概述

数字对象体系（Digital Object Architecture，下面简称为DOA）是图灵奖得主、互联网发明人Robert E. Kahn提出的一种实现异构信息系统互操作的新型计算架构。

在DOA架构中，将信息资源抽象表示为数字对象（Digital Object，下面简称DO），所有的DO都存储在DO Repository中，并且支持通过数字对象接口协议（Digital Object Interface Protocol，下面简称DOIP）来对外提供服务。每一种DO具有全球可解析的唯一标识，可以通过标识与解析协议（Identifier/Resolution Protocol，下面简称IRP）向Handle System进行注册与解析。同时每一个DO也具有对该DO进行描述的元信息，元信息存储在DO Registry（一种特殊的DO Repository）中，也支持通过DOIP来对外提供服务。

根据上面的描述，在DOA架构中，有三种构件和两种协议。三种构件分别是：Handle System，DO Repository和DO Registry。两种协议分别是：DOIP和IRP。北京大学提供了DOIP和IRP的代码实现，并提供了用于开发DO Repository，DO Registry以及Client的软件开发工具包（Software Development Kit，简称SDK）。服务提供者，可以基于该SDK实现DO Repository和DO Registry功能。同时，我们提供用于对接国内主流Handle System提供者的支持代码，比如中数、中科院网络中心等。服务提供者可以基于该代码开发用于提供标识与解析的服务。最终使用者，可以基于Client的SDK开发基于DOA的应用程序。

快速上手

环境准备

- JDK (>= Java 1.8)
- Gradle (>= 6.0)

下载示例代码

- example-Repository
  v0.1
- example-Client
  v0.1
- example-IRPProxy
  v0.1

运行代码

下载回来的示例代码是一个压缩包，解压后包含一个jar包和一个conf文件夹（里面包含示例代码运行需要使用的配置文件）。

启动Repository

将Repository示例代码解压后，打开命令窗口，进入包含jar包的目录，此时目录内容如下：

```
ls
conf repo-0.1-all.jar
```

执行如下命令，开启Repository：

```
java -jar repo-0.1-all.jar
```

此时，命令行出现如下日志信息，表示Repository启动成功。

```
log4j:WARN [./app.log] should be System.out or System.err.
log4j:WARN Using previously set target, System.out by default.
[INFO ]11:07:13.600 use config file: conf/default.conf (Config.java:36)
[INFO ]11:07:13.604 [CONFIG] read config from :conf/default.conf
(Config.java:77)
[INFO ]11:07:13.605 [CONFIG]file absolute path:conf/default.conf
(Config.java:89)
[INFO ]11:07:14.095 using rocksdb as storage backend (Main.java:16)
[INFO ]11:07:15.213 DOIPServiceInfo:
{"id":"86.5000.470/local.TLSRepository","publicKey":"MIIBIjANBgkqhkiG9w0BAQEFAAO
CAQ8AMIIBCgKCAQEAjtkzhbVFxW5yTLeqTnYRr+fmTdlfi5OYPOqSLiss3OzXCbpH7H5ywcT5L4LQ3Gm
ynyifzhLW2VNnQavDaBYeLzy1stcMwEyWSTmjf/O8ebvBvB3Gdm10GBsGO1xpcDupuDSXG6iesRS+/6j
nLDA3t9D913sH+JewBB8oea+yYKtWyJW8eXSjwsyjdz/OydFaxLKpNqL+vMP1+7leqjvXELWOZPOXhKb
iJRwkTRWs2OB/WHjsODXnPE8OQlEs31pxBPUOl8sFdcKFFUd4NgdFIAmYiY/OASB3yYPSORkTtKM1LJD
hVxx8OrJmJMQPbPSp5DzRYoidtbdAWFYWzVVrswIDAQAB","port":1716,"ipAddress":"127.0.0.
1","protocol":"tls","protocolVersion":"2.0","listenerInfos":
[{"url":"tls://127.0.0.1:1716","protocolVersion":"2.0","messageFormat":"delimite
r"}],"owner":"86.5000.470/dou.SUPER","repoType":"Repository"}
(DoipServer.java:40)
[INFO ]11:07:16.137 start at:1716 (TLSV2Listener.java:70)
```

