Thrift Tutorial

Release 1.0

Stratos Dimopoulos

Jun 19, 2017

Contents

The purpose of this document is to describe step by step Thrift's installation and give simple examples of its usage.

CHAPTER 1

Introduction to Thrift

Thrift is a lightweight, language-independent software stack with an associated code generation mechanism for RPC. Thrift provides clean abstractions for data transport, data serialization, and application level processing. Thrift was originally developed by Facebook and now it is open sourced as an Apache project. Apache Thrift is a set of code-generation tools that allows developers to build RPC clients and servers by just defining the data types and service interfaces in a simple definition file. Given this file as an input, code is generated to build RPC clients and servers that communicate seamlessly across programming languages.

In this tutorial I will describe how Thrift works and provide a guide for build and installation steps, how to write thrift files and how to generate from those files the source code that can be used from different client libraries to communicate with the server. Thrift supports a variety of languages including C++, Java, Python, PHP, Ruby but for simplicity I will focus this tutorial on examples that include Java and Python.

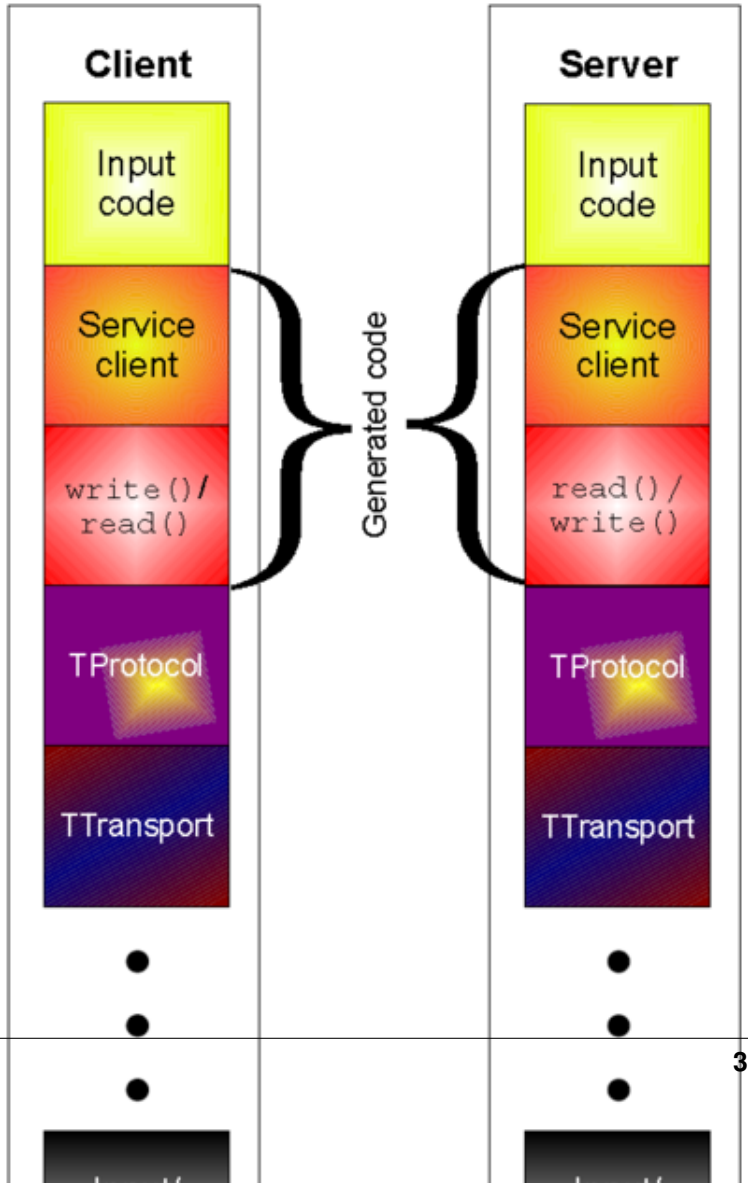


Fig. 1.2: Services using Thrift

CHAPTER 2

Thrift installation

Detailed information on how to install Thrift can be found here: http://thrift.apache.org/docs/install/

On Ubuntu Linux for example you just need to first install the dependencies and then you are ready to install Thrift.

1. **Install the languages with which you plan to use thrift. To use with Java for example, install a Java JDK you prefer. I**

    - To use with Java you will also need to install Apache Ant

    ```
    sudo apt-get install ant
    ```

2. Installing required tools and libraries:

```
sudo apt-get install libboost-dev libboost-test-dev libboost-program-
↪options-dev libboost-filesystem-dev libboost-thread-dev libevent-dev␣
↪automake libtool flex bison pkg-config g++ libssl-dev
```

- You can check for specific requirements for each language you wish to use here: http://thrift.apache.org/docs/install/

1. Download Thrift: http://thrift.apache.org/download
2. Copy the downloaded file into the desired directory and untar the file

```
tar -xvf thrift-0.9.3.tar.gz
```

3. For detailed instuctions on how to build Apache Thrift on your specific system you can read here: http://thrift.apache.org/docs/BuildingFromSource

    - For an Ubuntu linux distribution you just need to go to the thrift directory and type:

    ```
    ./bootstrap.sh
    ./configure
    ```

    - At the end of the output you should be able to see a list of all the libraries that are currently built in your system and ready to use with your desired programming languages. If a component is missing you should download the missing language and repeat the above step.

```
thrift 0.9.3

Building C++ Library ......... : yes
Building C (GLib) Library .... : no
Building Java Library ........ : yes
Building C# Library .......... : no
Building Python Library ...... : yes
Building Ruby Library ........ : no
Building Haskell Library ..... : no
Building Perl Library ........ : no
Building PHP Library ......... : no
Building Erlang Library ...... : no
Building Go Library .......... : no
Building D Library ........... : no

C++ Library:
   Build TZlibTransport ...... : yes
   Build TNonblockingServer .. : yes
   Build TQTcpServer (Qt) .... : no

Java Library:
   Using javac ............... : javac
   Using java ................ : java
   Using ant ................. : /usr/bin/ant

Python Library:
   Using Python .............. : /usr/bin/python
```

- Here http://thrift.apache.org/docs/install/debian/ you can find all the packages you might need to support your desired language in case some of them are missing.
- On the same directory run make to build Thrift

```
sudo make
```

- (Optional) Run the test suite if you want

```
sudo make check
```

- And finally you are ready to install Thrift by running

```
sudo make install
```

Verify installation

Now your Thrift installation is completed! To verify that you have succesfully installed Thrift just type

```
thrift -version
```

and you should be able to see something like the following:

```
Thrift version 0.9.0
```

Additionaly you can go to the tutorial directory and follow the instructions located on the README files on each of the targeted languages directories. For example for JAVA you should be able to verify the following:

```
thrift/tutorial/java$ file ../../lib/java/build/libthrift-${version}-${release}.jar
../../lib/java/libthrift.jar: Zip archive data, at least v1.0 to extract

thrift/tutorial/java$ file ../../compiler/cpp/thrift
../../compiler/cpp/thrift: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),␣
↪dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped

thrift/tutorial/java$ ls ../../lib/java/build/lib/
commons-lang-2.5.jar  junit-4.4.jar  servlet-api-2.5.jar  slf4j-api-1.5.8.jar  slf4j-
↪simple-1.5.8.jar
```

CHAPTER 3

Thrift protocol stack

To understand Thrift's protocol stack I highly recommend to take a look at the white-paper: http://thrift.apache.org/static/files/thrift-20070401.pdf and to the Thrift Architecture part of this very nice description here: http://jnb.ociweb.com/jnb/jnbJun2009.html - Part of the following content is directly derived from these sources. I will now briefly present Thrift's architecture.

