []byte("A"), []byte("234")})
    checkQuery(t, stub, "A", "678")
    checkQuery(t, stub, "B", "567")
    checkQuery(t, stub, "A", "678")
    checkQuery(t, stub, "B", "567")
}
```

Debugging

Execute the test function in the IDE.

Figure 2-1 Chaincode debugging



2.4 Java Chaincode Development

2.4.1 Chaincode Structure

This section uses the Java language as an example. A chaincode is a Java project. After creating the project, you can perform operations such as function development.

Notes and Constraints

The Java chaincode is supported only by Fabric v2.2 and later versions.

Chaincode Interface

A chaincode is invoked using the start function in the shim package. During chaincode development, define a class to extend ChaincodeBase. The following methods are overridden during the extension:

```
public class SimpleChaincodeSimple extends ChaincodeBase {
    @Override
    public Response init(ChaincodeStub stub) {
    }

    @Override
    public Response invoke(ChaincodeStub stub) {
    }
}
```

- **init** is called to initialize data during chaincode instantiation or update.
- **Invoke** is called to update or query the ledger. The service logic for responding to the call or query needs to be implemented in this method.

Chaincode Structure

The Java chaincode structure is as follows:

```
package main

//You only need to configure the required packages in Maven or Gradle. The packages are imported
automatically.
import org.hyperledger.fabric.shim.ChaincodeBase;
import org.hyperledger.fabric.shim.ChaincodeStub;

public class SimpleChaincodeSimple extends ChaincodeBase {
    @Override
    public Response init(ChaincodeStub stub) {
        //Implement the processing logic for chaincode initialization or update in this method.
        //stub APIs can be flexibly used during compilation.
    }

    @Override
    public Response invoke(ChaincodeStub stub) {
        //Implement the processing logic for responding to the call or query in this method.
        //stub APIs can be flexibly used during compilation.
    }

    //Main function. The shim.Start() method needs to be invoked.
    public static void main(String[] args) {
        new SimpleChaincode().start(args);
    }
}
```

2.4.2 Chaincode APIs

The shim package in the Fabric source code package provides the following types of APIs:

- Parameter parsing APIs: used to parse parameters transferred to the invoked function or method during chaincode invocation
- Ledger data operation APIs: used to provide methods for performing operations on ledger data, including status data query and transaction processing
- Transaction obtaining APIs: used to obtain information about transaction proposals
- Other APIs: used to set events and invoke other chaincodes

2.4.3 Sample Chaincode

The following is an example chaincode for reading and writing data. You can also refer to other chaincodes in the **official examples provided by Fabric**.

2.4.4 Chaincode Debugging

To debug a chaincode, you can use MockStub to perform unit tests on the chaincode. To obtain the chaincode used in this section, go to the BCS console and click **Use Cases**. Download **Chaincode_Java_Local_Demo** in the **Java SDK Demo** area.

Adding the Dependency

To use the mock() method, add the Mockito dependency.

- Gradle

  Add the following dependency to the **dependencies** block in the **build.gradle** file (not **dependencies** in the **buildscript** block):

  ```
  testCompile 'org.mockito:mockito-core:2.4.1'
  ```

- Maven

  Add the following configuration dependency to the **dependencies** block (add the block if it does not exist) in the **pom.xml** file:

  ```
  <dependency>
     <groupId>org.mockito</groupId>
     <artifactId>mockito-core</artifactId>
     <version>2.4.1</version>
  </dependency>
  ```

Writing Test Code

If the **test** folder does not exist during project creation, create it under **src**. Select **test\java** under **Gradle Source Sets**, and create the **SimpleChaincodeTest.java** test file, as shown in the following figures:

Figure 2-2 Creating a test file

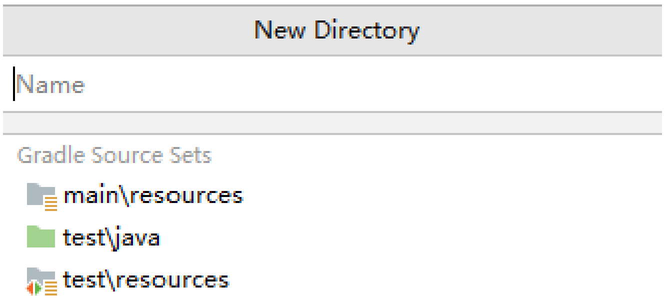
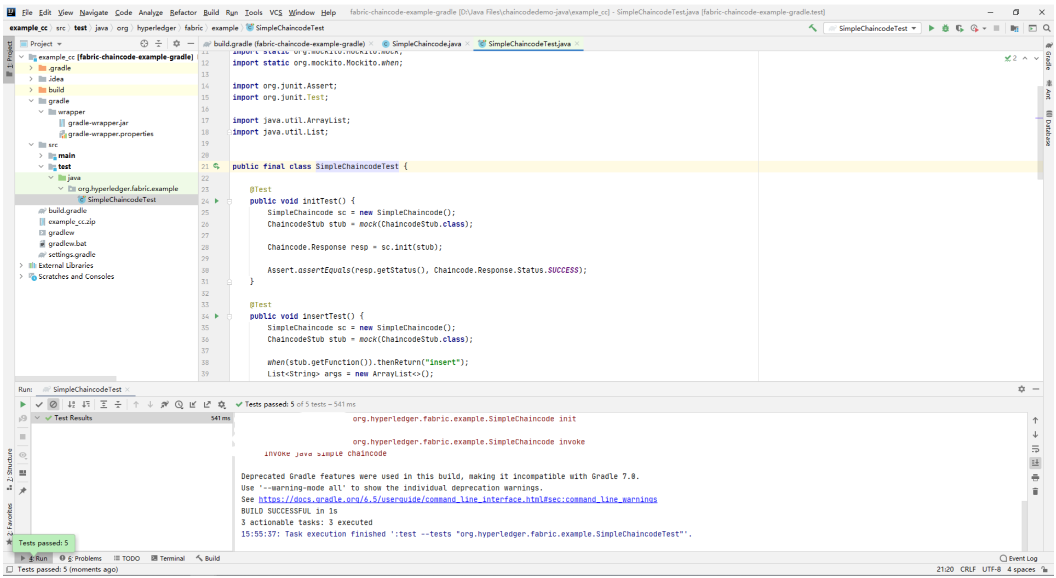


Figure 2-3 Test file



The content of the **SimpleChaincodeTest.java** test code is as follows:

```
import org.hyperledger.fabric.example.SimpleChaincode;
import org.hyperledger.fabric.shim.Chaincode;
```

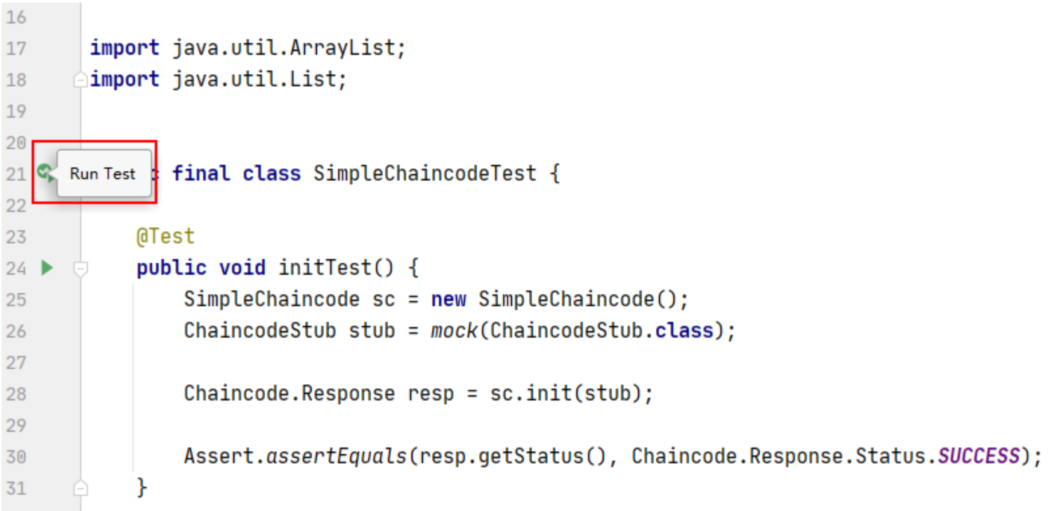
```
import org.hyperledger.fabric.shim.ChaincodeStub;
import org.junit.Assert;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

public final class SimpleChaincodeTest {

    @Test
    public void initTest() {
        SimpleChaincode sc = new SimpleChaincode();
        ChaincodeStub stub = mock(ChaincodeStub.class);
        Chaincode.Response resp = sc.init(stub);
        Assert.assertEquals(resp.getStatus(), Chaincode.Response.Status.SUCCESS);
    }

    @Test
    public void insertTest() {
        SimpleChaincode sc = new SimpleChaincode();
        ChaincodeStub stub = mock(ChaincodeStub.class);
        when(stub.getFunction()).thenReturn("insert");
        List<String> args = new ArrayList<>();
        args.add("a");
        args.add("100");
        when(stub.getParameters()).thenReturn(args);
        Chaincode.Response resp = sc.invoke(stub);
        Assert.assertEquals(resp.getStatus(), Chaincode.Response.Status.SUCCESS);
    }

    @Test
    public void insertTooManyArgsTest() {
        SimpleChaincode sc = new SimpleChaincode();
        ChaincodeStub stub = mock(ChaincodeStub.class);
        when(stub.getFunction()).thenReturn("insert");
        List<String> args = new ArrayList<>();
        args.add("a");
        args.add("100");
        args.add("b");
        args.add("100");
        when(stub.getParameters()).thenReturn(args);
        Chaincode.Response resp = sc.invoke(stub);
        Assert.assertEquals(resp.getMessage(), "Incorrect number of arguments. Expecting 2");
    }

    @Test
    public void queryTest() {
        SimpleChaincode sc = new SimpleChaincode();
        ChaincodeStub stub = mock(ChaincodeStub.class);
        when(stub.getFunction()).thenReturn("query");
        List<String> args = new ArrayList<>();
        args.add("a");
        when(stub.getParameters()).thenReturn(args);
        when(stub.getStringState("a")).thenReturn("100");
        Chaincode.Response resp = sc.invoke(stub);
        Assert.assertEquals(resp.getMessage(), "100");
    }

    @Test
    public void queryNoExistTest() {
        SimpleChaincode sc = new SimpleChaincode();
        ChaincodeStub stub = mock(ChaincodeStub.class);
        when(stub.getFunction()).thenReturn("query");
        List<String> args = new ArrayList<>();
        args.add("a");
        when(stub.getParameters()).thenReturn(args);
```

```
        when(stub.getStringState("a")).thenReturn(null);
        Chaincode.Response resp = sc.invoke(stub);
        Assert.assertEquals(resp.getMessage(), "{\"Error\":\"Null val for a\"}");
    }
}
```

Debugging

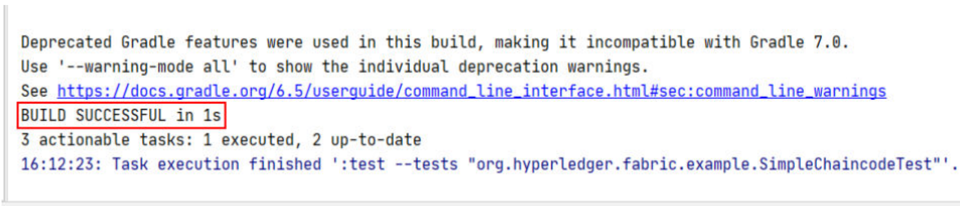
In **SimpleChaincodeTest.java**, click **Run Test** on the left of **SimpleChaincodeTest**.

Figure 2-4 Executing the test



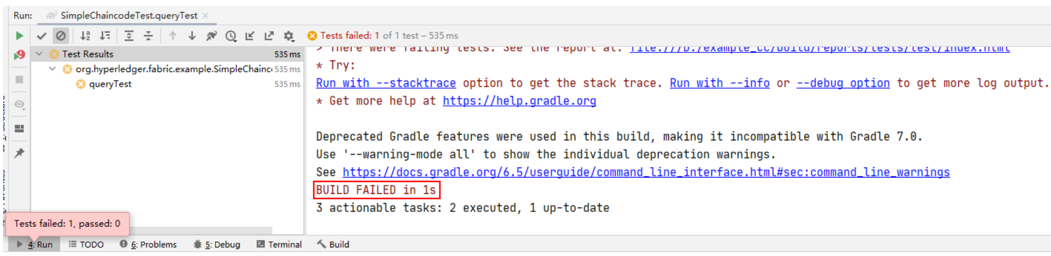
If the following information is displayed, the chaincode debugging is successful:

Figure 2-5 Successful



If the following information is displayed, the chaincode debugging failed. Edit the chaincode or check the logic of the test code based on the displayed information.

Figure 2-6 Failed