启动Client

将Client示例代码解压后，打开命令窗口，进入包含jar包的目录，此时目录内容如下：

```
ls
conf client-0.1-all.jar
```

执行如下命令，开启Client：

```
java -jar client-0.1-all.jar
```

此时，命令行出现如下日志信息，表示Client启动成功。

```
log4j:WARN [./app.log] should be System.out or System.err.
log4j:WARN Using previously set target, System.out by default.
[INFO ]11:11:55.436 client started (Main.java:13)
[INFO ]11:11:56.462 [DoipClient] Create TLS Client! (DoipTLSClient.java:55)
[INFO ]11:11:57.490 get server information (Main.java:17)
server info:  •
闐"id":"86.5000.470/local.TLSRepository","type":"0.TYPE/DO.DOIPServiceInfo","att
ributes":
{"owner":"86.5000.470/dou.SUPER","repoType":"Repository","publicKey":"MIIBIjANBg
kqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjtkzhbVFxW5yTLeqTnYRr+fmTdlfi5OYPOqSLiss3OzXCb
pH7H5ywcT5L4LQ3GmynyifzhLW2VNnQavDaBYeLzy1stcMwEyWSTmjf/O8ebvBvB3Gdm10GBsGO1xpcD
upuDSXG6iesRS+/6jnLDA3t9D913sH+JewBB8oea+yYKtWyJW8eXSjwsyjdz/0ydFaxLKpNqL+vMP1+7
leqjvXELWOZPOXhKbiJRwkTRWs2OB/WHjsODXnPE8OQlEs31pxBPUOl8sFdcKFFUd4NgdFIAmYiY/0AS
B3yYPSORkTtKM1LJDhVxx8OrJmJMQPbPSp5DzRYoidtbdAWFYWzVVrswIDAQAB","protocol":"tls"
,"protocolVersion":"2.0","port":1716,"ipAddress":"127.0.0.1","listenerInfos":"
[{\"url\":\"tls://127.0.0.1:1716\",\"protocolVersion\":\"2.0\",\"messageFormat\"
:\"delimiter\"}]"}}
send message header:
{"id":"86.5000.470/local.TLSRepository","operation":"0.DOIP/Op.Create"}
send message body:
j{"id":"86.5000.470/test.hello","type":"0.TYPE/DO.String","elements":
[{"id":"1","length":11,"type":"Doc"}]}hello world
```

默认情况下，Client示例程序将会连接上面启动的Repository，并获取服务端信息，同时创建一个DO并取回DO。

编写自己的应用

环境准备

- JDK
- Gradle

下载SDK代码

- doip-java-sdk
  v0.1

下载密钥生成工具

- key-generator
  v0.1

引入SDK

在源码目录下创建libs目录，将下载的SDK文件解压出来的jar包拷贝到libs目录中，并在build.gradle中添加如下代码：

```
repositories {
    maven { url 'http://maven.aliyun.com/nexus/content/groups/public' }
    flatDir{ dirs 'libs'}
}

dependencies {
```

```
    compile "bdware.doip.sdk:doip-java-sdk:0.1"
    compile "bdware.doip.codec:netty-codec:0.1"
    compile "bdware.doip.client:netty-doclient:0.1"
    compile "bdware.doip.server:netty-doserver:0.1"

    compile 'org.codehaus.groovy:groovy-all:2.3.11'
    compile 'net.sf.proguard:proguard-gradle:6.0.3'
    compile 'log4j:log4j:1.2.17'
    compile group: 'org.rocksdb', name: 'rocksdbjni', version: '6.4.6'
    compile group: 'org.apache.httpcomponents', name: 'httpclient', version:
'4.5.11'
    compile group: 'com.google.code.gson', name: 'gson', version: '2.8.6'
    compile group: 'io.netty', name: 'netty-all', version: '4.1.29.Final'
    compile group: 'org.bouncycastle', name: 'bcpkix-jdk15on', version: '1.66'
}
```

编写业务代码

实现基本的Repository

```
import bdware.doip.sdk.server.Config;
import bdware.doip.sdk.server.ServiceInfo;
import bdware.doip.sdk.server.RocksDBStore;
import bdware.doip.sdk.server.Server;

public class Main {
    public static void main(String[] args) throws Exception {
        String confPath = "default.conf";
        Config.LocalConfig conf = new Config(confPath).Parse();
        if (conf.storage.equals("rocksdb")) {
            new Server(new ServiceInfo(conf.repoID, conf.ownerID, conf.type,
conf.listeners), new RocksDBStore());
        } else {
            System.out.println("unsupported database, exit!");
            System.exit(1);
        }
    }
}
```