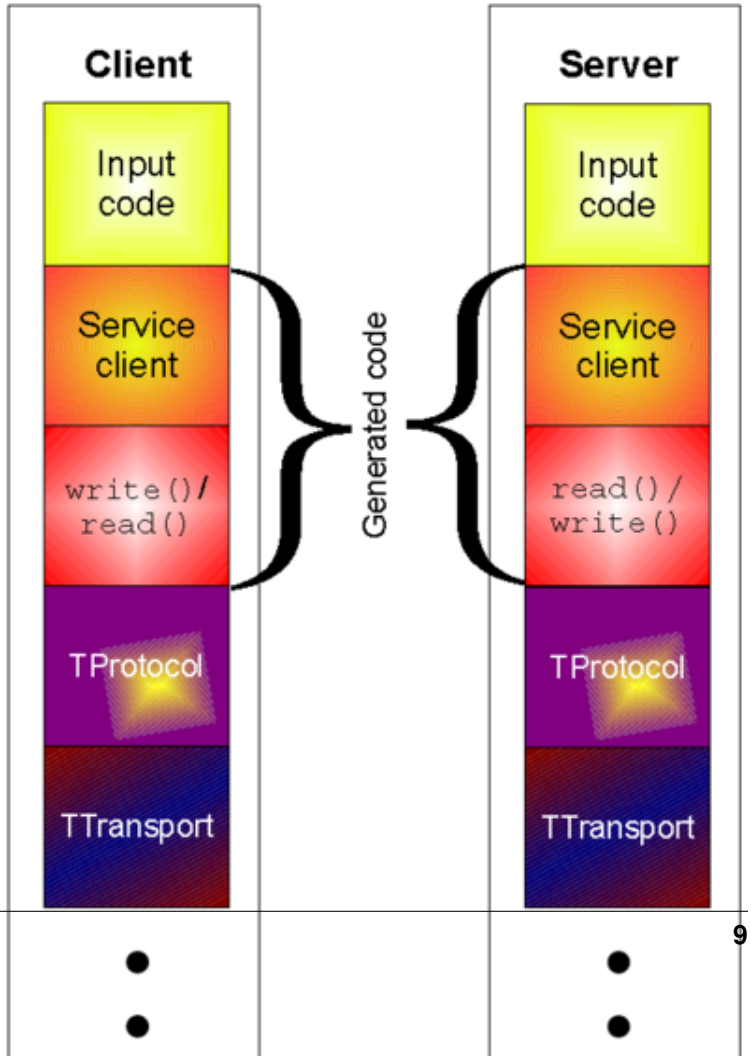
Runtime Library

The protocol and transport layer are part of the runtime library. This means that it is possible to define a service and change the protocol and transport without recompiling the code.

Protocol Layer

The protocol layer provides serialization and deserialization. Thrift supports the following protocols :

- TBinaryProtocol - A straight-forward binary format encoding numeric values as binary, rather than converting to text.
- TCompactProtocol - Very efficient, dense encoding of data (See details below).
- TDenseProtocol - Similar to TCompactProtocol but strips off the meta



information from what is transmitted, and adds it back in at the receiver. TDenseProtocol is still experimental and not yet available in the Java implementation.

- TJSONProtocol - Uses JSON for encoding of data.
- TSimpleJSONProtocol - A write-only protocol using JSON. Suitable for parsing by scripting languages
- TDebugProtocol - Uses a human-readable text format to aid in debugging.

Tranport Layer

The transport layer is responsible for reading from and writing to the wire. Thrift supports the following:

- TSocket - Uses blocking socket I/O for transport.
- TFramedTransport - Sends data in frames, where each frame is preceded by a length. This transport is required when using a non-blocking server.
- TFileTransport - This transport writes to a file. While this transport is not included with the Java implementation, it should be simple enough to implement.
- TMemoryTransport - Uses memory for I/O. The Java implementation uses a simple ByteArrayOutputStream internally.
- TZlibTransport - Performs compression using zlib. Used in conjunction with another transport. Not available in the Java implementation.

Processor

The processor takes as arguments an input and an output protocol. Reads data from the input, processes the data throught the Handler specified by the user and then writes the data to the output.

Supported Servers

A server will be listening for connections to a port and will send the data it receives to the Processor to handle.

- TSimpleServer - A single-threaded server using std blocking io. Useful for testing.
- TThreadPoolServer - A multi-threaded server using std blocking io.
- TNonblockingServer - A multi-threaded server using non-blocking io (Java implementation uses NIO channels). TFramedTransport must be used with this server.

CHAPTER 4

Thrift type system

The thrift type system includes base types like bool, byte, double, string and integer but also special types like binary and it also supports structs (equivalent to classes but without inheritance) and also containers (list, set, map) that correspond to commonly available interfaces in most programming languages. The type system focuses on key types available in all programming languages and omits types that are specific to only some programming languages.

- A detailed reference to Thrift type system follows next and more details can be found here: http://thrift.apache.org/docs/types
- If you want to check the Thrift interface description language that allows for the definition of Thrift types you can read here: http://thrift.apache.org/docs/idl

Base types

- **bool**: A boolean value (true or false)
- **byte**: An 8-bit signed integer
- **i16**: A 16-bit signed integer
- **i32**: A 32-bit signed integer
- **i64**: A 64-bit signed integer
- **double**: A 64-bit floating point number
- **string**: A text string encoded using UTF-8 encoding

Note: There is no support for unsigned integer types, due to the fact that there are no native unsigned integer types in many programming languages. Signed integers can be safely cast to their unsigned counterparts when necessary.

Special Types

binary: a sequence of unencoded bytes

Structs

A struct has a set of strongly typed fields, each with a unique name identifier. The look very similar to C-like structs.

```
struct Example {
  1:i32 number=10,
  2:i64 bigNumber,
  3:double decimals,
  4:string name="thrifty"
}
```

Containers

- **list** (Maps to c++ STL vector, Java ArrayList etc)
- **set** (Maps to an STL set, Java HashSet etc)
  - PHP doesn't support sets - so it is treated similar to a List map
- **map** (Maps to an STL map, Java HashMap etc)

All the above are the defaults but can be customized to correspond to different types of any language. For this reason custom code generation directives have been added.

Exceptions

They inherit from the native exception base class as appropriate in each target programming language.

```
exception InvalidOperation {
1: i32 what,
2: string why
}
```

Services

A service consists of a set of named functions, each with a list of parameters and a return type. It is semantically equivalent to defining an interface or a pure virtual abstract class.

```
service <name> {
<returntype> <name>(<arguments>)
[throws (<exceptions>)]
...
}

An example:
service StringCache {
void set(1:i32 key, 2:string value),
string get(1:i32 key) throws (1:KeyNotFound knf),
void delete(1:i32 key)
}
```

CHAPTER 5

Writing a .thrift file

In a .thrift file you can define the services that your server will implement and that they will be called by any clients. The Thrift compiler will read this file and generate source code to be used from the servers and clients you will write.

A simple .thrift file with which we defined our simple multiplication service for this demo looks like this:

```
namespace java tutorial
namespace py tutorial

/*
 C like comments are supported
*/
// This is also a valid comment

typedef i32 int // We can use typedef to get pretty names for the types we␣
↪are using
service MultiplicationService
{
        int multiply(1:int n1, 2:int n2),
}
```

In the above file we just define a service to multiply two numbers and return their product to avoid making our first demo hard to understand. If you named this file multiplication.thrift and you want to use java and python all you need to do is to run

```
thrift --gen java multiplication.thrift
thrift --gen py multiplication.thrift
```

Thrift will now generate code for you and place it in the gen-java and gen-py directories respectively. Make sure you have sufficient rights to write inside the directory. Otherwise you might need to run the above commands as sudo user.

- Notice in the above file the namespaces we define. They dictate that Thrift should generate a sub-directory named tutorial inside gen-java and gen-py and place the output files there. We could have specified different namespaces for java and python. We could also omit specifying a namespace. In the latter case the files would be place directly inside the gen-java and gen-py directories.

- Notice also that we can typedef Thrift types with something more easy to remember (i32 with int in this case)

A more detailed example of how you can specify structs, exceptions and other thrift language constructs inside your definition file is specified on tutorial.thrift file that comes along with your Thrift installation. The file contains descriptive comments and it is self-explanatory. Below we notice its most importand parts.