```
expected:<[{"Error":"Null val for a"}]> but was:<[100]>
Expected :{"Error":"Null val for a"}
Actual   :100
<Click to see difference>
```

3 Application Development

3.1 Overview

User applications interact with the ledger using chaincodes. An application can be written in a variety of languages, including Go, Solidity, Java, C++, Python and Node.js. The languages used by the application and the chaincode do not necessarily have to be the same, as long as the application can invoke the chaincode using the SDK.

NOTE

Enhanced Hyperledger Fabric blockchains open gRPC APIs to applications, just like the open-source version. These APIs are usually called by SDKs. For details, see **the definition of the SDK APIs**.

3.2 Preparations

User applications interact with the ledger using chaincodes. An application can be written in a variety of languages, including Go, Solidity, Java, C++, Python and Node.js. The languages used by the application and the chaincode do not necessarily have to be the same, as long as the application can invoke the chaincode using the SDK.

1. Create a BCS instance.

   BCS instances can be deployed in CCE clusters. For details, see **Using a CCE Cluster**.

2. Obtain the required SDK configuration file and certificates. For details, see **BCS Access**.

3.3 Development

Develop your own application code. You can use the **SDK** provided by BCS or the SDK provided by the official Fabric community that matches the version of your instance.

NOTE

The version of the Fabric package should match that of the blockchain. For example, if a blockchain is v4.x.x, it uses Fabric v2.2, so you need to download the Fabric v2.2 package.

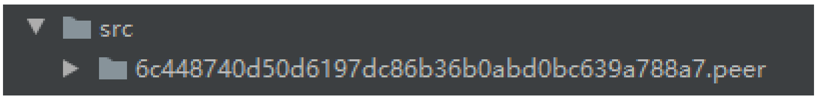
Configuring the Organization ID

Modify the application code for configuring the organization ID of the BCS instance. After the downloaded certificate file is decompressed, the peer file name contains the directory name and the organization ID.

The following figure is for reference only. Use the actual certificate file.

After the certificate file is decompressed, the directory name is **6c448740d50d6197dc86b36b0abd0bc639a788a7.peer** and the organization ID is **6c448740d50d6197dc86b36b0abd0bc639a788a7**.

Figure 3-1 Decompressing the certificate file



Configuring the SDK file

1. Modify the application code related to the SDK configuration file. As shown in the following example, you must configure the correct absolute path of the SDK configuration file.

```
var (
   configFile = "/root/gosdkdemo/config/go-sdk-demo-channel-sdk-config.yaml"
   org = " 6c448740d50d6197dc86b36b0abd0bc639a788a7"
)
```

2. If the paths in the SDK configuration file are different from the actual ones, you must manually change all certificate paths in the SDK configuration file.

4 Demos

4.1 Go SDK Demo

This section provides a Go SDK–based demo to help you develop your own Go client applications.

Preparations

- Prepare an ECS.
- Install the golang environment on the ECS. The Go version must be 1.12 or later, and earlier than 1.16.
- Obtain the Go SDK source code. To obtain it, log in to the BCS console, choose **Interactive Walkthroughs** > **Use Cases** and download the source code in the **Go SDK Demo** area.

Creating a BCS Instance

For details, see **Using a CCE Cluster**.

Installing and Instantiating a Chaincode

To obtain the chaincode used in this demo, go to the BCS console and choose **Interactive Walkthroughs** > **Use Cases** in the navigation pane. Download the example Go chaincode in the **Go SDK Demo** area.

For details, see **User Guide > Blockchain Management > Chaincode Management**.

Downloading SDK Configurations and Certificates

Step 1 Log in to the BCS console.

Step 2 On the **Instance Management** page, click **Download Client Configuration** on an instance card.

Step 3 Select **SDK Configuration File** and set the parameters as described in the following table.

Parameter	Setting
Chaincode Name	Enter **chaincode**. **NOTE** The chaincode name must be the same as the name specified during chaincode installation and instantiation.
Certificate Root Path	Enter **/root/gosdkdemo/config**.
Channel	Select **channel**.
Organization & Peer	Retain the default value.

Select **Orderer Certificate**.

Select **Peer Certificates**, select **organization** for **Peer Organization**, and select **Administrator certificate**.

Step 4 Click **Download**. The SDK configuration file and the administrator certificates for the **orderer** and **organization** organizations are downloaded.

----End

Deploying the Application

1. Download the Go SDK source code to the **/root** directory of the ECS and decompress the package.

   To obtain it, go to the BCS console and click **Use Cases**. Download the source code in the **Go SDK Demo** area.

2. Decompress the .zip package obtained in **Downloading SDK Configurations and Certificates** and copy the **orderer** and **peer** folders and the **sdk-config.json** and **sdk-config.yaml** files from the **configs** folder to the **/root/gosdkdemo/config/** directory.

3. Find the **/gosdkdemo/src/main.go** file in the code and modify it as follows:

   a. Change the value of **configFile** to the actual name of the SDK configuration file, for example, **demo-channel-sdk-config.yaml**.

   b. Change the value of **org** to the hash value of **organization**.

      On the **Channel Management** page, click **View Peer**. The organization ID is the value of **MSP ID** without "MSP".

```
var (
    configFile = "/root/gosdkdemo/config/go-sdk-demo-channel-sdk-config.yaml"
    org = " 9103f17cb6b4f69d75982eb48bececcc51aa3125"
 )
```

4. Use **go.mod** to set GOPATH based on the actual installation path.

   a. Set the environment variable **GO111MODULE** to **on**.

```
export GO111MODULE=on
```

   b. The following figure shows the **go.mod** file. Modify the "replace" line based on the actual installation path.

```
module main
go 1.15
//Specify the dependency to be imported and its version.
```

```
require (
        github.com/bitly/go-simplejson v0.5.0
        github.com/bmizerany/assert v0.0.0-20160611221934-b7ed37b82869// indirect
        github.com/ghodss/yaml v1.0.0
        github.com/hyperledger/fabric-sdk-go v1.0.0
        github.com/pkg/errors v0.9.1
        github.com/spf13/viper v1.7.1
)
//The project path /root/gosdkdemo/src is used as an example.
replace github.com/hyperledger/fabric-sdk-go => /root/gosdkdemo/src/github.com/hyperledger/
fabric-sdk-go
```

5. Find the **main.go** file in the **gosdkdemo/src** directory and run the following command:

```
go run main.go
```

Common APIs

fabric-sdk-go mainly uses the FabricSDK class, which can be constructed by using the NewSDK() method.

FabricClient, ChannelClient, ChannelMgmtClient, and ResourceMgmtClient can perform common operations of **fabric-sdk-go**.

- **FabricSDK**

  FabricSDK uses the New() method to generate objects in **pkg\fabsdk\fabsdk.go**. The New() method has an **Options** parameter. The following is an example of generating FabricSDK:

  var opts []fabsdk.Option

  opts = append(opts, fabsdk.WithOrgid(org))

  opts = append(opts, fabsdk.WithUserName("Admin"))

  sdk, err = fabsdk.New(config.FromFile(configFile), opts...)

  configFile indicates the configuration file path. **OrgId** is the organization ID in the SDK configuration file.

  FabricSDK uses the NewSDK() method to generate objects in **def/fabapi/fabapi.go**. The NewSDK() method has an Options parameter. The following is an example of generating the **Options** parameter:

  ```
  deffab.Options{ConfigFile: configFile, LoggerFa
  logging.LoggerProvider(), UserName: sysadmin}
  ```

  ConfigFile indicates the configuration file path. **LoggerFactory** is optional. If it is not specified, logs are printed to the console by default.

- **FabricClient**

  FabricClient provides the following common APIs:

API	Description	Setting	Returned Values
CreateChannel	Creates a channel.	request CreateChannelRequest	txn.TransactionID, error

API	Description	Setting	Returned Values
QueryChannelInfo	Queries channel information.	name string, peers []Peer	Channel, error
InstallChaincode	Installs chaincodes in a blockchain.	request InstallChaincodeRequest	[]*txn.TransactionProposlResponse, string, error
InstallChaincode	Installs chaincodes in a blockchain.	request InstallChaincodeRequest	[]*txn.TransactionProposlResponse, string, error
QueryChannels	Queries created channels in a blockchain.	peer Peer	*pb.ChannelQueryResponse, error
QueryInstalledChaincodes	Queries installed chaincodes in a blockchain.	peer Peer	*pb.ChaincodeQueryResponse, error

- **ChannelClient**

  ChannelClient provides chaincode query and invoking APIs.

API	Description	Setting	Returned Values
Query	Queries data by using the chaincode.	request QueryRequest	[]byte, error
QueryWithOpts	Similar to the **Query** API, but can specify notifier, peers, and timeout with **QueryOpts**.	request QueryRequest, opt QueryOpts	[]byte, error
ExecuteTx	Invokes the chaincode.	request ExecuteTxRequest	TransactionID, error

API	Description	Setting	Returned Values
ExecuteTxWithOpts	Similar to the **ExecuteTx** API, but can specify notifier, peers, and timeout with the **ExecuteTxOpts** parameter.	request ExecuteTxRequest opt ExecuteTxOpts	TransactionID, error

- **ChannelMgmtCLient**

  ChannelMgmtClient provides only two APIs: **SaveChannel(req SaveChannelRequest) error** and **SaveChannelWithOpts(req SaveChannelRequest, opts SaveChannelOpts) error**, which are used to create channels and need to invoke the **createChannel()** API of FabricClient.

- **ResourceMgmtClient**

  ResourceMgmtClient provides chaincode lifecycle management APIs and an API for adding peers to a channel.

  NOTE

  The chaincode deleting API is provided by BCS and can only delete the chaincode installation package.

API	Description	Setting	Returned Values
InstallCC	Installs chaincodes.	reqInstallCCRequest	[]InstallCCResponse, error
InstallCCWithOpts	Similar to the **InstallCC** API, but can specify peers with **InstallCCOpts**.	reqInstallCCRequest,opts InstallCCOpts	[]InstallCCResponse, error
InstantiateCC	Instantiates chaincodes.	channelID string,reqInstantiateCCRequest	error
InstantiateCCWithOpts	Similar to the **InstantiateCC** API, but can specify peers and timeout with **InstantiateCCOpts**.	channelID string,reqInstantiateCCRequest, optsInstantiateCCOpts	error
UpgradeCC	Upgrades chaincodes.	channelID string,reqUpgradeCCRequest	error
UpgradeCCWithOpts	Similar to the **UpgradeCC** API, but can specify peers and timeout with **UpgradeCCOpts**.	channelID string,reqUpgradeCCRequest, optsUpgradeCCOpts	error

API	Description	Setting	Returned Values
DeleteCC	Deletes chaincodes (only the chaincode installation packages).	channelID string,reqDeleteCCRequest	error
DeleteCCWithOpts	Similar to the **DeleteCC** API, but can specify peers and timeout with **DeleteCCWithOpts**.	channelID string,reqDeleteCCRequest,opts DeleteCCOpts	error
JoinChannel	Adds peers to a channel.	channelID string	error
JoinChannelWithOpts	Similar to the **JoinChannel** API, but can specify peers and timeout with **JoinChannelOpts**.	channelID string,optsJoinChannelOpts	error

NOTE

All APIs with options can specify peers. The peers can be generated through NewPeer(userName string, orgName string, url string, certificate string, serverHostOverride string, config config.Config) (fab.Peer, error) in **def/fabapi/pkgfactory.go**. Compared with the native NewPeer method, this method has two more parameters: **userName** and **orgName**, which are used by the peers to find the corresponding TLS certificate with bidirectional TLS.

Invoking a Contract

Main.go is a simple client application sample program, which is used to help you quickly get started with the client development process. The main steps are as follows:

```
//1. Import packages. The SDK package provides some APIs for user applications to access chaincodes.
import (
   "fmt"
   "github.com/hyperledger/fabric-sdk-go/pkg/client/channel"
   "github.com/hyperledger/fabric-sdk-go/pkg/fabsdk"    ......
)
//2. Create file configurations. This step encapsulates some common configurations required for application development, including the SDK configuration file path and organization name.
var (
   configFile = "/root/fabric-go-demo/config/go-sdk-demo-channel-sdk-config.yaml"
   org = "9103f17cb6b4f69d75982eb48bececcc51aa3125"
   ......
)
//3. Load the configuration file.
loadConfig()
//4. Initialize the SDK.
initializeSdk()
// 5. Execute the chaincode and write the data to the ledger. The key is "testuser", and the value is "100".
insert("insert",[][]byte{
   []byte("testuser"),
   []byte("100"),
```