实现基本的Registry

实现基本的Client

```
import bdware.doip.codec.bean.DigitalObject;
import bdware.doip.codec.bean.DoResponse;
import bdware.doip.codec.bean.DoType;
import bdware.doip.codec.bean.Element;
import bdware.doip.codec.message.DoMessage;
import bdware.doip.sdk.client.Client;
import bdware.doip.sdk.client.Config;

public class Main {
    static String testConf = "default_client.conf";
    static String repoId = "86.5000.470/local.TLSRepository";
    static Config conf;
```

```
    static {
        try {
            conf = Config.fromFile(testConf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    static Client client = new Client(conf.serverUrl, conf.messageFormat,
repoId);

    public static void main(String[] args) {
        String serverInfo = getServerInfo(repoId);
        System.out.println("server info:"+serverInfo);
        // pre-request: get do identifier from irp server

        if (put("86.5000.470/test.hello", "hello doa world".getBytes())) {
            System.out.println("create success");
        } else {
            System.out.println("create failed, exit");
        }

        System.out.println("contents for do:"+get("86.5000.470/test.hello"));
    }

    // hello
    public static String getServerInfo(String serverId) {
        DoMessage res = client.doipClient.hello(serverId);
        System.out.println("server info:"+new String(res.body));
        return new String(res.body);
    }

    // create
    public static boolean put(String doId, byte[] data) {
        String id = "1";
        String type = "text/plain";
        DigitalObject digitalObject = new DigitalObject(doId, DoType.DoString);
        Element element = new Element(id, type);
        element.setData(data);
        digitalObject.addElements(element);
        DoMessage res = client.doipClient.create(client.serverId,
digitalObject);
        System.out.println("put res:"+new String(res.body));
        return res.parameters.response == DoResponse.Success;
    }

    // retrieve
    public static String get(String doId) {
        DoMessage doMessage = client.doipClient.retrieve(doId, "1", "true");
        return new String(doMessage.body);
    }

    // update
    public static boolean update(String doId, byte[] data) {
        DigitalObject digitalObject = new DigitalObject(doId, DoType.DoString);
        String id = "1";
        String type = "text/plain";
```

```
        Element element = new Element(id, type);
        element.setData(data);
        digitalObject.addElements(element);
        DoMessage res = client.doipClient.update(digitalObject);
        System.out.println("update res:"+new String(res.body));
        return res.parameters.response == DoResponse.Success;
    }

    //delete
    public static boolean delete(String doId) {
        DoMessage res = client.doipClient.delete(doId);
        System.out.println("delete res:"+new String(res.body));
        return res.parameters.response == DoResponse.Success;
    }
    // search, listOp
}
```

配置信息准备

生成证书

由于客户端与服务端采用了TLS通信协议，需要配置密钥。请确保环境中安装有keytools工具，然后使用如下命令，生成服务端使用的私钥和证书。（TODO:开发keytool）

- 为Repository生成证书

```
keytool -genkey -keyalg RSA -keysize 2048 -validity 365 -keypass 123456 -
keystore doip_service_repository.keystore -storepass 123456 -dname
"UID=86.5000.470/doip.RepositoryTLSService"
```

- 为Registry生成证书

```
keytool -genkey -keyalg RSA -keysize 2048 -validity 365 -keypass 123456 -
keystore doip_service_registry.keystore -storepass 123456 -dname
"UID=86.5000.470/doip.RegistryTLSService"
```

编写配置文件

Repository

创建名称为default_repository.conf的文件，写入如下内容。

```
{
  "type": "Repository",
  "ownerID": "86.5000.470/dou.SUPER",
  "repoID": "86.5000.470/local.TLSRepository",
  "listeners": [
    {
      "url": "tls://127.0.0.1:1716",
      "protocolVersion": "2.0",
      "messageFormat": "delimiter"
    }
  ],
```

```
  "serviceDescription": "local tls repository",
  "serviceName": "DOA Repository",
  "keyFilePath": "./doip_service_repository.keystore",
  "keyPass": "123456",
  "storage": "rocksdb"
}
```

Registry

创建名称为default_registry.conf的文件，写入如下内容。

```
{
  "type": "Registry",
  "ownerID": "86.5000.470/dou.SUPER",
  "repoID": "86.5000.470/local.TLSRegistry",
  "listeners": [
    {
      "url": "tls://127.0.0.1:1717",
      "protocolVersion": "2.0",
      "messageFormat": "delimiter"
    }
  ],
  "serviceDescription": "local tls registry",
  "serviceName": "DOA Registry",
  "keyFilePath": "./doip_service_registry.keystore",
  "keyPass": "123456",
  "storage": "rocksdb"
}
```