Include other .thrift files

Included objects are accessed using the name of the .thrift file as a prefix. i.e. shared.SharedObject

```
include "shared.thrift"
```

Define C-style enumerations

Values are optional and start at 1 if not specified.

```
enum Operation {
  ADD = 1,
  SUBTRACT = 2,
  MULTIPLY = 3,
  DIVIDE = 4
}
```

Define Structs

Fields can be declared "optional", which ensures they will not be included in the serialized output if they aren't set - This requires some manual management in some languages. Do you notice the numbers before each field type? Fields identifiers is the way Thrift performs versioning. The Thrift definition language supports automatic assignment of field identifiers, but it is good programming practice to always explicitly specify field identifiers. As explained in Thrift's whitepaper *To avoid conflicts between manually and automatically assigned identifiers, fields with identifiers omitted are assigned identifiers decrementing from -1, and the language only supports the manual assignment of positive identifiers.*

```
struct Work {
  1: i32 num1 = 0,
  2: i32 num2,
  3: Operation op,
  4: optional string comment,
}
```

Define Exceptions

```
exception InvalidOperation {
  1: i32 what,
  2: string why
}
```

Define a Service

```
service Calculator extends shared.SharedService {

   void ping(),

   i32 add(1:i32 num1, 2:i32 num2),

   i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperation ouch),

   /**
    * This method has a oneway modifier. That means the client only makes
    * a request and does not listen for any response at all. Oneway methods
    * must be void.
    */
   oneway void zip()

}
```

CHAPTER 6

Thrift by Example

Generating code with Thrift

After creating a .thrift file (See *Writing a .thrift file*) you are now ready to run Thrift in order to generate code in your targeted languages. The usage of thrift command is:

```
thrift [options] file
```

For example if you want to generate code for java you should type:

```
thrift -r --gen java filame.thrift
```

The above command will generate a directory gen-java which will contain all the code generated by Thrift in java. The option -r is to specify that I want to generate recursively code for potential includes on our .thrift file. After generating code you are now ready to create your client and server code and use the Thrift generated code to do the hard work for you, as you will se below.

Notice: When trying to run the python server you may encounter the error:

```
ImportError: No module named Thrift
```

This can be easily fixed by going to the directory lib/py on the thrift installation directory and run the following command to install the thrift module to your python libraries:

```
sudo python setup.py install
```

A Simple Example to warm-up

On this example I will demonstrate the creation of a simple multiplication service.

- Lets create first the .thrift definition of our service. The .thrift file I am going to use is the same you saw in the *Writing a .thrift file* and is as shown below:

```
namespace java tutorial
namespace py tutorial

typedef i32 int // We can use typedef to get pretty names for the types we are␣
↪using
service MultiplicationService
{
        int multiply(1:int n1, 2:int n2),
}
```

- Name this file multiple.thrift and then run the below commands to generate code for java and python

```
thrift --gen java multiple.thrift
thrift --gen py multiple.thrift
```

After running the commands Thrift should generate code inside the directories gen-java/tutorial and gen-py/tutorial for Java and Python respectively. `Remember to use sudo in case the directories are not created!` It would be useful to take a look on this code to get a better understanding of what code Thrift generates for you. You can find a short explanation of the code here: *Thrift's auto-generated code for the multiplication service example*

- Now we are ready to write our own code. Lets first write some Java code for our client and server and then we will also write a Python client to send requests to our server. We will not need to change anything on the server part to do this!

Multiplication Handler

Here is the MultiplicationHandler class in which I implement the interface specified before in our multi.thrift definition and for which Thrift had already generated code.

```
import org.apache.thrift.TException;

public class MultiplicationHandler implements MultiplicationService.Iface {

        @Override
         public int multiply(int n1, int n2) throws TException {
            System.out.println("Multiply(" + n1 + "," + n2 + ")");
            return n1 * n2;
         }


}
```

Java Multiplication Server

Here is the MultiplicationServer class in which I have implemented a simple server. With slightly different configuration this server can become secure as you will see in the next example. The only thing worth messioning about the server implementation is the usage of the Processor class that is auto-generated by Thrift. The Processor does two simple things. Reads data from an input stream and writes data to an output stream. The processor reads data from the input, processes the data (actually uses the handler specified by the user to process the data) and the writes the processed data to the output. Finally, I should mention that for this example I used the simple server implemention, but it would be as easy to use any of the implementations offered by thrift (threadPoolServer or nonBlockingServer).

```
import org.apache.thrift.server.TServer;
import org.apache.thrift.server.TServer.Args;
import org.apache.thrift.server.TSimpleServer;
import org.apache.thrift.transport.TServerSocket;
import org.apache.thrift.transport.TServerTransport;

public class MultiplicationServer {

  public static MultiplicationHandler handler;

  public static MultiplicationService.Processor processor;

  public static void main(String [] args) {
    try {
      handler = new MultiplicationHandler();
      processor = new MultiplicationService.Processor(handler);

      Runnable simple = new Runnable() {
        public void run() {
          simple(processor);
        }
      };

      new Thread(simple).start();
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  public static void simple(MultiplicationService.Processor processor) {
    try {
      TServerTransport serverTransport = new TServerSocket(9090);
      TServer server = new TSimpleServer(new Args(serverTransport).
↪processor(processor));

      System.out.println("Starting the simple server...");
      server.serve();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

Java Multiplication Client

Important things to notice on the client code is the use of the TBinaryProtocol for the serializitaion and deserialization part. I could also use the compact, the JSON protocol or any other protocol supported by thrift. For more details on the protocols you can use please refer to the *Thrift protocol stack* section of this tutorial. Another importand thing to notice is the use of the client and the coresponding client.multiply() method provided to us by the auto-generated thrift code. This call behind the scenes calls the TServiceClient.sendBase() method that will write the data to the wire.

```
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.protocol.TProtocol;
import org.apache.thrift.transport.TSocket;
```

```
import org.apache.thrift.transport.TTransport;

public class MultiplicationClient {
  public static void main(String [] args) {


    try {
      TTransport transport;

      transport = new TSocket("localhost", 9090);
      transport.open();

      TProtocol protocol = new  TBinaryProtocol(transport);
      MultiplicationService.Client client = new MultiplicationService.
↪Client(protocol);

      perform(client);

      transport.close();
    } catch (TException x) {
      x.printStackTrace();
    }
  }

  private static void perform(MultiplicationService.Client client) throws␣
↪TException
  {

    int product = client.multiply(3,5);
    System.out.println("3*5=" + product);
  }
}
```

Python Multiplication Client

The python client implements anything as discussed for the java client. The language syntax is the only thing I had to change on my approach.

```
#!/usr/bin/env python

import sys
sys.path.append('../gen-py')

from tutorial import MultiplicationService
from tutorial.ttypes import *

from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol

try:

  # Make socket
  transport = TSocket.TSocket('localhost', 9090)
```

```
  # Buffering is critical. Raw sockets are very slow
  transport = TTransport.TBufferedTransport(transport)

  # Wrap in a protocol
  protocol = TBinaryProtocol.TBinaryProtocol(transport)

  # Create a client to use the protocol encoder
  client = MultiplicationService.Client(protocol)

  # Connect!
  transport.open()

  product = client.multiply(4,5)
  print '4*5=%d' % (product)

  # Close!
  transport.close()

except Thrift.TException, tx:
  print '%s' % (tx.message)
```

A More Complicated Example

Now you are ready to see something a little bit more complicated. The bellow example describes the Thrift tutorial code included in the Thrift installation directory. The .thrift file for the calculator service provided on the Thrift installation, also servers the role of a short Thrift documentation. For this reason, I have deleted part of the comments on the file to avoid repeating things that I have already discussed. Follow the comments to understand how you can use thrift to generate code for more complicated services. Notice that this time code will be generated by Thrift not only for the Calculator but also for the exception, the enum Operation, the tutorialConstants and the struct Work. Of course the code for those files is much simpler as expected.