```
})
// 6. Query the chaincode and output the query result. The key is "testuser".
query("query",
[][]byte{
[]byte("testuser"),
})
```

Table 4-1 Functions

Function	Description
getOptsToInitializeSDK	Parses the configuration file, and creates and returns the fabsdk.Option object.
GetDefaultChaincodeId	Parses the configuration file and returns **chaincodeID**.
GetDefaultChannel	Parses the configuration file and returns **channelID**.
UserIdentityWithOrgAndName	Verifies the identity of a user. The input parameters are the organization name and username. The verification result is returned.
ChannelClient	Create the *channel.Client object. The input parameters are the organization name, username, and channel ID. The *channel.Client object is returned.
insert	Writes data to the ledger. The input parameters are the method name of the chaincode and the key-value pair to be inserted. The write result is returned.
query	Queries chain data. The input parameters are the method name of the chaincode and the data to be queried. The query result is returned.

4.2 Java SDK Demo

This section provides a demo application that uses a Java SDK and supports the OSCCA-published cryptographic algorithms to help you quickly understand the concepts and process of using BCS.

NOTE

This is a demo only and is not for actual use.

Preparations

Action	Description
Install an integrated development environment (IDE).	Install Java Development Kit (JDK), Maven, and Eclipse. You can replace Eclipse with another IDE you prefer. The JDK version must be 1.8 (64-bit). If you have installed JDK, run the **java -version** command in the command line to check the JDK version.

Creating a BCS Instance

Step 1 Log in to the BCS console.

Step 2 Click **Create BCS Instance** in the upper right corner of the page.

Step 3 Configure basic information about the BCS instance by referring to **Table 4-2**.

NOTICE

To ensure that the demo runs properly, set the parameters as described in the following table.

Table 4-2 Basic settings

Parameter	Setting
Region	Retain the default value.
Enterprise Project	Select an existing enterprise project, for example, **default**. If the enterprise management service is not enabled, this parameter is unavailable.
Instance Name	Enter **java-sdk-demo**.
Edition	Select **Professional**. If OSCCA-published cryptographic algorithms must be used, select **Enterprise**.
Blockchain Type	Select **Private**.
Enhanced Hyperledger Fabric Version	v2.2
Consensus Mechanism	Select **Raft (CFT)**.
Resource Access Initial Password	Enter a password.
Confirm Password	-

Step 4 Click **Next: Configure Resources**. **Table 4-3** describes the resource parameters.

Table 4-3 Resource configurations

Parameter	Example
Environment Resources	Select **Custom**.
Cluster	Select **Create a new CCE cluster**.
AZ	Select an AZ.

Parameter	Example
Cross-AZ Scheduling	Select **No**.
Use EIP of a CCE Node	Select **Yes**.

Step 5 Click **Next: Configure Blockchain**. Table 4-4 describes the blockchain parameters.

Table 4-4 Blockchain configurations

Parameter	Example
Blockchain Configuration	Select **Custom**.
Blockchain Mgmt. Initial Password	If you do not enter a password here, the previously specified resource access initial password will be used.
Confirm Password	-
Volume Type	Select **SFS Turbo** or select another one as prompted.
Storage Capacity of Peer Organization (GB)	Retain the default value.
Ledger Storage	Select **File database (GoLevelDB)**.
Peer Organization	A peer organization named **organization** has been automatically created. Change the peer quantity to 1.
Channel Configuration	The **organization** organization has been added to the channel automatically. Retain this default setting.
Orderer Quantity	Retain the default value.
Security Mechanism	Select **ECDSA**. **NOTICE** The **OSCCA-published cryptographic algorithms** option is available. If you select this option, other modifications are required for the demo deployment. Pay attention to the descriptions of modifications.
Configure Block Generation	Select **No**.
Enable Support for RESTful API	Select **No**.

Step 6 Click **Next: Confirm**.

Step 7 Confirm the configurations and finish the creation process.

Wait for several minutes. After a message is displayed indicating successful installation, check the status of the instance. If it is **Normal**, the deployment is completed.

----End

Installing and Instantiating a Chaincode

Step 1 Log in to the BCS console.

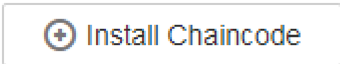
Step 2 In the navigation pane on the left, click **Instance Management**.

Step 3 Find the instance you just created and click **Manage Blockchain** to go to the Blockchain Management console.

Step 4 On the login page, enter the username and password, and click **Log In**.

NOTE

The username is **admin**, and the password is the **Blockchain Mgmt. Initial Password** set when you created the BCS instance. If you have not set this password, use the resource access initial password.

Step 5 Click 

in the upper left corner of the page.

The parameters for chaincode installation are as follows.

Table 4-5 Installation parameters

Parameter	Setting
Chaincode Name	Enter **chaincode**.
Chaincode Version	Select **2.0**.
Ledger Storage	**File database (goleveldb)**
Select All Peers	Check the box.
Organization & Peer	Select **peer-0**.
Language	Select **Golang**.
Chaincode File	Chaincode_Go_Demo: To obtain it, go to the BCS console and click **Use Cases**. Download the example Go chaincode in the **Java SDK Demo** area.
Chaincode Description	Enter a description of the chaincode.
Code Security Check	This option is displayed only when the chaincode language is Golang. Enable this option to check chaincode security.

Step 6 Click **Install**.

Step 7 After installing the chaincode, click **Instantiate** in the **Operation** column of the chaincode list.

The parameters for chaincode instantiation are as follows.

Table 4-6 Instantiation parameters

Parameter	Setting
Chaincode Name	Enter **chaincode**.
Channel	Select **channel**.
Chaincode Version	Select **2.0**.
Initialization Function	Enter **init**.
Chaincode Parameters	a,200,b,250
Endorsement Policy	Select **Endorsement from any of the following organizations**.
Endorsing Organizations	Select **organization**.
Privacy Protection Configuration	Select **No**.

Step 8 Click **Instantiate**.

Wait for 2 to 3 minutes and refresh the page. Click **View more** in the **Instantiation** column to check the instantiation status.

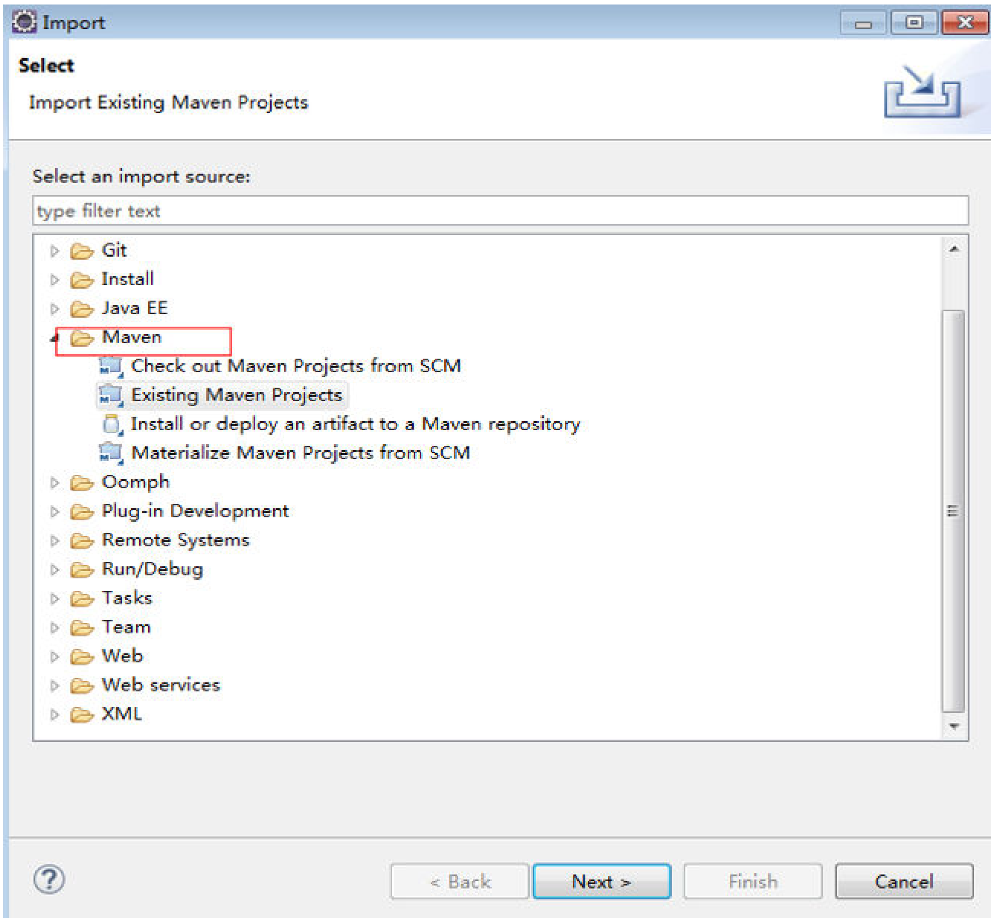
----End

Configuring the Application

Step 1 Import the project.

Obtain the project code and decompress it. To obtain the project code, go to the BCS console and choose **Use Cases**. Download the example Java chaincode in the **Java SDK Demo** area.

Right-click the Eclipse page, and choose **Import** from the shortcut menu to import the project code file (Maven project) to Eclipse.



Step 2 Download SDK configurations and certificates.

1. On the **Instance Management** page, click **Download Client Configuration** on an instance card.
2. Select **SDK Configuration File** and set the parameters as described in the following table.

Table 4-7 SDK parameters

Parameter	Description
Chaincode Name	Enter **chaincode**. **NOTICE** The chaincode name must be the same as the name specified during chaincode installation and instantiation.
Certificate Root Path	Enter the path to the **config** folder of the **javasdkdemo** project. **NOTICE** Change the backslashes (\) to slashes (/), for example, **D:/javasdkdemo/config**.
Channel	Select **channel**.
Organization & Peer	Select all peers in the channel.

Select **Orderer Certificate**.

Select **Peer Certificates**, select **organization** for **Peer Organization**, and select **Administrator certificate**.

3. Click **Download** to download the SDK configuration file and the administrator certificates for the **java-sdk-demo-orderer** and **organization** organizations to the **config** directory in the demo project.

Step 3 Copy and decompress the package.

Step 4 Decompress **demo-config.zip** and copy the contents in the **java-sdk-demo-orderer-admin-cert**, **organization-admin-cert**, and **sdk-config** folders to the **config** directory in the demo project.

----End

Deploying the Application

Step 1 In the Maven project, find the **Main.java** file in the **/javasdkdemo/src/main/java/handler/** directory, and change the file path in the following code of the Main class to the absolute path of the **java-sdk-demo-sdk-config.yaml** file. The path can be found in . Change the backslashes (\) to slashes (/).