Client

创建名称为default_client.conf的文件，写入如下内容。

```
{
  "repository": {
    "serverUrl": "tls://127.0.0.1:1716",
    "messageFormat": "delimiter",
    "keyFilePath": "./default_client_repository.keystore",
    "keyPass": "123456"
  },
  "registry": {
    "serverUrl": "tls://127.0.0.1:1717",
    "messageFormat": "delimiter",
    "keyFilePath": "./default_client_registry.keystore",
    "keyPass": "123456"
  }
}
```

IRPPoxy SDK

IRPProxy SDK是基于CNRI提供的Handle Client Java lib库对IRP协议所支持的操作进行了封装。该lib库对IRP协议进行了实现，我们在该lib库的基础上对DOA架构中的Handle System进行了实现，并提供对DONA旗下国内外多个MPA相关Handle System系统的接入与兼容，包括CNRI(HDL)、中科院(DataPid)、中数等。用户(开发者)可以直接基于IRPProxy SDK对Handle标识进行解析与管理，也可以

通过DOA的Client对DO进行标识管理。

IRPProxy SDK接口文档

IRPProxy SDK API Javadoc

IRPProxy SDK使用说明

将下载好的IRPProxy SDK Jar包直接导入项目工程的libs文件夹下，然后通过package的方式进行调用，详见下文调用示例。

IRPProxy SDK调用示例

```
package pku.doa.sample;

import net.handle.hdllib.*;
import pku.doa.proxy.HandleOperator;

public class Test {
    public static void main(String[] args) throws Exception {
        HandleOperator.resolve("47.102.45.68","86.5000.470/do.hello");

 //HandleOperator.remove("47.102.45.68","20.500.12750/Test_Handle",602,"/Users/vincent/Desktop/DOA实现/admpriv.bin","0.NA/20.500.12750",300);

        HandleValue[] valuesToCreate = {
                new HandleValue(1, Common.STD_TYPE_URL,
Util.encodeString("http://handle.net/")), new HandleValue(2,
Common.STD_TYPE_EMAIL, Util.encodeString("hdladmin@cnri.reston.va.us")),
                new HandleValue(3, Common.STD_TYPE_HSADMIN,
Encoder.encodeAdminRecord(new
AdminRecord(Util.encodeString("0.NA/20.500.12750"), 200, true, true, true, true,
true, true, true, true, true, true, true, true))),
        };

 HandleOperator.create("47.102.45.68","20.500.12750/Create_Test_Handle0","D://Desktop/DOA/IRP Handle//admpriv.bin","0.NA/20.500.12750",300,valuesToCreate);

        HandleValue value = new HandleValue(500, "URL",
"http://www.test5001.com");
        HandleValue[] valuesToAdd = new HandleValue[]{value};

 HandleOperator.add("47.102.45.68","20.500.12750/Create_Test_Handle0","D://Desktop/DOA/IRP Handle//admpriv.bin","0.NA/20.500.12750",300,valuesToAdd);


 HandleOperator.modify("47.102.45.68","20.500.12750/Create_Test_Handle0","D://Desktop/DOA/IRP Handle//admpriv.bin","0.NA/20.500.12750",300,valuesToAdd);


 HandleOperator.delete("47.102.45.68","20.500.12750/Create_Test_Handle0","D://Desktop/DOA/IRP Handle//admpriv.bin","0.NA/20.500.12750",300);

    }
}
```

部署程序

- 将编译好的Repository代码与default_server_repository.keystore，default_repository.conf放在同一目录，然后执行Repository。
- 将编译好的Registry代码与default_server_registry.keystore，default_registry.conf放在同一目录，然后执行Registry。
- 将编译好的Client代码与default_client_repository.keystore，default_client_registry.keystore，default_client.conf放在同一目录，然后执行Client。

进一步阅读

实现自己的应用

基于Repository，Registry和Client构建自己的应用。

设计元数据格式

设计Registry的检索功能

设计多Registry的联合工作功能

DOIP协议

通过openssl工具测试DOIP协议

接口说明

详细的SDK接口说明文档

查看源代码

- doip-java-sdk-src
  v0.1
- doa-irpproxy-sdk-src
  v0.1