```
include "shared.thrift"

namespace cpp tutorial
namespace d tutorial
namespace java tutorial
namespace php tutorial
namespace perl tutorial

/**
 * Thrift also lets you define constants for use across languages. Complex
 * types and structs are specified using JSON notation.
 */
const i32 INT32CONSTANT = 9853
const map<string,string> MAPCONSTANT = {'hello':'world', 'goodnight':'moon'}

/**
 * You can define enums, which are just 32 bit integers. Values are optional
 * and start at 1 if not supplied, C style again.
 */
enum Operation {
  ADD = 1,
  SUBTRACT = 2,
  MULTIPLY = 3,
```

```
  DIVIDE = 4
}

/**
 * Structs are the basic complex data structures. They are comprised of␣
↪fields
 * which each have an integer identifier, a type, a symbolic name, and an
 * optional default value.
 *
 * Fields can be declared "optional", which ensures they will not be included
 * in the serialized output if they aren't set.  Note that this requires some
 * manual management in some languages.
 */
struct Work {
  1: i32 num1 = 0,
  2: i32 num2,
  3: Operation op,
  4: optional string comment,
}

/**
 * Structs can also be exceptions, if they are nasty.
 */
exception InvalidOperation {
  1: i32 what,
  2: string why
}

/**
 * Ahh, now onto the cool part, defining a service. Services just need a name
 * and can optionally inherit from another service using the extends keyword.
 */
service Calculator extends shared.SharedService {

  /**
   * A method definition looks like C code. It has a return type, arguments,
   * and optionally a list of exceptions that it may throw. Note that␣
↪argument
   * lists and exception lists are specified using the exact same syntax as
   * field lists in struct or exception definitions.
   */

   void ping(),

   i32 add(1:i32 num1, 2:i32 num2),

   i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperation ouch),

   /**
    * This method has a oneway modifier. That means the client only makes
    * a request and does not listen for any response at all. Oneway methods
    * must be void.
    */
   oneway void zip()

}
```

Calculator Service Handler

Nothing really changed here compared to the simple example you just saw. There are just more method interfaces implemented.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

import org.apache.thrift.TException;

// Generated code
import tutorial.*;
import shared.*;

import java.util.HashMap;

public class CalculatorHandler implements Calculator.Iface {

  private HashMap<Integer,SharedStruct> log;

  public CalculatorHandler() {
    log = new HashMap<Integer, SharedStruct>();
  }

  public void ping() {
    System.out.println("ping()");
  }

  public int add(int n1, int n2) {
    System.out.println("add(" + n1 + "," + n2 + ")");
    return n1 + n2;
  }

  public int calculate(int logid, Work work) throws InvalidOperation {
    System.out.println("calculate(" + logid + ", {" + work.op + "," + work.num1 + ","␣
↪+ work.num2 + "})");
    int val = 0;
    switch (work.op) {
    case ADD:
      val = work.num1 + work.num2;
      break;
    case SUBTRACT:
```

```
      val = work.num1 - work.num2;
      break;
    case MULTIPLY:
      val = work.num1 * work.num2;
      break;
    case DIVIDE:
      if (work.num2 == 0) {
        InvalidOperation io = new InvalidOperation();
        io.what = work.op.getValue();
        io.why = "Cannot divide by 0";
        throw io;
      }
      val = work.num1 / work.num2;
      break;
    default:
      InvalidOperation io = new InvalidOperation();
      io.what = work.op.getValue();
      io.why = "Unknown operation";
      throw io;
    }

    SharedStruct entry = new SharedStruct();
    entry.key = logid;
    entry.value = Integer.toString(val);
    log.put(logid, entry);

    return val;
  }

  public SharedStruct getStruct(int key) {
    System.out.println("getStruct(" + key + ")");
    return log.get(key);
  }

  public void zip() {
    System.out.println("zip()");
  }

}
```

Java Calculator Server

Most of the code for the server remains the same as above. What is added here is the option of a secure server that you can use to provide authenticated service. With comments you can also see what the code for a multi threaded server looks like.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
```

```
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

import org.apache.thrift.server.TServer;
import org.apache.thrift.server.TServer.Args;
import org.apache.thrift.server.TSimpleServer;
import org.apache.thrift.server.TThreadPoolServer;
import org.apache.thrift.transport.TSSLTransportFactory;
import org.apache.thrift.transport.TServerSocket;
import org.apache.thrift.transport.TServerTransport;
import org.apache.thrift.transport.TSSLTransportFactory.TSSLTransportParameters;

// Generated code
import tutorial.*;
import shared.*;

import java.util.HashMap;

public class JavaServer {

  public static CalculatorHandler handler;

  public static Calculator.Processor processor;

  public static void main(String [] args) {
    try {
      handler = new CalculatorHandler();
      processor = new Calculator.Processor(handler);

      Runnable simple = new Runnable() {
        public void run() {
          simple(processor);
        }
      };
      Runnable secure = new Runnable() {
        public void run() {
          secure(processor);
        }
      };

      new Thread(simple).start();
      new Thread(secure).start();
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  public static void simple(Calculator.Processor processor) {
    try {
      TServerTransport serverTransport = new TServerSocket(9090);
      TServer server = new TSimpleServer(new Args(serverTransport).
↪processor(processor));
```

```
      // Use this for a multithreaded server
      // TServer server = new TThreadPoolServer(new TThreadPoolServer.
↪Args(serverTransport).processor(processor));

      System.out.println("Starting the simple server...");
      server.serve();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void secure(Calculator.Processor processor) {
    try {
      /*
       * Use TSSLTransportParameters to setup the required SSL parameters. In this␣
↪example
       * we are setting the keystore and the keystore password. Other things like␣
↪algorithms,
       * cipher suites, client auth etc can be set.
       */
      TSSLTransportParameters params = new TSSLTransportParameters();
      // The Keystore contains the private key
      params.setKeyStore("../../lib/java/test/.keystore", "thrift", null, null);

      /*
       * Use any of the TSSLTransportFactory to get a server transport with the␣
↪appropriate
       * SSL configuration. You can use the default settings if properties are set in␣
↪the command line.
       * Ex: -Djavax.net.ssl.keyStore=.keystore and -Djavax.net.ssl.
↪keyStorePassword=thrift
       *
       * Note: You need not explicitly call open(). The underlying server socket is␣
↪bound on return
       * from the factory class.
       */
      TServerTransport serverTransport = TSSLTransportFactory.getServerSocket(9091, 0,
↪ null, params);
      TServer server = new TSimpleServer(new Args(serverTransport).
↪processor(processor));

      // Use this for a multi threaded server
      // TServer server = new TThreadPoolServer(new TThreadPoolServer.
↪Args(serverTransport).processor(processor));

      System.out.println("Starting the secure server...");
      server.serve();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Java Calculator Client

But for the option given to the user to choose between the simple and the secure server nothing changed to the code of the client as well. We simpe call more operations with some additional calls.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

// Generated code
import tutorial.*;
import shared.*;

import org.apache.thrift.TException;
import org.apache.thrift.transport.TSSLTransportFactory;
import org.apache.thrift.transport.TTransport;
import org.apache.thrift.transport.TSocket;
import org.apache.thrift.transport.TSSLTransportFactory.TSSLTransportParameters;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.protocol.TProtocol;

public class JavaClient {
  public static void main(String [] args) {

    if (args.length != 1) {
      System.out.println("Please enter 'simple' or 'secure'");
      System.exit(0);
    }

    try {
      TTransport transport;
      if (args[0].contains("simple")) {
        transport = new TSocket("localhost", 9090);
        transport.open();
      }
      else {
        /*
         * Similar to the server, you can use the parameters to setup client␣
↪parameters or
         * use the default settings. On the client side, you will need a TrustStore␣
↪which
         * contains the trusted certificate along with the public key.
         * For this example it's a self-signed cert.
```

```
         */
        TSSLTransportParameters params = new TSSLTransportParameters();
        params.setTrustStore("../../lib/java/test/.truststore", "thrift", "SunX509",
↪"JKS");
        /*
         * Get a client transport instead of a server transport. The connection is␣
↪opened on
         * invocation of the factory method, no need to specifically call open()
         */
        transport = TSSLTransportFactory.getClientSocket("localhost", 9091, 0,␣
↪params);
      }

      TProtocol protocol = new  TBinaryProtocol(transport);
      Calculator.Client client = new Calculator.Client(protocol);

      perform(client);

      transport.close();
    } catch (TException x) {
      x.printStackTrace();
    }
  }

  private static void perform(Calculator.Client client) throws TException
  {
    client.ping();
    System.out.println("ping()");

    int sum = client.add(1,1);
    System.out.println("1+1=" + sum);

    Work work = new Work();

    work.op = Operation.DIVIDE;
    work.num1 = 1;
    work.num2 = 0;
    try {
      int quotient = client.calculate(1, work);
      System.out.println("Whoa we can divide by 0");
    } catch (InvalidOperation io) {
      System.out.println("Invalid operation: " + io.why);
    }

    work.op = Operation.SUBTRACT;
    work.num1 = 15;
    work.num2 = 10;
    try {
      int diff = client.calculate(1, work);
      System.out.println("15-10=" + diff);
    } catch (InvalidOperation io) {
      System.out.println("Invalid operation: " + io.why);
    }