```
helper.setConfigCtx("E:/yourdir/.yaml");
```

For example, change the path to **helper.setConfigCtx("D:/javasdkdemo/config/java-sdk-demo-channel-sdk-config.yaml")**.

NOTICE

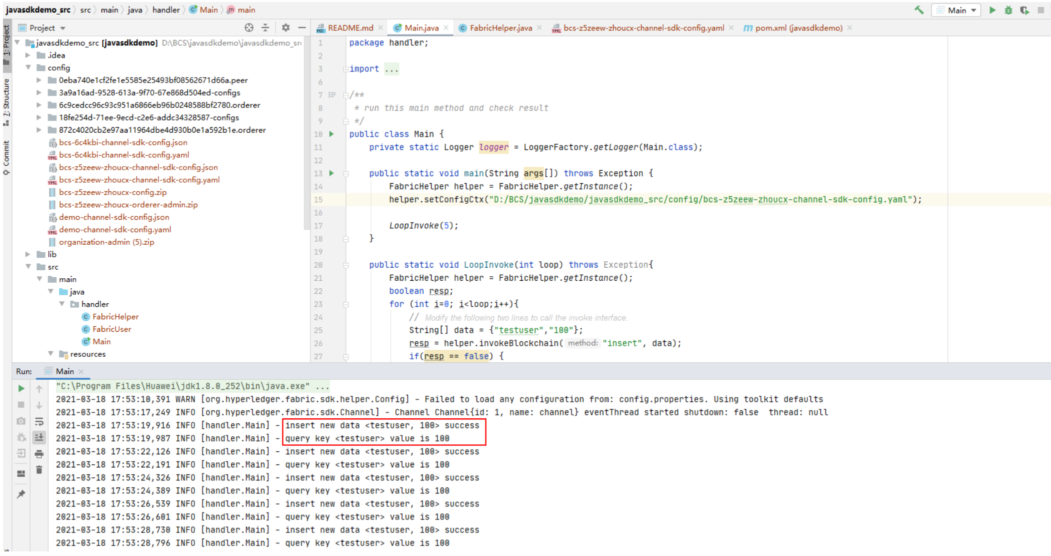
Add the dependency upon **fabric-sdk-java-1.4.1-jar-with-dependencies.jar** in the **lib** folder of the project to **pom.xml** because OSCCA-published cryptographic algorithms and other cryptographic algorithms need to refer to the **fabric-sdk** dependency package. To add the dependency, remove the comments on the dependency as shown in the following figure. Otherwise, the project may not run properly.

Figure 4-1 File details

Step 2 Run the main function.

Each time the command is successfully executed, the key-value pair <**testuser, 100**> is saved to the blockchain. If you query key **testuser**, the value is **100**. You can also view the transaction records in **Block Browser**.

Figure 4-2 Transaction records



----End

4.3 Gateway Java Demo

This section provides a demo based on Fabric Gateway for Java. Fabric Gateway Java encapsulates the Java SDK, which reduces the code amount and helps users develop Java client applications.

Common APIs

When you use Fabric-Gateway-Java to initiate transactions and query data, the Network and Contract interfaces are used. For more interfaces, see the **Fabric official website**.

- **Network**

  The common interfaces are as follows:

API	Description	Setting	Returned Values
getContract	Gets an instance of a contract.	String chaincodeId	Contract
addBlockListener	Adds a listener to listen to block events.	Consumer<org.hyperledger.fabric.sdk.BlockEvent> listener	Consumer<org.hyperledger.fabric.sdk.BlockEvent>
getChannel	Gets the channel associated with the network.	/	org.hyperledger.fabric.sdk.Channel

API	Description	Setting	Returned Values
removeBlockListener	Removes a listener.	Consumer<org.hyperledger.fabric.sdk.BlockEvent> listener	void

- **Contract**
  The common interfaces are as follows:

API	Description	Setting	Returned Values
submitTransaction	Submits a transaction. The invocation method and parameters need to be entered.	String name, String... args	byte[]
evaluateTransaction	Evaluates a transaction. The invocation method and parameters need to be entered.	String name, String... args	byte[]
createTransaction	Creates a transaction. The transaction needs to be submitted.	String name	Transaction
addContractListener	Adds a listener to listen to events emitted by committed transactions.	Consumer<ContractEvent> listener	Consumer<ContractEvent> listener
removeContractListener	Removes a listener.	Consumer<ContractEvent> listener	void

4.4 RESTful API Demo

BCS provides RESTful APIs to simplify the usage of blockchains. You only need to develop applications that support RESTful APIs to access blockchains without the need to learn Hyperledger Fabric SDKs for Golang, Java, and Node.js. This demo uses a Golang client to show how RESTful APIs are used to invoke a chaincode.

NOTE

This is a demo only and is not for actual use.

Creating a BCS Instance

Step 1 Log in to the BCS console.

Step 2 Click **Create BCS Instance** in the upper right corner of the page.

Step 3 Configure basic information about the BCS instance by referring to **Table 4-8**.

NOTICE

To ensure that the demo runs properly, set the parameters as described in the following table.

Table 4-8 Basic settings

Parameter	Setting
Region	Retain the default value.
Enterprise Project	Select **default**.
Instance Name	Enter **demo**.
Edition	Professional
Blockchain Type	Select **Private**.
Enhanced Hyperledger Fabric Version	v2.2
Consensus Mechanism	Raft(CFT)
Resource Access Initial Password	Enter a password.
Confirm Password	Confirm the password.

Step 4 Click **Next: Configure Resources**. **Table 4-9** describes the resource parameters.

Table 4-9 Resource configurations

Parameter	Example
Environment Resources	Select **Custom**.
Cluster	Select **Create a new CCE cluster**.
AZ	Select an AZ.
ECS Specifications	Select the flavor for **4 vCPUs | 8 GB**.

Parameter	Example
ECS Quantity	Enter **1**.
High Availability	Select **No**.
VPC	Select **Automatically create VPC**.
Subnet	Select **Automatically create subnet**.
ECS Login Method	Select **Password**.
Password of Root User	If you do not enter a password here, the previously specified resource access initial password will be used.
Confirm Password	-
Use EIP of a CCE Node	Select **Yes**.
EIP Billed By	Retain the default value.
EIP Bandwidth	Set it to 5 Mbit/s.

Step 5 Click **Next: Configure Blockchain**. Table 4-10 describes the blockchain parameters.

Table 4-10 Blockchain configurations

Parameter	Example
Blockchain Configuration	Select **Custom**.
Blockchain Mgmt. Initial Password	If you do not enter a password here, the previously specified resource access initial password will be used.
Confirm Password	-
Volume Type	Select **SFS Turbo**.
Storage Capacity of Peer Organization (GB)	Retain the default value.
Ledger Storage	Select **File database (GoLevelDB)**.
Peer Organization	A peer organization named **organization** has been automatically created. Change the peer quantity to 1.
Channel Configuration	The **organization** organization has been added to the channel automatically. Retain this default setting.
Orderer Quantity	Retain the default value.

Parameter	Example
Security Mechanism	Select **ECDSA**. **NOTICE** Only **ECDSA** can be selected.
Configure Block Generation	Select **No**.
Enable Support for RESTful API	Select **Yes**. If you select **No**, you can enable support for RESTful APIs later by performing the following steps: 1. In the navigation pane on the left, choose **Add-on Management**. 2. On the **Add-on Repository** tab page, hover the mouse pointer over the **baas-restapi** card. 3. Click **Install** and select the created BCS instance.

Step 6 Click **Next: Confirm**.

Step 7 Confirm the configurations and finish the creation process.

Wait for several minutes. After a message is displayed indicating successful installation, check the status of the instance. If it is **Normal**, the deployment is completed.

----End

Installing and Instantiating a Chaincode

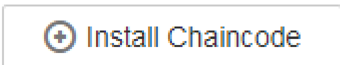
Step 1 Log in to the BCS console.

Step 2 Find the instance you just created and click **Manage Blockchain** to go to the Blockchain Management console.

Step 3 On the login page, enter the username and password, and click **Log In**.

NOTE

The username is **admin**, and the password is the **Blockchain Mgmt. Initial Password** set when you created the BCS instance. If you have not set this password, use the resource access initial password.

Step 4 Click 

in the upper left corner of the page.

The parameters for chaincode installation are as follows.

Parameter	Setting
Chaincode Name	Enter **bcsysq**.
Chaincode Version	Enter **1.0**
Ledger Storage	File database (goleveldb)

Parameter	Setting
Select All Peers	Check the box.
Organization & Peer	Select **peer-0**.
Language	Select **Golang**.
Chaincode File	Add the downloaded chaincode file **chaincode_example02.zip**.
Chaincode Description	Enter a description of the chaincode.
Code Security Check	This option is displayed only when the chaincode language is Golang. Enable this option to check chaincode security.

Step 5 Click **Install**.

Step 6 After installing the chaincode, click **Instantiate** in the **Operation** column of the chaincode list.

The parameters for chaincode instantiation are as follows.

Parameter	Setting
Channel	Select **channel**.
Chaincode Version	Enter **1.0**
Initialization Function	Enter **init**.
Chaincode Parameters	Enter **a,200,b,250**.
Endorsement Policy	Select **Endorsement from any of the following organizations**.
Endorsing Organizations	Select **organization**.
Privacy Protection Configuration	Select **No**.

----End

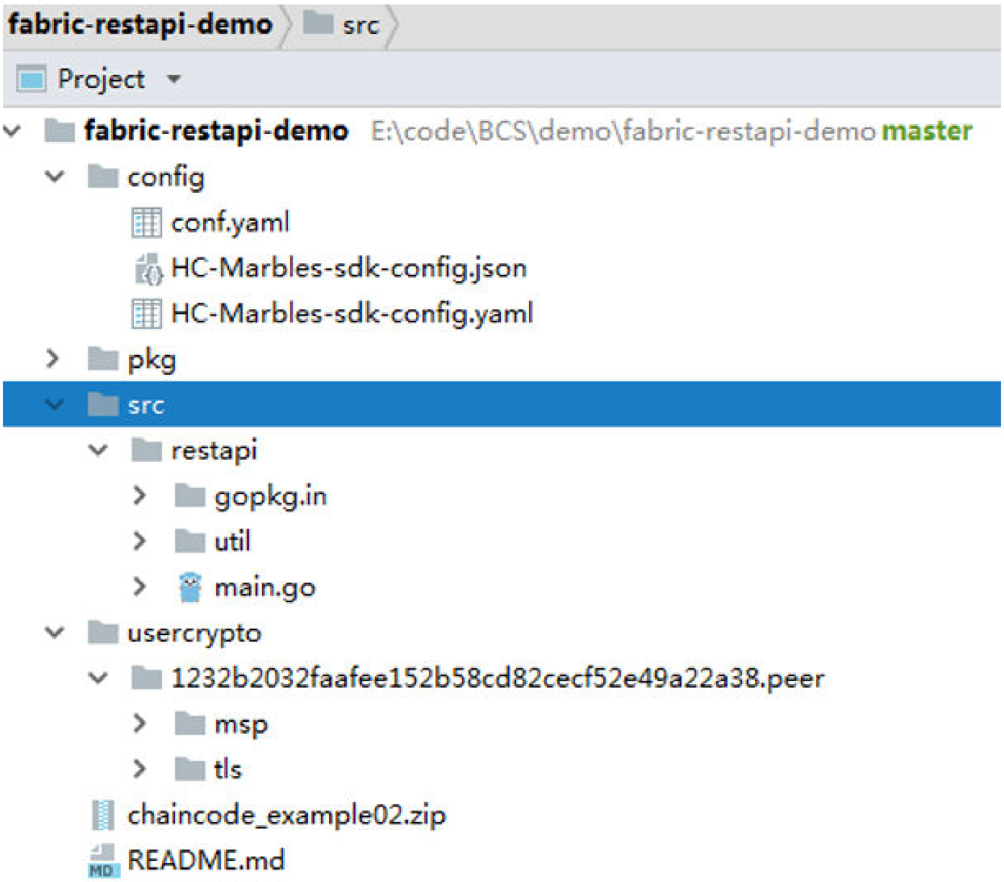
Configuring the Application

Step 1 On the **Instance Management** page, click **Download Client Configuration** on an instance card.

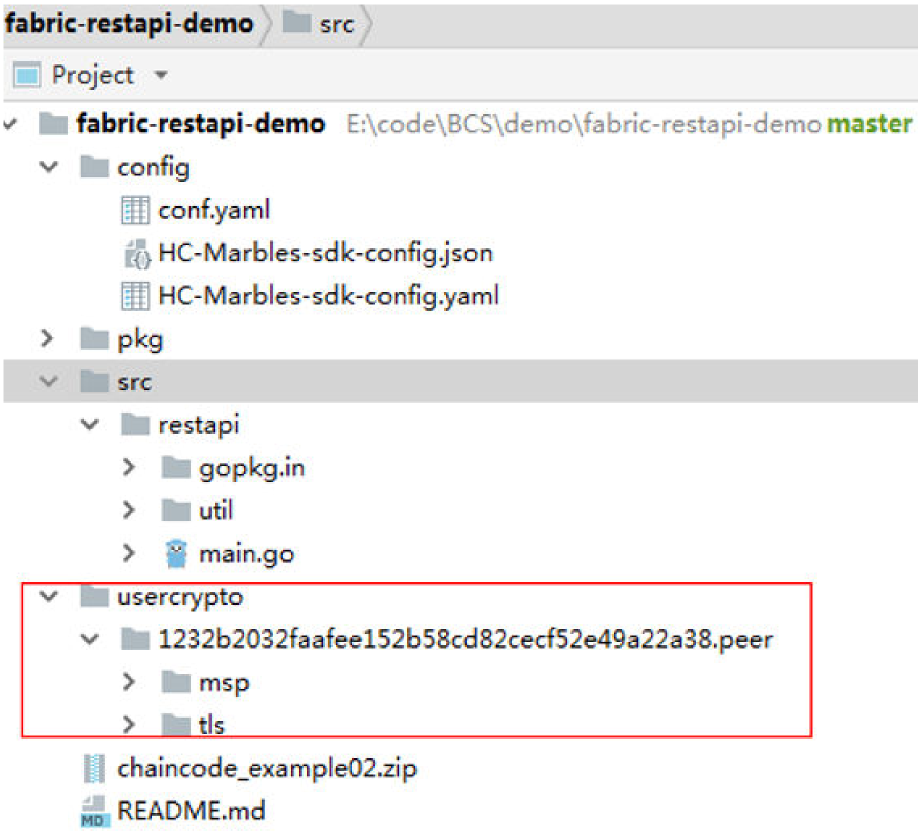
Step 2 Select **Peer Certificates**, select **organization** for **Peer Organization**, and select **User certificate** to download.

Step 3 Download and decompress the demo project code package **fabric-restapi-demo.zip** to the local PC, and use an IDE to open it.

This demo project is a RESTful client compiled using Golang. It enables chaincode invocation through RESTful APIs to achieve chaincode-based money transfer. Use an IDE such as GoLand to open the package. The following figure shows the content of the project.



Step 4 Decompress the downloaded user certificate to the **usercrypto** directory of the project, as shown in the following figure.



Step 5 Modify parameter settings.

1. Modify the parameters in the **conf.yaml** file in the **config** directory as shown and described in the following figure and tables.

2. Modify the **main.go** file in the **src/restapi** directory, as shown in the following figure and tables.

NOTE

For each peer that needs to participate in the endorsement, construct an OrgPeer structure including the organization ID and the domain name of the peer. Add the structure to an array of the OrgPeer type, convert the structure into a byte array using the json.Marshal() method, and then convert the structure into a character string. The OrgPeer structure is as follows:

```
type OrgPeer struct {
OrgId string `json:"orgId"`
PeerDomainName string `json:"peerDomainName"`
}
```

Table 4-11 Parameters

Parameter	Description
Endpoint	IP address and port number of the server bearing the RESTful service endpoint, which can be obtained by performing the following steps: 1. On an instance card, click **Container Cluster** to go to the CCE console. 2. In the navigation pane, choose **Resource Management > Nodes**. 3. In the **IP** column of the target instance, obtain the EIP. The port is fixed to 32621.
Path	Path to the RESTful APIs service. Retain the default value.
CryptoMethod	Encryption algorithm. If the ECDSA algorithm is used, set this parameter to **SW**.
SignCert	Path to the signature in the downloaded certificate.
PrvKey	Private key in the downloaded certificate.
InvokeReq	Request body parameters. Set these parameters based on the deployed chaincode. The **InvokeReq** parameter descriptions in the following table are for your reference.
QueryReq	Similar to **InvokeReq**. Set this parameter based on the deployed chaincode.

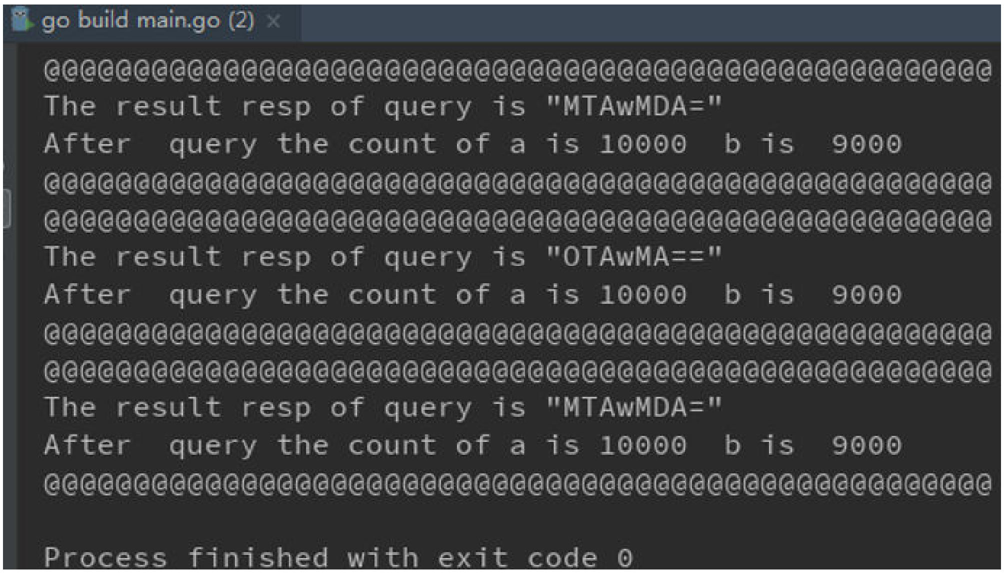
Table 4-12 InvokeReq parameters

Parameter	Description	Example Value
SignGzipSwitch	Indication of whether GZIP compression is enabled. **0** indicates disabling, and **1** indicates enabling.	"1"

Parameter	Description	Example Value
ChannelId	Blockchain channel name.	"channel"
ChaincodeId	Chaincode name.	"testcode"
Chaincode Version	Chaincode version.	"1.0"
UserId	User ID issued by the organization CA. The default value for BCS is **User1**.	"User1"
OrgId	Organization ID in a blockchain. **NOTE** On the **Channel Management** page, click **View Peer**. The organization ID is the value of **MSP ID** without "MSP". For example, if the MSP ID is **1232b2032faafee152b58cd82cecf52e49a22a38MSP**, the blockchain organization ID is **1232b2032faafee152b58cd82cecf52e49a22a38**.	"1232b2032faafee152b58cd82cecf52e49a22a38"
OrgPeers	Organization ID and domain name of each peer.	"[{OrgId:" 1232b2032faafee152b58cd82cecf52e49a22a38", PeerDomainName:"peer-1232b2032faafee152b58cd82cecf52e49a22a38-0 .peer-1232b2032faafee152b58cd82cecf52e49a22a38.default.svc.cluster.local "}]"
Opmethod	Purpose, that is, to **invoke** or **query** chaincodes.	"invoke"
Args	Chaincode invoking parameters.	["invoke","a","b","100"]

Step 6 Build and run main().

The code will read the **QueryReq** and **InvokeReq** parameters in **conf.yaml** and **main.go** and call **/v1/chaincode/operation** of the RESTful APIs to invoke the chaincode. The code running result is as follows.



NOTE

This demo uses a simple REST client to invoke the chaincode through RESTful APIs. The returned invocation result is TransactionID encrypted using Base64, and the query result is data encrypted using Base64. The code is for reference only. You can use this project code to understand how to invoke RESTful APIs.

----End

4.5 Node.js SDK Demo

This demo provides a Node.js chaincode and a program that uses the Hyperledger Fabric SDK for Node.js (fabric-nodejs-sdk) to invoke the chaincode to describe how to use a Node.js SDK to access BCS. For details about the APIs of Hyperledger Fabric SDK for Node.js, visit **https://hyperledger.github.io/fabric-sdk-node/release-1.4/index.html**.

NOTE

This is a demo only and is not for actual use.

Preparations

Step	Action	Description
1	Install a development tool.	Download and install the Node.js source code: **https://nodejs.org/en/download/**
2	Download the demo project code.	Project code: **nodejs-demo.zip**

Creating a BCS Instance

Step 1 Log in to the BCS console.

Step 2 Click **Create BCS Instance** in the upper right corner.

Step 3 Configure basic information about the BCS instance by referring to **Table 4-13**.

NOTICE

To ensure that the demo runs properly, set the parameters as described in the following table.

Table 4-13 Basic settings

Parameter	Setting
Region	Retain the default value.
Enterprise project	Select **default**.
Instance Name	Enter **node-sdk-demo**.
Edition	Professional
Blockchain Type	Select **Private**.
Enhanced Hyperledger Fabric Version	v2.2
Consensus Mechanism	Select **SOLO**.
Resource Access Initial Password	Enter a password.
Confirm Password	Confirm the password.

Step 4 Click **Next: Configure Resources**. **Table 4-14** describes the resource parameters.

Table 4-14 Resource configurations

Parameter	Example
Environment Resources	Select **Custom**.
Cluster	Select **Create a new CCE cluster**.
AZ	Select an AZ.
ECS Specifications	Select the flavor for **4 vCPUs | 8 GB**.
ECS Quantity	Enter **1**.
High Availability	Select **No**.

Parameter	Example
VPC	Select **Automatically create VPC**.
Subnet	Select **Automatically create subnet**.
ECS Login Method	Select **Password**.
Password of Root User	If you do not enter a password here, the previously specified resource access initial password will be used.
Confirm Password	-
Use EIP of a CCE Node	Select **Yes**.
EIP Billed By	Retain the default value.
EIP Bandwidth	Set it to 5 Mbit/s.

Step 5 Click **Next: Configure Blockchain**. Table 4-15 describes the blockchain parameters.

Table 4-15 Blockchain configurations

Parameter	Example
Blockchain Configuration	Select **Custom**.
Blockchain Mgmt. Initial Password	If you do not enter a password here, the previously specified resource access initial password will be used.
Confirm Password	-
Volume Type	Select **SFS Turbo**.
Storage Capacity of Peer Organization (GB)	Retain the default value.
Ledger Storage	Select **File database (GoLevelDB)**.
Peer Organization	A peer organization named **organization** has been automatically created. Change the peer quantity to 1.
Channel Configuration	The **organization** organization has been added to the channel automatically. Retain this default setting.
Orderer Quantity	Retain the default value.
Security Mechanism	Select **ECDSA**. **NOTICE** Only **ECDSA** can be selected.
Configure Block Generation	Select **No**.

Parameter	Example
Enable Support for RESTful API	Select **No**.

Step 6 Click **Next: Confirm**.

Step 7 Confirm the configurations and finish the creation process.

Wait for several minutes. After a message is displayed indicating successful installation, check the status of the instance. If it is **Normal**, the deployment is completed.

----End

Installing and Instantiating a Chaincode

Step 1 Log in to the BCS console.

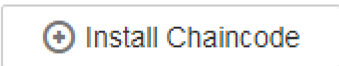
Step 2 In the navigation pane on the left, click **Instance Management**.

Step 3 Find the instance you just created and click **Manage Blockchain** to go to the Blockchain Management console.

Step 4 On the login page, enter the username and password, and click **Log In**.

NOTE

The username is **admin**, and the password is the **Blockchain Mgmt. Initial Password** set when you created the BCS instance. If you have not set this password, use the resource access initial password.

Step 5 Click 

in the upper left corner of the page.

The parameters for chaincode installation are as follows.

Parameter	Setting
Chaincode Name	Enter **bcsysq**.
Chaincode Version	Enter **1.0**
Ledger Storage	File database (goleveldb)
Select All Peers	Check the box.
Organization & Peer	Select **peer-0**.
Language	Select **Node.js**.
Chaincode File	Add the downloaded chaincode file **nodejs-chaincode.zip**.
Chaincode Description	Enter a description of the chaincode.

Step 6 Click **Install**.

Step 7 After installing the chaincode, click **Instantiate** in the **Operation** column of the chaincode list.

The parameters for chaincode instantiation are as follows.

Parameter	Setting
Chaincode Name	Enter **bcsysq**.
Channel	Select **channel**.
Chaincode Version	Enter **1.0**
Initialization Function	Enter **init**.
Chaincode Parameters	Enter **a,200,b,250**.
Endorsement Policy	Select **Endorsement from any of the following organizations**.
Endorsing Organizations	Select **organization**.
Privacy Protection Configuration	Select **No**.

----End

Deploying the Application

Step 1 Download and decompress the demo project code package **nodejs-demo.zip** to the local PC, and use an IDE to open it.

This demo is compiled using Node.js. It contains the fabric-client library to enable money transfer from user A to user B by running the chaincode. Use the IDE that you prefer to open the script. The following table lists the files included in the project.

Table 4-16 Project content

Fie	Description
invoke.js	Invoking the chaincode to transfer money from user A to user B.
query.js	Querying the chaincode to determine the account balance of user A.
sdk-config.js	Parsing the blockchain network configuration file **sdk-config.json** downloaded from the BCS console.
sdk-config.json	Containing the blockchain network configuration data.

Step 2 Download SDK configurations and certificates.

1. On the **Instance Management** page, click **Download Client Configuration** on an instance card.
2. Select **SDK Configuration File** and set the parameters as described in the following table.

Parameter	Description
Chaincode Name	Enter **bcsysq**. **NOTICE** The chaincode name must be the same as the name specified during chaincode installation and instantiation.
Certificate Root Path	Specify a path for storing certificates. In this example, the path is **E:\code\temp**.
Channel	Select **channel**.
Organization & Peer	Retain the default value.

Select **Orderer Certificate**.

Select **Peer Certificates**, select **organization** for **Peer Organization**, and select **Administrator certificate**.

3. Click **Download** to download the SDK configuration file and the administrator certificates for the **node-sdk-demo-orderer** and **organization** organizations.

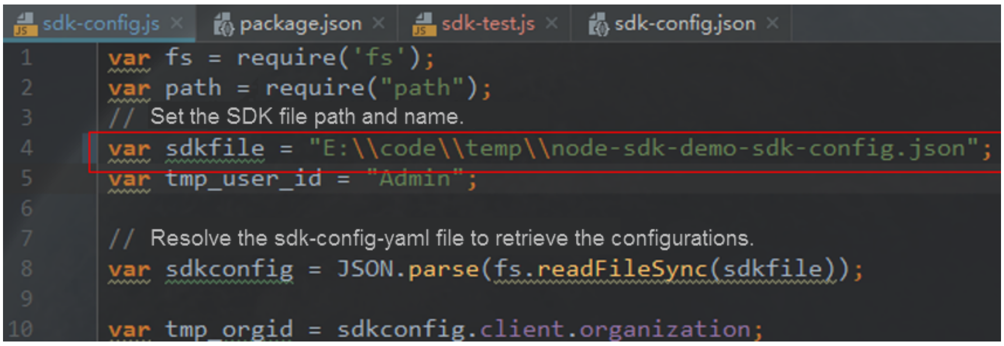
Step 3 Copy and decompress the package.

1. Download the project source code **javasdkdemo_src.zip** and decompress it.
2. Decompress the .zip package obtained in **Step 2** and copy the content in the **orderer** and **peer** folders to the certificate storage path. Copy the **sdk-config.json** file to the directory where the certificate is stored and name the file **node-sdk-demo-sdk-config.json**.

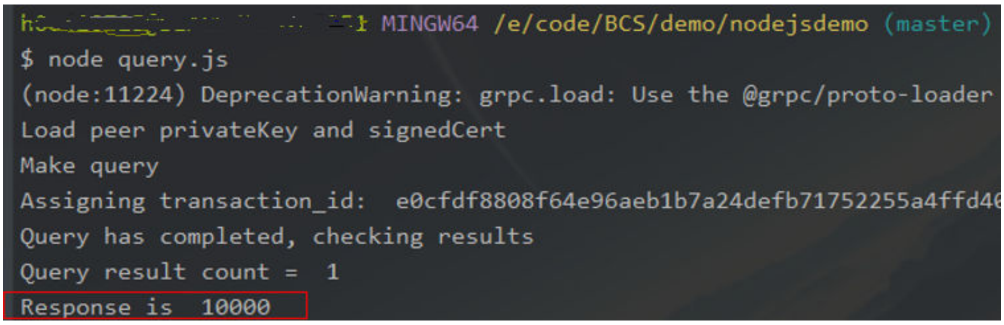
----End

Debugging the Application

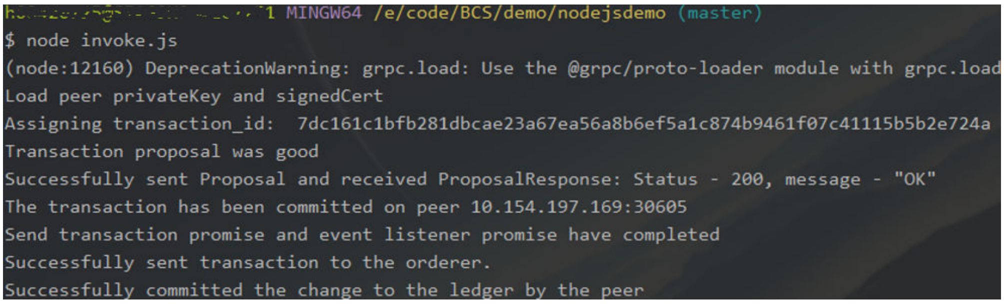
Step 1 Open the **sdk-config.js** file, and change the path of the SDK configuration file to the path of **node-sdk-demo-sdk-config.json**.



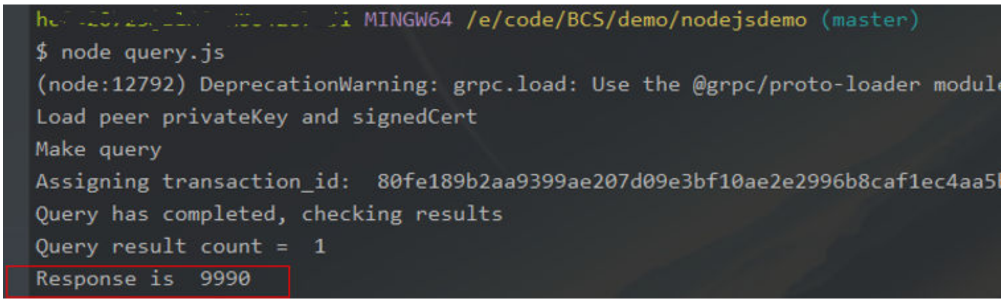
Step 2 Run the **node query.js** command in the path where the project is located to query the account balance of user A. (As shown in the following figure, user A's balance is 10,000 in this example.)



Step 3 Run the **node invoke.js** command in the path where the project is located to execute the chaincode to transfer money from user A to user B. As shown in the following figure, the chaincode is successfully executed.



Step 4 Query the account balance of user A again. The result shows that the money has been successfully transferred. (As shown in the following figure, user A's balance is 9990 in this example.)



----End

Description

- This demo uses demos in the Fabric community as references. For more demos, visit **https://github.com/hyperledger/fabric-samples**.
- Method of customizing an npm repository in the chaincode
  - Create a personal configuration file named **.npmrc** in the path of the chaincode package. The content of the file is as follows: Registry=https://registry.npm.taobao.org/
  - In this way, when the chaincode container is instantiated, the class library is extracted from the specified repository.

– Run the **npm config get registry** command to check whether the address of the customized npm repository has been correctly configured.

5 Blockchain Middleware APIs

5.1 Overview

This chapter describes data plane APIs. For details about management plane APIs, see *API Reference*.

The endpoint of data plane requests can be obtained from the value of the **basic_info->agent_portal_addrs** field returned by the API for querying BCS instance details. An example request is **https://192.168.0.90:30603/v2/agent/apis/tokens**.

5.2 Chaincode Invoking

Function

This API is used to invoke and query the instantiated chaincodes of deployed BCS services.

URI

POST /v1/chaincode/operation

Request

Table 5-1 Request parameters

Parameter	Mandatory	Type	Description
channelId	Yes	String	Channel ID in a blockchain.
chaincodeId	Yes	String	Chaincode ID.
chaincodeVersion	No	String	Chaincode version.

Parameter	Mandatory	Type	Description
userId	Yes	String	User ID issued by the organization CA. Currently, the default value generated for BCS is **User1**.
orgId	Yes	String	Organization ID in a blockchain.
orgPeers	Yes	String	A character string consisting of the organization ID and domain name of each peer in an organization. The format is as follows: [{"orgId":"7258adda1803f4137eff4813e7aba323018200c5","peerDomainName":"peer-7258adda1803f4137eff4813e7aba323018200c5-0.peer-7258adda1803f4137eff4813e7aba323018200c5.default.svc.cluster.local"}]
opmethod	Yes	String	Purpose, that is, to **invoke** or **query** chaincodes.
args	Yes	String	Arguments, for example, ["Invoke", "a", "b", "1"]
timestamp	Yes	String	For example, 2018-10-31T17:28:16+08:00.
cert	Yes	String	User certificate file, which is uploaded in the form of a character string.

NOTE

For details about how to obtain the values of the preceding parameters, see **Chaincode Management** and **Block Browser**.

- On the **Chaincode Management** page, click in front of a chaincode to view its details, including the version, installation, and instantiation information.
- On the **Block Browser** page, select a channel to view real-time blockchain information, including the block quantity, transaction quantity, block details, transaction details, performance, and peer statuses.
- To ensure transaction security, you must use the private key in the Fabric user certificate (downloaded by following instructions in **Downloading the User Certificate**) to sign the request body. Currently, only the ECDSA encryption method is supported. Other encryption algorithms such as OSCCA-published cryptographic algorithms are not supported. Then, place the signature result in the **x-bcs-signature-sign** field in the request header.

Table 5-2 lists the request header parameters customized for the chaincode invoking RESTful API.

Table 5-2 Customized header parameters

Parameter	Mandatory	Description
x-bcs-signature-sign	Yes	Signature of the chaincode invoking request message body
x-bcs-signature-method	Yes	Encryption type, which is fixed at **SW** now.
x-bcs-signature-sign-gzip	Yes	Indication of whether GZIP compression is enabled. **0**: disabled; **1**: enabled.

NOTE

x-bcs-signature-sign: To ensure that only authorized invocation entities can invoke chaincodes, the user private key (downloaded by following instructions in **Downloading the User Certificate**) and the ECDSA encryption method must be used to encrypt and sign the SHA256 hash of the entire request body. The value of **x-bcs-signature-sign** is the encrypted and signed hash.

Downloading the User Certificate

Download the user certificate that is configured in BCS to call the APIs.

Step 1 Log in to the BCS console.

Step 2 On the **Instance Management** page, check the BCS instances.

Step 3 Click **Download Client Configuration**, and select **Peer Certificates**. Specify the peer organization and select **User certificate**.

Step 4 Click **Download**.

Step 5 Decompress the certificates. In the **msp** folder, the private key of the organization is stored in **keystore**, and the user certificate (public key) in **signcerts**.

----End

Response

- If **opmethod** is **invoke**, the **transactionID** encrypted and encoded using Base64 is returned.
- If **opmethod** is **query**, the query result returned by the chaincode is also encrypted and encoded using Base64.

Examples

The following is an example of invoking a chaincode:

- Example request

