Apache Thrift™

The **Apache Thrift** software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages.

Example

Apache Thrift allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages. Instead of writing a load of boilerplate code to serialize and transport your objects and invoke remote methods, you can get right down to business.

The following example is a simple service to store user objects for a web front end.

Thrift Definition File

```
/**
 * 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()

}
```

Python Client

```
def main():
    # 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 = Calculator.Client(protocol)

    # Connect!
    transport.open()

    client.ping()
    print('ping()')

    sum_ = client.add(1, 1)
```

Java Server

Initialize the Server:

```
    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();
    }
```

This snippet was generated by Apache Thrift's **source tree docs**:
[tutorial/java/src/JavaServer.java]

The CalculatorHandler:

```
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.whatOp = work.op.getValue();
          io.why = "Cannot divide by 0";
          throw io;
        }
        val = work.num1 / work.num2;
        break;
      default:
        InvalidOperation io = new InvalidOperation();
        io.whatOp = 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()");
    }

}
```