    SharedStruct log = client.getStruct(1);
    System.out.println("Check log: " + log.value);
  }
}
```

Thrift's auto-generated code for the multiplication service example

```
/**
 * Autogenerated by Thrift Compiler (0.9.0)
 *
 * DO NOT EDIT UNLESS YOU ARE SURE THAT YOU KNOW WHAT YOU ARE DOING
 *  @generated
 */
import java.util.BitSet;
import java.util.Collections;
import java.util.EnumMap;
import java.util.EnumSet;
import java.util.HashMap;
import java.util.Map;

import org.apache.thrift.EncodingUtils;
import org.apache.thrift.protocol.TTupleProtocol;
import org.apache.thrift.scheme.IScheme;
import org.apache.thrift.scheme.SchemeFactory;
import org.apache.thrift.scheme.StandardScheme;
import org.apache.thrift.scheme.TupleScheme;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MultiplicationService {

  public interface Iface {

    public int multiply(int n1, int n2) throws org.apache.thrift.TException;

  }

  public interface AsyncIface {

    public void multiply(int n1, int n2, org.apache.thrift.async.
↪AsyncMethodCallback<AsyncClient.multiply_call> resultHandler) throws org.
↪apache.thrift.TException;

  }

  public static class Client extends org.apache.thrift.TServiceClient␣
↪implements Iface {
    public static class Factory implements org.apache.thrift.
↪TServiceClientFactory<Client> {
      public Factory() {}
      public Client getClient(org.apache.thrift.protocol.TProtocol prot) {
        return new Client(prot);
      }
      public Client getClient(org.apache.thrift.protocol.TProtocol iprot,␣
↪org.apache.thrift.protocol.TProtocol oprot) {
        return new Client(iprot, oprot);
      }
    }

    public Client(org.apache.thrift.protocol.TProtocol prot)
    {
      super(prot, prot);
    }
```

```
    public Client(org.apache.thrift.protocol.TProtocol iprot, org.apache.
↪thrift.protocol.TProtocol oprot) {
      super(iprot, oprot);
    }

    public int multiply(int n1, int n2) throws org.apache.thrift.TException
    {
      send_multiply(n1, n2);
      return recv_multiply();
    }

    public void send_multiply(int n1, int n2) throws org.apache.thrift.
↪TException
    {
      multiply_args args = new multiply_args();
      args.setN1(n1);
      args.setN2(n2);
      sendBase("multiply", args);
    }

    public int recv_multiply() throws org.apache.thrift.TException
    {
      multiply_result result = new multiply_result();
      receiveBase(result, "multiply");
      if (result.isSetSuccess()) {
        return result.success;
      }
      throw new org.apache.thrift.TApplicationException(org.apache.thrift.
↪TApplicationException.MISSING_RESULT, "multiply failed: unknown result");
    }

  }
  public static class AsyncClient extends org.apache.thrift.async.
↪TAsyncClient implements AsyncIface {
    public static class Factory implements org.apache.thrift.async.
↪TAsyncClientFactory<AsyncClient> {
      private org.apache.thrift.async.TAsyncClientManager clientManager;
      private org.apache.thrift.protocol.TProtocolFactory protocolFactory;
      public Factory(org.apache.thrift.async.TAsyncClientManager␣
↪clientManager, org.apache.thrift.protocol.TProtocolFactory␣
↪protocolFactory) {
        this.clientManager = clientManager;
        this.protocolFactory = protocolFactory;
      }
      public AsyncClient getAsyncClient(org.apache.thrift.transport.
↪TNonblockingTransport transport) {
        return new AsyncClient(protocolFactory, clientManager, transport);
      }
    }

    public AsyncClient(org.apache.thrift.protocol.TProtocolFactory␣
↪protocolFactory, org.apache.thrift.async.TAsyncClientManager clientManager,
↪ org.apache.thrift.transport.TNonblockingTransport transport) {
      super(protocolFactory, clientManager, transport);
    }

    public void multiply(int n1, int n2, org.apache.thrift.async.
↪AsyncMethodCallback<multiply_call> resultHandler) throws org.apache.thrift.
↪TException {
```

```
      checkReady();
      multiply_call method_call = new multiply_call(n1, n2, resultHandler,␣
↪this, ___protocolFactory, ___transport);
      this.___currentMethod = method_call;
      ___manager.call(method_call);
    }

    public static class multiply_call extends org.apache.thrift.async.
↪TAsyncMethodCall {
      private int n1;
      private int n2;
      public multiply_call(int n1, int n2, org.apache.thrift.async.
↪AsyncMethodCallback<multiply_call> resultHandler, org.apache.thrift.async.
↪TAsyncClient client, org.apache.thrift.protocol.TProtocolFactory␣
↪protocolFactory, org.apache.thrift.transport.TNonblockingTransport␣
↪transport) throws org.apache.thrift.TException {
        super(client, protocolFactory, transport, resultHandler, false);
        this.n1 = n1;
        this.n2 = n2;
      }

      public void write_args(org.apache.thrift.protocol.TProtocol prot)␣
↪throws org.apache.thrift.TException {
        prot.writeMessageBegin(new org.apache.thrift.protocol.TMessage(
↪"multiply", org.apache.thrift.protocol.TMessageType.CALL, 0));
        multiply_args args = new multiply_args();
        args.setN1(n1);
        args.setN2(n2);
        args.write(prot);
        prot.writeMessageEnd();
      }

      public int getResult() throws org.apache.thrift.TException {
        if (getState() != org.apache.thrift.async.TAsyncMethodCall.State.
↪RESPONSE_READ) {
          throw new IllegalStateException("Method call not finished!");
        }
        org.apache.thrift.transport.TMemoryInputTransport memoryTransport =␣
↪new org.apache.thrift.transport.TMemoryInputTransport(getFrameBuffer().
↪array());
        org.apache.thrift.protocol.TProtocol prot = client.
↪getProtocolFactory().getProtocol(memoryTransport);
        return (new Client(prot)).recv_multiply();
      }
    }

  }

  public static class Processor<I extends Iface> extends org.apache.thrift.
↪TBaseProcessor<I> implements org.apache.thrift.TProcessor {
    private static final Logger LOGGER = LoggerFactory.getLogger(Processor.
↪class.getName());
    public Processor(I iface) {
      super(iface, getProcessMap(new HashMap<String, org.apache.thrift.
↪ProcessFunction<I, ? extends org.apache.thrift.TBase>>()));
    }

    protected Processor(I iface, Map<String,  org.apache.thrift.
↪ProcessFunction<I, ? extends  org.apache.thrift.TBase>> processMap) {
```

```
      super(iface, getProcessMap(processMap));
    }

    private static <I extends Iface> Map<String,  org.apache.thrift.
↪ProcessFunction<I, ? extends  org.apache.thrift.TBase>> getProcessMap(Map
↪<String,  org.apache.thrift.ProcessFunction<I, ? extends  org.apache.
↪thrift.TBase>> processMap) {
      processMap.put("multiply", new multiply());
      return processMap;
    }

    public static class multiply<I extends Iface> extends org.apache.thrift.
↪ProcessFunction<I, multiply_args> {
      public multiply() {
        super("multiply");
      }

      public multiply_args getEmptyArgsInstance() {
        return new multiply_args();
      }

      protected boolean isOneway() {
        return false;
      }

      public multiply_result getResult(I iface, multiply_args args) throws␣
↪org.apache.thrift.TException {
        multiply_result result = new multiply_result();
        result.success = iface.multiply(args.n1, args.n2);
        result.setSuccessIsSet(true);
        return result;
      }
    }

  }

  public static class multiply_args implements org.apache.thrift.TBase
↪<multiply_args, multiply_args._Fields>, java.io.Serializable, Cloneable   {
    private static final org.apache.thrift.protocol.TStruct STRUCT_DESC =␣
↪new org.apache.thrift.protocol.TStruct("multiply_args");

    private static final org.apache.thrift.protocol.TField N1_FIELD_DESC =␣
↪new org.apache.thrift.protocol.TField("n1", org.apache.thrift.protocol.
↪TType.I32, (short)1);
    private static final org.apache.thrift.protocol.TField N2_FIELD_DESC =␣
↪new org.apache.thrift.protocol.TField("n2", org.apache.thrift.protocol.
↪TType.I32, (short)2);

    private static final Map<Class<? extends IScheme>, SchemeFactory>␣
↪schemes = new HashMap<Class<? extends IScheme>, SchemeFactory>();
    static {
      schemes.put(StandardScheme.class, new multiply_
↪argsStandardSchemeFactory());
      schemes.put(TupleScheme.class, new multiply_argsTupleSchemeFactory());
    }

    public int n1; // required
    public int n2; // required
```

```
    /** The set of fields this struct contains, along with convenience␣
↪methods for finding and manipulating them. */
    public enum _Fields implements org.apache.thrift.TFieldIdEnum {
      N1((short)1, "n1"),
      N2((short)2, "n2");

      private static final Map<String, _Fields> byName = new HashMap<String,␣
↪_Fields>();

      static {
        for (_Fields field : EnumSet.allOf(_Fields.class)) {
          byName.put(field.getFieldName(), field);
        }
      }