```
{
  "channelId": "testchannel",
  "chaincodeId": "zmmcode",
```

```
  "chaincodeVersion": "1.0",
  "userId": "User1",
  "orgId": "7258adda1803f4137eff4813e7aba323018200c5",
  "orgPeers": "[{\"orgId\":\"7258adda1803f4137eff4813e7aba323018200c5\",\"peerDomainName\":
\"peer-7258adda1803f4137eff4813e7aba323018200c5-0.peer-7258adda1803f4137eff4813e7aba32301
8200c5.default.svc.cluster.local\"}]",
  "opmethod": "invoke",
  "args": "[\"invoke\",\"a\",\"b\",\"1\"]",
  "timestamp": "2018-10-31T17:28:16+08:00",
  "cert": "-----BEGIN CERTIFICATE-----
\nMIIDBzCCAq2gAwIBAgIQEXPZlMsReamxVtVNnKwCCzAKBggqhkjOPQQDAjCCAQQx
\nDjAMBgNVBAYTBUNISU5BMRAwDgYDVQQIEwdCRUlKSU5HMRAwDgYDVQQHEwdCRUlK
\nSU5HMXkwdwYDVQQKE3A3MjU4YWRkYTE4MDNmNDEzN2VmZjQ4MTNlN2FiYTMyMzAx
\nODIwMGM1LnBlZXItNzI1OGFkZGExODAzZjQxMzdlZmY0ODEzZTdhYmEzMjMwMTgy
\nMDBjNS5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsMVMwUQYDVQQDE0pjYS5wZWVy
\nLTcyNThhZGRhMTgwM2Y0MTM3ZWZmNDgxM2U3YWJhMzIzMDE4MjAwYzUuZGVmYXVs
\ndC5zdmMuY2x1c3Rlci5sb2NhbDAeFw0xODEwMzAwMjQ5MjZaFw0yODEwMjcwMjQ5\nMjZaMIG1M
Q4wDAYDVQQGEwVDSElOQTEQMA4GA1UECBMHQkVJSklORzEQMA4GA1UE\nBxMHQkVJSklORzF/
MH0GA1UEAwx2VXNlcjFANzI1OGFkZGExODAzZjQxMzdlZmY0\nODEzZTdhYmEzMjMwMTgyMDBjNS5
wZWVyLTcyNThhZGRhMTgwM2Y0MTM3ZWZmNDgx
\nM2U3YWJhMzIzMDE4MjAwYzUuZGVmYXVsdC5zdmMuY2x1c3Rlci5sb2NhbDBZMBMG
\nByqGSM49AgEGCCqGSM49AwEHA0IABPMrzoJL/MHeSFPFOJWLqnJ0sqB0it7wDIOq\n
+eTSvvPpGk1BIDmb2n13K5V04RO8xNezDQ7I6rW4LF2elq14eH+jTTBLMA4GA1Ud\nDwEB/
wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1UdIwQkMCKAIFBXQ5TC4acFeTlT
\nJuDZg62XkXCdnOfvbejSeKI2TXoIMAoGCCqGSM49BAMCA0gAMEUCIQCadHIKl0Mk
\nYn0WZizyDZYR4rT2q0nzjFaiW+YfV5FBjAIgNalKUe3rIwXJvXORV4ZXurEua2Ag\nQmhcjRnVwPTjpTE=
\n-----END CERTIFICATE-----\n"
}
```

- Example response