      /**
       * Find the _Fields constant that matches fieldId, or null if its not␣
↪found.
       */
      public static _Fields findByThriftId(int fieldId) {
        switch(fieldId) {
          case 1: // N1
            return N1;
          case 2: // N2
            return N2;
          default:
            return null;
        }
      }

      /**
       * Find the _Fields constant that matches fieldId, throwing an␣
↪exception
       * if it is not found.
       */
      public static _Fields findByThriftIdOrThrow(int fieldId) {
        _Fields fields = findByThriftId(fieldId);
        if (fields == null) throw new IllegalArgumentException("Field " +␣
↪fieldId + " doesn't exist!");
        return fields;
      }

      /**
       * Find the _Fields constant that matches name, or null if its not␣
↪found.
       */
      public static _Fields findByName(String name) {
        return byName.get(name);
      }

      private final short _thriftId;
      private final String _fieldName;

      _Fields(short thriftId, String fieldName) {
        _thriftId = thriftId;
        _fieldName = fieldName;
      }
```

```
      public short getThriftFieldId() {
        return _thriftId;
      }

      public String getFieldName() {
        return _fieldName;
      }
    }

    // isset id assignments
    private static final int __N1_ISSET_ID = 0;
    private static final int __N2_ISSET_ID = 1;
    private byte __isset_bitfield = 0;
    public static final Map<_Fields, org.apache.thrift.meta_data.
↪FieldMetaData> metaDataMap;
    static {
      Map<_Fields, org.apache.thrift.meta_data.FieldMetaData> tmpMap = new␣
↪EnumMap<_Fields, org.apache.thrift.meta_data.FieldMetaData>(_Fields.class);
      tmpMap.put(_Fields.N1, new org.apache.thrift.meta_data.FieldMetaData(
↪"n1", org.apache.thrift.TFieldRequirementType.DEFAULT,
          new org.apache.thrift.meta_data.FieldValueMetaData(org.apache.
↪thrift.protocol.TType.I32          , "int")));
      tmpMap.put(_Fields.N2, new org.apache.thrift.meta_data.FieldMetaData(
↪"n2", org.apache.thrift.TFieldRequirementType.DEFAULT,
          new org.apache.thrift.meta_data.FieldValueMetaData(org.apache.
↪thrift.protocol.TType.I32          , "int")));
      metaDataMap = Collections.unmodifiableMap(tmpMap);
      org.apache.thrift.meta_data.FieldMetaData.
↪addStructMetaDataMap(multiply_args.class, metaDataMap);
    }

    public multiply_args() {
    }

    public multiply_args(
      int n1,
      int n2)
    {
      this();
      this.n1 = n1;
      setN1IsSet(true);
      this.n2 = n2;
      setN2IsSet(true);
    }

    /**
     * Performs a deep copy on <i>other</i>.
     */
    public multiply_args(multiply_args other) {
      __isset_bitfield = other.__isset_bitfield;
      this.n1 = other.n1;
      this.n2 = other.n2;
    }

    public multiply_args deepCopy() {
      return new multiply_args(this);
    }
```

```
    @Override
    public void clear() {
      setN1IsSet(false);
      this.n1 = 0;
      setN2IsSet(false);
      this.n2 = 0;
    }

    public int getN1() {
      return this.n1;
    }

    public multiply_args setN1(int n1) {
      this.n1 = n1;
      setN1IsSet(true);
      return this;
    }

    public void unsetN1() {
      __isset_bitfield = EncodingUtils.clearBit(__isset_bitfield, __N1_ISSET_
↪ID);
    }

    /** Returns true if field n1 is set (has been assigned a value) and␣
↪false otherwise */
    public boolean isSetN1() {
      return EncodingUtils.testBit(__isset_bitfield, __N1_ISSET_ID);
    }

    public void setN1IsSet(boolean value) {
      __isset_bitfield = EncodingUtils.setBit(__isset_bitfield, __N1_ISSET_
↪ID, value);
    }

    public int getN2() {
      return this.n2;
    }

    public multiply_args setN2(int n2) {
      this.n2 = n2;
      setN2IsSet(true);
      return this;
    }

    public void unsetN2() {
      __isset_bitfield = EncodingUtils.clearBit(__isset_bitfield, __N2_ISSET_
↪ID);
    }

    /** Returns true if field n2 is set (has been assigned a value) and␣
↪false otherwise */
    public boolean isSetN2() {
      return EncodingUtils.testBit(__isset_bitfield, __N2_ISSET_ID);
    }

    public void setN2IsSet(boolean value) {
      __isset_bitfield = EncodingUtils.setBit(__isset_bitfield, __N2_ISSET_
↪ID, value);
```

```
    }

    public void setFieldValue(_Fields field, Object value) {
      switch (field) {
      case N1:
        if (value == null) {
          unsetN1();
        } else {
          setN1((Integer)value);
        }
        break;

      case N2:
        if (value == null) {
          unsetN2();
        } else {
          setN2((Integer)value);
        }
        break;

      }
    }

    public Object getFieldValue(_Fields field) {
      switch (field) {
      case N1:
        return Integer.valueOf(getN1());

      case N2:
        return Integer.valueOf(getN2());

      }
      throw new IllegalStateException();
    }

    /** Returns true if field corresponding to fieldID is set (has been
↪assigned a value) and false otherwise */
    public boolean isSet(_Fields field) {
      if (field == null) {
        throw new IllegalArgumentException();
      }

      switch (field) {
      case N1:
        return isSetN1();
      case N2:
        return isSetN2();
      }
      throw new IllegalStateException();
    }

    @Override
    public boolean equals(Object that) {
      if (that == null)
        return false;
      if (that instanceof multiply_args)
        return this.equals((multiply_args)that);
      return false;
```

```
    }

    public boolean equals(multiply_args that) {
      if (that == null)
        return false;

      boolean this_present_n1 = true;
      boolean that_present_n1 = true;
      if (this_present_n1 || that_present_n1) {
        if (!(this_present_n1 && that_present_n1))
          return false;
        if (this.n1 != that.n1)
          return false;
      }

      boolean this_present_n2 = true;
      boolean that_present_n2 = true;
      if (this_present_n2 || that_present_n2) {
        if (!(this_present_n2 && that_present_n2))
          return false;
        if (this.n2 != that.n2)
          return false;
      }

      return true;
    }

    @Override
    public int hashCode() {
      return 0;
    }

    public int compareTo(multiply_args other) {
      if (!getClass().equals(other.getClass())) {
        return getClass().getName().compareTo(other.getClass().getName());
      }

      int lastComparison = 0;
      multiply_args typedOther = (multiply_args)other;

      lastComparison = Boolean.valueOf(isSetN1()).compareTo(typedOther.
↪isSetN1());
      if (lastComparison != 0) {
        return lastComparison;
      }
      if (isSetN1()) {
        lastComparison = org.apache.thrift.TBaseHelper.compareTo(this.n1,␣
↪typedOther.n1);
        if (lastComparison != 0) {
          return lastComparison;
        }
      }
      lastComparison = Boolean.valueOf(isSetN2()).compareTo(typedOther.
↪isSetN2());
      if (lastComparison != 0) {
        return lastComparison;
      }
      if (isSetN2()) {
```

```
        lastComparison = org.apache.thrift.TBaseHelper.compareTo(this.n2,␣
↪typedOther.n2);
        if (lastComparison != 0) {
          return lastComparison;
        }
      }
      return 0;
    }

    public _Fields fieldForId(int fieldId) {
      return _Fields.findByThriftId(fieldId);
    }

    public void read(org.apache.thrift.protocol.TProtocol iprot) throws org.
↪apache.thrift.TException {
      schemes.get(iprot.getScheme()).getScheme().read(iprot, this);
    }

    public void write(org.apache.thrift.protocol.TProtocol oprot) throws org.
↪apache.thrift.TException {
      schemes.get(oprot.getScheme()).getScheme().write(oprot, this);
    }

    @Override
    public String toString() {
      StringBuilder sb = new StringBuilder("multiply_args(");
      boolean first = true;

      sb.append("n1:");
      sb.append(this.n1);
      first = false;
      if (!first) sb.append(", ");
      sb.append("n2:");
      sb.append(this.n2);
      first = false;
      sb.append(")");
      return sb.toString();
    }

    public void validate() throws org.apache.thrift.TException {
      // check for required fields
      // check for sub-struct validity
    }

    private void writeObject(java.io.ObjectOutputStream out) throws java.io.
↪IOException {
      try {
        write(new org.apache.thrift.protocol.TCompactProtocol(new org.apache.
↪thrift.transport.TIOStreamTransport(out)));
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
      }
    }

    private void readObject(java.io.ObjectInputStream in) throws java.io.
↪IOException, ClassNotFoundException {
      try {
        // it doesn't seem like you should have to do this, but java␣
↪serialization is wacky, and doesn't call the default constructor.
```

```
        __isset_bitfield = 0;
        read(new org.apache.thrift.protocol.TCompactProtocol(new org.apache.
↪thrift.transport.TIOStreamTransport(in)));
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
      }
    }

    private static class multiply_argsStandardSchemeFactory implements␣
↪SchemeFactory {
      public multiply_argsStandardScheme getScheme() {
        return new multiply_argsStandardScheme();
      }
    }

    private static class multiply_argsStandardScheme extends StandardScheme
↪<multiply_args> {

      public void read(org.apache.thrift.protocol.TProtocol iprot, multiply_
↪args struct) throws org.apache.thrift.TException {
        org.apache.thrift.protocol.TField schemeField;
        iprot.readStructBegin();
        while (true)
        {
          schemeField = iprot.readFieldBegin();
          if (schemeField.type == org.apache.thrift.protocol.TType.STOP) {
            break;
          }
          switch (schemeField.id) {
            case 1: // N1
              if (schemeField.type == org.apache.thrift.protocol.TType.I32) {
                struct.n1 = iprot.readI32();
                struct.setN1IsSet(true);
              } else {
                org.apache.thrift.protocol.TProtocolUtil.skip(iprot,␣
↪schemeField.type);
              }
              break;
            case 2: // N2
              if (schemeField.type == org.apache.thrift.protocol.TType.I32) {
                struct.n2 = iprot.readI32();
                struct.setN2IsSet(true);
              } else {
                org.apache.thrift.protocol.TProtocolUtil.skip(iprot,␣
↪schemeField.type);
              }
              break;
            default:
              org.apache.thrift.protocol.TProtocolUtil.skip(iprot,␣
↪schemeField.type);
          }
          iprot.readFieldEnd();
        }
        iprot.readStructEnd();

        // check for required fields of primitive type, which can't be␣
↪checked in the validate method
        struct.validate();
```

```
      }

      public void write(org.apache.thrift.protocol.TProtocol oprot, multiply_
↪args struct) throws org.apache.thrift.TException {
        struct.validate();

        oprot.writeStructBegin(STRUCT_DESC);
        oprot.writeFieldBegin(N1_FIELD_DESC);
        oprot.writeI32(struct.n1);
        oprot.writeFieldEnd();
        oprot.writeFieldBegin(N2_FIELD_DESC);
        oprot.writeI32(struct.n2);
        oprot.writeFieldEnd();
        oprot.writeFieldStop();
        oprot.writeStructEnd();
      }

    }

    private static class multiply_argsTupleSchemeFactory implements␣
↪SchemeFactory {
      public multiply_argsTupleScheme getScheme() {
        return new multiply_argsTupleScheme();
      }
    }

    private static class multiply_argsTupleScheme extends TupleScheme
↪<multiply_args> {

      @Override
      public void write(org.apache.thrift.protocol.TProtocol prot, multiply_
↪args struct) throws org.apache.thrift.TException {
        TTupleProtocol oprot = (TTupleProtocol) prot;
        BitSet optionals = new BitSet();
        if (struct.isSetN1()) {
          optionals.set(0);
        }
        if (struct.isSetN2()) {
          optionals.set(1);
        }
        oprot.writeBitSet(optionals, 2);
        if (struct.isSetN1()) {
          oprot.writeI32(struct.n1);
        }
        if (struct.isSetN2()) {
          oprot.writeI32(struct.n2);
        }
      }

      @Override
      public void read(org.apache.thrift.protocol.TProtocol prot, multiply_
↪args struct) throws org.apache.thrift.TException {
        TTupleProtocol iprot = (TTupleProtocol) prot;
        BitSet incoming = iprot.readBitSet(2);
        if (incoming.get(0)) {
          struct.n1 = iprot.readI32();
          struct.setN1IsSet(true);
        }
```

```
        if (incoming.get(1)) {
          struct.n2 = iprot.readI32();
          struct.setN2IsSet(true);
        }
      }
    }

  }

  public static class multiply_result implements org.apache.thrift.TBase
↪<multiply_result, multiply_result._Fields>, java.io.Serializable,␣
↪Cloneable   {
    private static final org.apache.thrift.protocol.TStruct STRUCT_DESC =␣
↪new org.apache.thrift.protocol.TStruct("multiply_result");

    private static final org.apache.thrift.protocol.TField SUCCESS_FIELD_
↪DESC = new org.apache.thrift.protocol.TField("success", org.apache.thrift.
↪protocol.TType.I32, (short)0);

    private static final Map<Class<? extends IScheme>, SchemeFactory>␣
↪schemes = new HashMap<Class<? extends IScheme>, SchemeFactory>();
    static {
      schemes.put(StandardScheme.class, new multiply_
↪resultStandardSchemeFactory());
      schemes.put(TupleScheme.class, new multiply_
↪resultTupleSchemeFactory());
    }

    public int success; // required

    /** The set of fields this struct contains, along with convenience␣
↪methods for finding and manipulating them. */
    public enum _Fields implements org.apache.thrift.TFieldIdEnum {
      SUCCESS((short)0, "success");

      private static final Map<String, _Fields> byName = new HashMap<String,␣
↪_Fields>();

      static {
        for (_Fields field : EnumSet.allOf(_Fields.class)) {
          byName.put(field.getFieldName(), field);
        }
      }

      /**
       * Find the _Fields constant that matches fieldId, or null if its not␣
↪found.
       */
      public static _Fields findByThriftId(int fieldId) {
        switch(fieldId) {
          case 0: // SUCCESS
            return SUCCESS;
          default:
            return null;
        }
      }

      /**
```

```
       * Find the _Fields constant that matches fieldId, throwing an␣
↪exception
       * if it is not found.
       */
      public static _Fields findByThriftIdOrThrow(int fieldId) {
        _Fields fields = findByThriftId(fieldId);
        if (fields == null) throw new IllegalArgumentException("Field " +␣
↪fieldId + " doesn't exist!");
        return fields;
      }

      /**
       * Find the _Fields constant that matches name, or null if its not␣
↪found.
       */
      public static _Fields findByName(String name) {
        return byName.get(name);
      }

      private final short _thriftId;
      private final String _fieldName;

      _Fields(short thriftId, String fieldName) {
        _thriftId = thriftId;
        _fieldName = fieldName;
      }

      public short getThriftFieldId() {
        return _thriftId;
      }

      public String getFieldName() {
        return _fieldName;
      }
    }

    // isset id assignments
    private static final int __SUCCESS_ISSET_ID = 0;
    private byte __isset_bitfield = 0;
    public static final Map<_Fields, org.apache.thrift.meta_data.
↪FieldMetaData> metaDataMap;
    static {
      Map<_Fields, org.apache.thrift.meta_data.FieldMetaData> tmpMap = new␣
↪EnumMap<_Fields, org.apache.thrift.meta_data.FieldMetaData>(_Fields.class);
      tmpMap.put(_Fields.SUCCESS, new org.apache.thrift.meta_data.
↪FieldMetaData("success", org.apache.thrift.TFieldRequirementType.DEFAULT,
          new org.apache.thrift.meta_data.FieldValueMetaData(org.apache.
↪thrift.protocol.TType.I32          , "int")));
      metaDataMap = Collections.unmodifiableMap(tmpMap);
      org.apache.thrift.meta_data.FieldMetaData.
↪addStructMetaDataMap(multiply_result.class, metaDataMap);
    }

    public multiply_result() {
    }

    public multiply_result(
      int success)
```

```
    {
      this();
      this.success = success;
      setSuccessIsSet(true);
    }