```
After  invoke the count of a is 188  b is  262
```

Error Codes

See **Error Codes**.

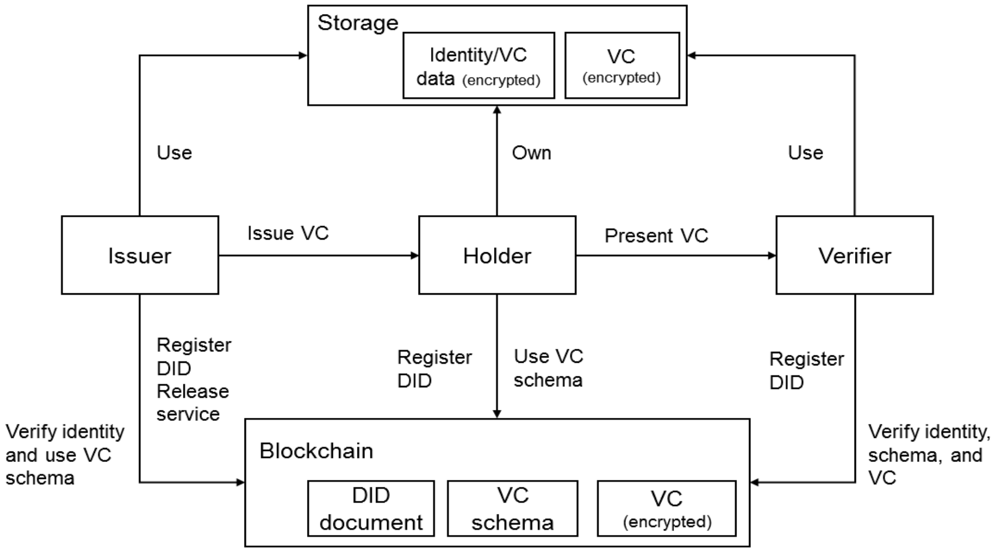
5.3 Distributed Identity

5.3.1 Overview

Distributed identity (DID) is a blockchain-based identity management technology. It allows you to create user identities, and register, issue, and verify verifiable credentials (VCs). BCS's DID is implemented based on the W3C DID and VC specifications. It provides unified, self-explainable, and portable distributed identifiers for individual and enterprise users to address privacy issues and identity authentication across departments, enterprises, and regions.

Figure 5-1, **Figure 5-2**, **Figure 5-3**, and **Figure 5-4** illustrate the implementation and usage of DID.

Figure 5-1 DID implementation



Implementation

1. Each role can call the **Enterprise Identity Registration (with Service)** and **Registering a DID** APIs to generate their own DID which they fully control, and then publish the DID document to the blockchain to complete identity registration. The DID of an enterprise includes information about the services the enterprise can provide in various application scenarios.

   NOTE

   There are three roles: issuer, holder, and verifier. Each role can be a device, an application, an individual, or an organization.

2. VCs are used to authenticate identities. Relevant entities register and continuously maintain credential schemas on the blockchain. Holders initiate authentication requests to issuers, obtain credentials, and provide the credentials to verifiers to complete verification.
3. Verifiers verify VCs presented by holders by using an API to ensure that holders are qualified and permitted to proceed with relevant services.

Usage

The DID middleware is a set of microservices deployed on the user side to simplify the calling of blockchain APIs. When calling the DID API, the user private key and the Fabric-signed root certificate must be transferred.

NOTE

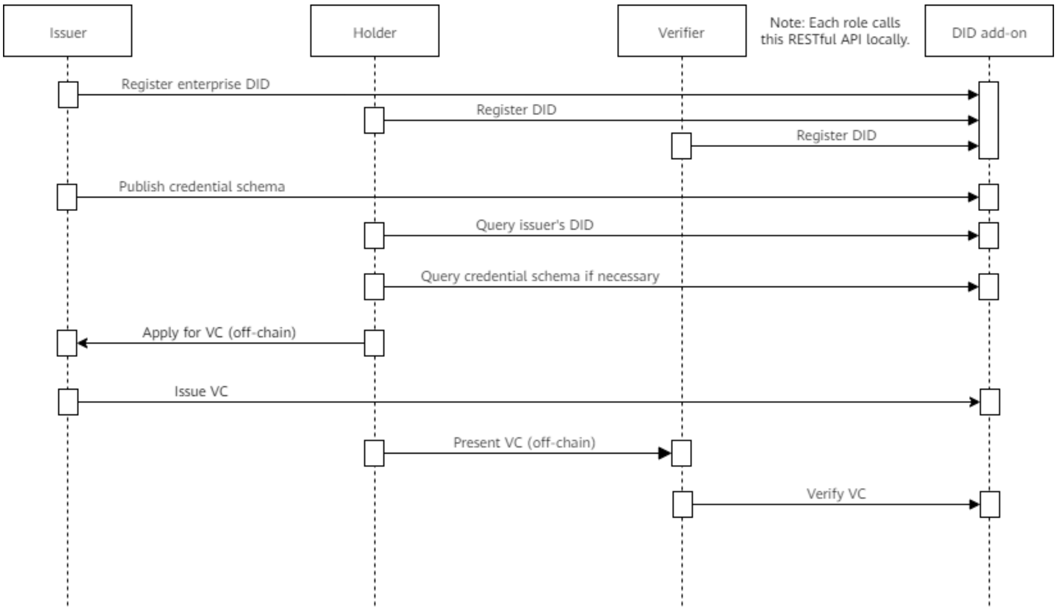
You can download the private key and certificate on the BCS console or generate them using OpenSSL. For details, see *Blockchain Service User Guide* > FAQs > How Can I Obtain Private Keys and Certificates of Fabric Users?

Holders can apply for VCs on or off the blockchain.

- Off-chain application: The holder sends the identity or VC data for applying for a VC to the issuer.

- On-chain application: The holder encrypts and stores the identity or VC data for applying for a VC on the blockchain.

Figure 5-2 DID usage (off-chain application)



On-chain application can be online or offline, depending on whether communication channels are required between the holder and the issuer.

Figure 5-3 DID usage (on-chain, online application)

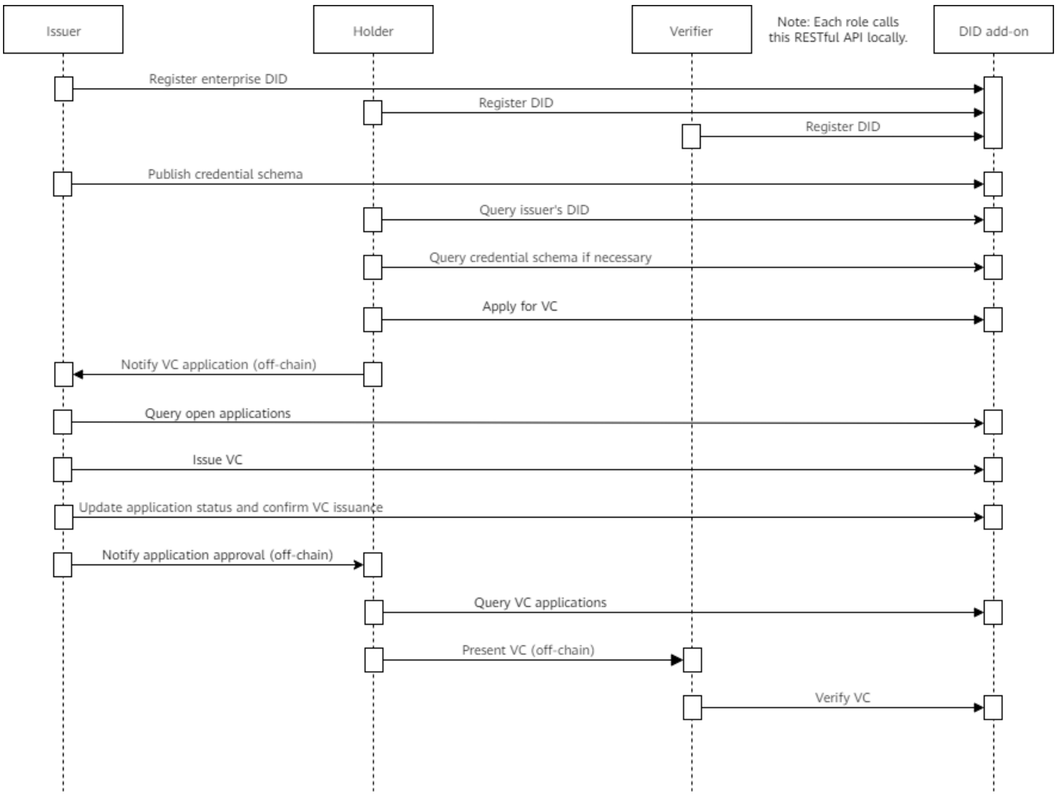
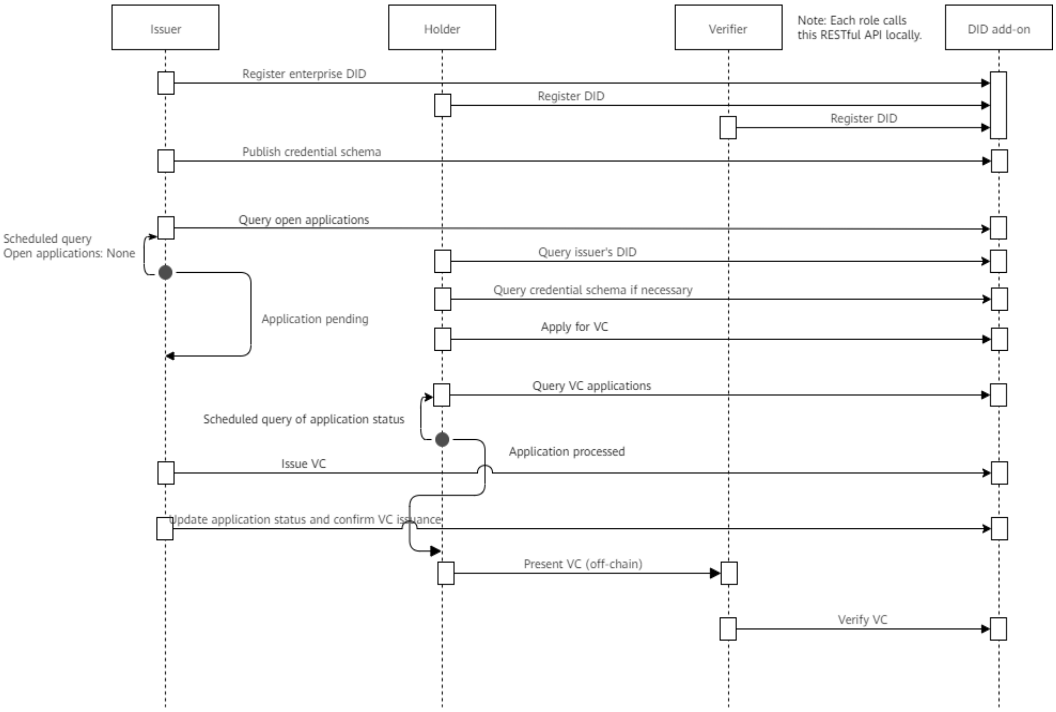


Figure 5-4 DID usage (on-chain, offline application)



5.4 Trusted Data Exchange

5.4.1 Overview

Data is crucial to business. In distributed applications, data is exchanged to break data silos and maximize data value. Trusted data exchange based on the blockchain can ensure data privacy and trustworthy data sharing. BCS's trusted data exchange middleware is integrated with the RESTful APIs add-on, which can be quickly installed and uninstalled, and supports elastic scaling. Users can access the blockchain system through RESTful APIs to quickly integrate data release, authorization, sharing, encryption, decryption, and fine-grained access control capabilities.

Function

- Trusted data exchange involves two main data structures: data sets and data orders. Data sets contain data description and access control information (attributed-based encryption policies). Data orders contain data application and review information.
- Trusted data exchange supports three modes: application-authorization, proactive sharing, and fine-grained access control, as described in **Exchange Modes**.

NOTE

You can choose from multiple storage services to store the encrypted data to be exchanged. The one calling the API is responsible for storing ciphertexts to publicly accessible storage devices.

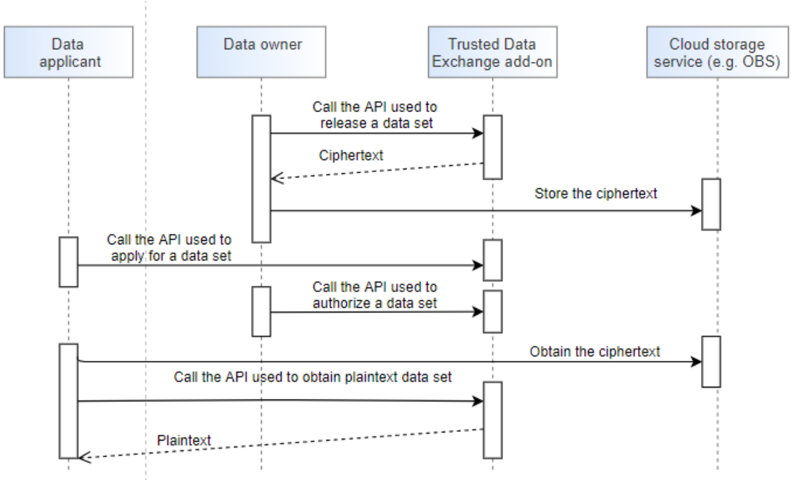
Roles

Trusted data exchange involves two roles: data owner and data applicant. Each user can be both an owner and an applicant.

Exchange Modes

- Application-authorization, which is illustrated in **Figure 5-5**.
  - The data owner calls the API used to release a data set to encrypt plaintext user data and register and release data description information.
  - The data applicant calls the API used to apply for a data set to invoke a chaincode to trigger the application-authorization process.
  - The data owner decides whether to authorize or reject the application based on the application information and the applicant's DID and VC information.

Figure 5-5 Application-authorization

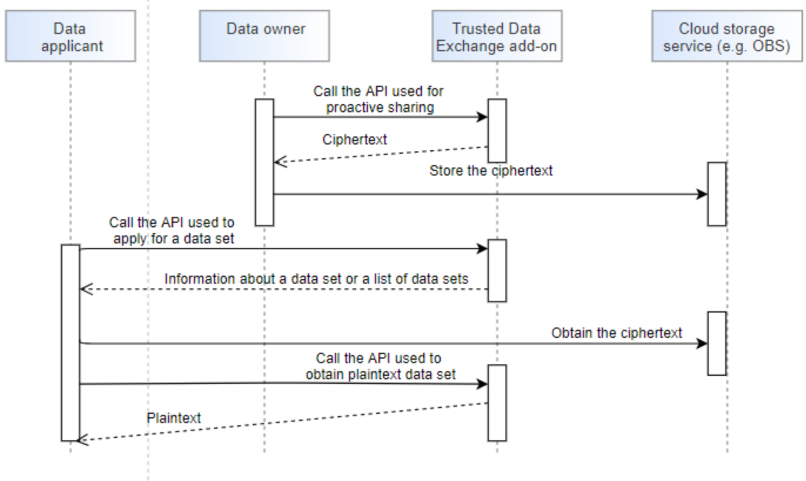


- Proactive sharing, which is illustrated in **Figure 5-6**.

  The API used to proactively sharing data sets is a combination of the APIs used to release and authorize data sets. The data owner releases a dataset to the blockchain and authorizes an applicant to decrypt the dataset. The authorized applicant can then directly decrypt the dataset. Other users can obtain the data description information by calling the APIs used to query a specific dataset and the dataset list, and then obtain the data decryption permission through the application-authorization process.

  For details about the APIs, see "Data Set Management" and "Data Order Management".

Figure 5-6 Proactive sharing



- Fine-grained access control, which is illustrated in **Figure 5-7**.

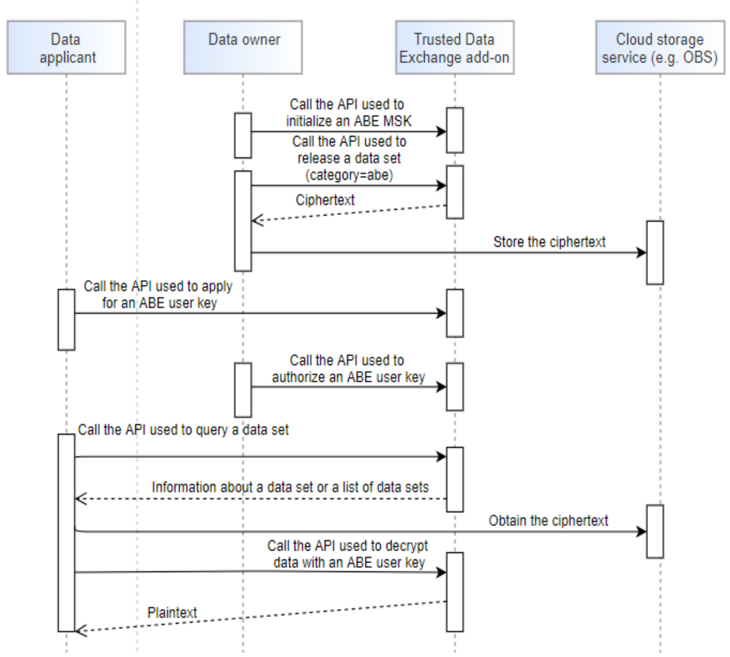
  NOTE

  Attribute-based encryption (ABE) achieves fine-grained, attribute-level access control for data exchange. Each data set is configured with an appropriate, owner-defined sharing policy. Data applicants with sufficient attributes are allowed to access the ciphertext.

  Fine-grained access control is implemented through ciphertext-policy ABE (CP-ABE). The policy is embedded in the ciphertext and the attribute is embedded in the user key.

  – Each data owner initializes its own master public key and private key only once.
  – When a data applicant needs to use some data, the applicant applies to the owner of the data for a user key. If attributes do not change, the applicant only applies for a user key once.
  – With a user key and sufficient attributes, the data applicant can asynchronously decrypt all data released by the data owner at any time as required.
  – For details about the APIs, see "Attribute-based Encryption Key Management".

Figure 5-7 Fine-grained access control



NOTE

Basic concepts:

- Attribute: An attribute describes the association between an entity and its nature.
- Policy: A policy is a combination of attribute sets and logical relationships. A policy is defined by the data owner and embedded in the ciphertext. For example, the policy "age>26 && gender=man" indicates that the age must be greater than 26 and the gender must be man.
- ABE master key: An ABE master key consists of a master public key (MPK) and a master secret key (MSK). They are owned by the data owner and are used to encrypt data and generate a user (data applicant) key.
- User key: A data applicant applies to data owners for a user key after submitting a set of attributes to the data user. A user key contains the user attribute information and is used to decrypt a ciphertext.

6 Appendix

6.1 Encryption Using OSCCA-Published Cryptographic Algorithms

6.1.1 Overview

OSCCA-published cryptographic algorithms are cryptographic technologies and products used for the encryption or authentication of information that does not involve state secrets.

OSCCA-published cryptographic algorithms are independent and controllable algorithms developed by the State Cryptography Administration. They can improve the encryption strength and encryption and decryption performance. OSCCA-published cryptographic algorithms meet the requirements of government agencies, public institutions, large state-owned enterprises, financial banks, and other industries for localized modernization.

BCS provides the OSCCA-published cryptographic algorithm SDKs for you to develop client programs while protecting your private keys.

Downloading Resources

Table 6-1 SDK list

Matching Hyperledger Fabric Version	Language	Link
Fabric v2.2	Go	Go to the BCS console and click **Use Cases**. Download **Fabric_SDK_Go**.
	Java	Go to the BCS console and choose **Interactive Walkthroughs** > **Use Cases**. Download **Fabric_SDK_Gateway_Java** or **Fabric_SDK_Java**.

Matching Hyperledger Fabric Version	Language	Link
NOTE • The Go version must be v1.12 or later but must be earlier than v1.16. • The OSCCA-published cryptographic algorithm SDKs offer all functions of common SDKs and additionally support the OSCCA-published cryptographic algorithms. • Fabric_SDK_Gateway_Java encapsulates some interfaces of the SDK, covering Fabric_SDK_Java and making it easier to use. Therefore, Fabric_SDK_Gateway_Java is recommended.		

After decompressing the downloaded package, you will obtain the directories listed in the following table.

Directory	Description
src (Golang only)	Stores the Go SDK source code.
jar (Java only)	Stores the JAR package of the Java SDK.

6.1.2 Using SDKs

Installing the SDK

For details about how to obtain the Golang and Java packages, see **Overview**.

- Golang: Decompress the downloaded package to the **$GOPATH** directory.
- Java: Add the JAR file in the downloaded package to the dependency of the project as follows:
  a. Run the following command to register the downloaded SDK JAR package to the local Maven repository:
```
Mvn install:install-file -Dfile=fabric-sdk-java-2.2.6-jar-with-dependencies.jar -DgroupId=org.hyperledger.fabric-sdk-java -DartifactId=fabric-sdk-java -Dversion=2.2.6-BCS -Dpackaging=jar
```
  b. Add the SDK dependency in the project by using the following code:
```
<dependency>
    <groupId>org.hyperledger.fabric-sdk-java</groupId>
    <artifactId>fabric-sdk-java</artifactId>
    <version>2.2.6-BCS</version>
 </dependency>
```

Running the Client Program

To run the client program, you need to set the configuration file path, channel name, chaincode name, and organization ID.

- Configuration file path: the path for storing the downloaded configuration file
- Channel name: the channel name specified for the BCS instance
- Chaincode name: the name specified when the chaincode is installed for the BCS instance

- Organization ID: 02f23ab00f6e1ffcde8a27bfd3ac2290edc18127 in the following example configuration file