    /**
     * Performs a deep copy on <i>other</i>.
     */
    public multiply_result(multiply_result other) {
      __isset_bitfield = other.__isset_bitfield;
      this.success = other.success;
    }

    public multiply_result deepCopy() {
      return new multiply_result(this);
    }

    @Override
    public void clear() {
      setSuccessIsSet(false);
      this.success = 0;
    }

    public int getSuccess() {
      return this.success;
    }

    public multiply_result setSuccess(int success) {
      this.success = success;
      setSuccessIsSet(true);
      return this;
    }

    public void unsetSuccess() {
      __isset_bitfield = EncodingUtils.clearBit(__isset_bitfield, __SUCCESS_
↪ISSET_ID);
    }

    /** Returns true if field success is set (has been assigned a value) and␣
↪false otherwise */
    public boolean isSetSuccess() {
      return EncodingUtils.testBit(__isset_bitfield, __SUCCESS_ISSET_ID);
    }

    public void setSuccessIsSet(boolean value) {
      __isset_bitfield = EncodingUtils.setBit(__isset_bitfield, __SUCCESS_
↪ISSET_ID, value);
    }

    public void setFieldValue(_Fields field, Object value) {
      switch (field) {
      case SUCCESS:
        if (value == null) {
          unsetSuccess();
        } else {
          setSuccess((Integer)value);
        }
```

```
        break;

      }
    }

    public Object getFieldValue(_Fields field) {
      switch (field) {
      case SUCCESS:
        return Integer.valueOf(getSuccess());

      }
      throw new IllegalStateException();
    }

    /** Returns true if field corresponding to fieldID is set (has been
↪assigned a value) and false otherwise */
    public boolean isSet(_Fields field) {
      if (field == null) {
        throw new IllegalArgumentException();
      }

      switch (field) {
      case SUCCESS:
        return isSetSuccess();
      }
      throw new IllegalStateException();
    }

    @Override
    public boolean equals(Object that) {
      if (that == null)
        return false;
      if (that instanceof multiply_result)
        return this.equals((multiply_result)that);
      return false;
    }

    public boolean equals(multiply_result that) {
      if (that == null)
        return false;

      boolean this_present_success = true;
      boolean that_present_success = true;
      if (this_present_success || that_present_success) {
        if (!(this_present_success && that_present_success))
          return false;
        if (this.success != that.success)
          return false;
      }

      return true;
    }

    @Override
    public int hashCode() {
      return 0;
    }
```

```
    public int compareTo(multiply_result other) {
      if (!getClass().equals(other.getClass())) {
        return getClass().getName().compareTo(other.getClass().getName());
      }

      int lastComparison = 0;
      multiply_result typedOther = (multiply_result)other;

      lastComparison = Boolean.valueOf(isSetSuccess()).compareTo(typedOther.
↪isSetSuccess());
      if (lastComparison != 0) {
        return lastComparison;
      }
      if (isSetSuccess()) {
        lastComparison = org.apache.thrift.TBaseHelper.compareTo(this.
↪success, typedOther.success);
        if (lastComparison != 0) {
          return lastComparison;
        }
      }
      return 0;
    }

    public _Fields fieldForId(int fieldId) {
      return _Fields.findByThriftId(fieldId);
    }

    public void read(org.apache.thrift.protocol.TProtocol iprot) throws org.
↪apache.thrift.TException {
      schemes.get(iprot.getScheme()).getScheme().read(iprot, this);
    }

    public void write(org.apache.thrift.protocol.TProtocol oprot) throws org.
↪apache.thrift.TException {
      schemes.get(oprot.getScheme()).getScheme().write(oprot, this);
      }

    @Override
    public String toString() {
      StringBuilder sb = new StringBuilder("multiply_result(");
      boolean first = true;

      sb.append("success:");
      sb.append(this.success);
      first = false;
      sb.append(")");
      return sb.toString();
    }

    public void validate() throws org.apache.thrift.TException {
      // check for required fields
      // check for sub-struct validity
    }

    private void writeObject(java.io.ObjectOutputStream out) throws java.io.
↪IOException {
      try {
        write(new org.apache.thrift.protocol.TCompactProtocol(new org.apache.
↪thrift.transport.TIOStreamTransport(out)));
```

```
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
      }
    }

    private void readObject(java.io.ObjectInputStream in) throws java.io.
↪IOException, ClassNotFoundException {
      try {
        // it doesn't seem like you should have to do this, but java␣
↪serialization is wacky, and doesn't call the default constructor.
        __isset_bitfield = 0;
        read(new org.apache.thrift.protocol.TCompactProtocol(new org.apache.
↪thrift.transport.TIOStreamTransport(in)));
      } catch (org.apache.thrift.TException te) {
        throw new java.io.IOException(te);
      }
    }

    private static class multiply_resultStandardSchemeFactory implements␣
↪SchemeFactory {
      public multiply_resultStandardScheme getScheme() {
        return new multiply_resultStandardScheme();
      }
    }

    private static class multiply_resultStandardScheme extends StandardScheme
↪<multiply_result> {

      public void read(org.apache.thrift.protocol.TProtocol iprot, multiply_
↪result struct) throws org.apache.thrift.TException {
        org.apache.thrift.protocol.TField schemeField;
        iprot.readStructBegin();
        while (true)
        {
          schemeField = iprot.readFieldBegin();
          if (schemeField.type == org.apache.thrift.protocol.TType.STOP) {
            break;
          }
          switch (schemeField.id) {
            case 0: // SUCCESS
              if (schemeField.type == org.apache.thrift.protocol.TType.I32) {
                struct.success = iprot.readI32();
                struct.setSuccessIsSet(true);
              } else {
                org.apache.thrift.protocol.TProtocolUtil.skip(iprot,␣
↪schemeField.type);
              }
              break;
            default:
              org.apache.thrift.protocol.TProtocolUtil.skip(iprot,␣
↪schemeField.type);
          }
          iprot.readFieldEnd();
        }
        iprot.readStructEnd();

        // check for required fields of primitive type, which can't be␣
↪checked in the validate method
```

```
        struct.validate();
      }

      public void write(org.apache.thrift.protocol.TProtocol oprot, multiply_
↪result struct) throws org.apache.thrift.TException {
        struct.validate();

        oprot.writeStructBegin(STRUCT_DESC);
        if (struct.isSetSuccess()) {
          oprot.writeFieldBegin(SUCCESS_FIELD_DESC);
          oprot.writeI32(struct.success);
          oprot.writeFieldEnd();
        }
        oprot.writeFieldStop();
        oprot.writeStructEnd();
      }

    }

    private static class multiply_resultTupleSchemeFactory implements␣
↪SchemeFactory {
      public multiply_resultTupleScheme getScheme() {
        return new multiply_resultTupleScheme();
      }
    }

    private static class multiply_resultTupleScheme extends TupleScheme
↪<multiply_result> {

      @Override
      public void write(org.apache.thrift.protocol.TProtocol prot, multiply_
↪result struct) throws org.apache.thrift.TException {
        TTupleProtocol oprot = (TTupleProtocol) prot;
        BitSet optionals = new BitSet();
        if (struct.isSetSuccess()) {
          optionals.set(0);
        }
        oprot.writeBitSet(optionals, 1);
        if (struct.isSetSuccess()) {
          oprot.writeI32(struct.success);
        }
      }

      @Override
      public void read(org.apache.thrift.protocol.TProtocol prot, multiply_
↪result struct) throws org.apache.thrift.TException {
        TTupleProtocol iprot = (TTupleProtocol) prot;
        BitSet incoming = iprot.readBitSet(1);
        if (incoming.get(0)) {
          struct.success = iprot.readI32();
          struct.setSuccessIsSet(true);
        }
      }
    }

  }

}
```

CHAPTER 7

Indices and tables

- genindex
- search