```
client:
organization: 02f23ab00f6e1ffcde8a27bfd3ac2290edc18127
```

6.1.3 Appendix

The following table lists the third-party packages that **fabric-sdk-client/go** depends on.

No.	Package
1	github.com/Knetic/govaluate
2	github.com/VividCortex/gohistogram
3	github.com/cloudflare/cfssl
4	github.com/go-kit/kit
5	github.com/golang/mock
6	github.com/golang/protobuf
7	github.com/hashicorp/hcl
8	github.com/hyperledger/fabric-config
9	github.com/hyperledger/fabric-lib-go
10	github.com/hyperledger/fabric-protos-go
11	github.com/magiconair/properties
12	github.com/miekg/pkcs11
13	github.com/mitchellh/mapstructure
14	github.com/pelletier/go-toml
15	github.com/pkg/errors
16	github.com/prometheus/client_golang
17	github.com/spf13/afero
18	github.com/spf13/cast
19	github.com/spf13/jwalterweatherman
20	github.com/spf13/pflag
21	github.com/spf13/viper
22	github.com/stretchr/testify
23	github.com/tjfoc/gmsm
24	google.golang.org/grpc
25	gopkg.in/yaml.v2

6.2 Error Codes

If an error occurs in API calling, no result is returned. Identify the cause of error based on the error codes and error information of each API. If an error occurs in API calling, HTTPS status code 4*xx* or 5*xx* is returned. The response body contains the specific error code and information. If you are unable to identify the cause of an error, contact O&M personnel and provide the error code so that O&M personnel can help you solve the problem as soon as possible.

Format of an Error Response Body

If an error occurs during API calling, the system returns an error code and message to you. **Table 6-2** shows the format of an error response body.

Table 6-2 Error response parameters

Parameter	Type	Description
error_msg	String	Error code
error_code	String	Error description

Example:

```
{
  "error_code": "BCS.4006009",
  "error_msg": "one of parameters is nil"
}
```

Error Code Description

Table 6-3 Error codes

Status Code	Error Code	Error Message	Solution
200	BCS.2001002	OneStepPurchase4Need successfully.	Request successful.
400	BCS.4006009	one of parameters is nil	Specify a value for the parameters.
400	BCS.4001051	Input org org1 is not exist	Check the parameters.
400	BCS.4030403	forbidden to project [project_id]	Check the project ID.

Status Code	Error Code	Error Message	Solution
400	BCS.4006003	json: cannot unmarshal object into Go value of type []apis.PeersToChannelAdd	Check whether the parameter meets requirements.
400	BCS.4006005	ActOnDetailNotification fail	Contact technical support.

7 Description

Date	Description
2023-04-21	Updated **Blockchain Middleware APIs**.
2021-01-15	This issue is the first official release.

