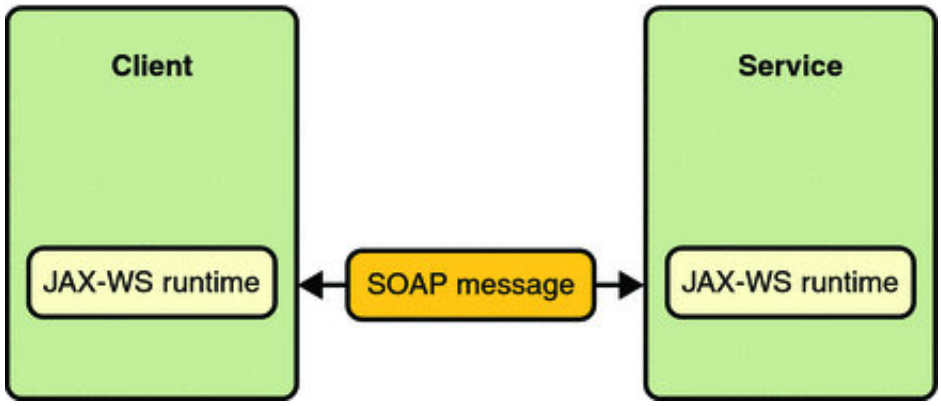
Introduction

SOAP stands for Simple Object Access Protocol. It is a XML based protocol for accessing web service. By using SOAP, we can interact with other programming language applications. In another word, SOAP web services can be written in any programming language and executed in any platform.

Structure

You can find how SOAP works in Java in the Oracle Docs. This project is implemented both server and client sides using SOAP. Our server and client communicates like the following image.



This documentation will introduce how server and client programs work, how we designed with code and detailed explaination. Lastly, client changes its system time by adding half of *RTT* (Round-trip time) and server's system time. Screenshots are attached as a proof of success.

Software, libraries and IDEs

- Eclipse Java EE IDE for Web Developers version: Oxygen.2 Release (4.7.2) as IDE
- Apache Tomcat v8.0 as the target runtime with default configuration
- Apache Axis as the web service runtime
- Java 8
- Maven to organize libraries
- `org.json` used to serialize and deserialize the mapping between JSON string and object

Server

Server is created as a dynamic web project so that eclipse can do some automatic configurations to save life. Server runs on Apache 8. Server's responsiblity is pretty simple. It generates server's system time and returns the result as a JSON string.

```
public String getServerTime() {
    JSONObject jsonObject = new JSONObject();
    Calendar cal = Calendar.getInstance();
    Date date = cal.getTime();

    long timeInMillis = date.getTime();
    jsonObject.put("timeInMillis", String.valueOf(timeInMillis));
    return jsonObject.toString();
}
```

Server converts date to milliseconds in a format of `long`. It is convenient for client to sum up its RTT. Then, the result will be put into a `JSONObject` as key `"timeInMillis"`.

Server uses the Web Service Definition Language WSDL (created by Eclipse) to describe the functionality offered by itself. It contains the information that how the service can be called, and what parameters, data structures it expectes and returns.

Let's take a look at one important section in `wsdl` file.

```
<wsdl:service name="SOAPClassService">

    <wsdl:port binding="impl:SOAPClassSoapBinding" name="SOAPClass">

        <wsdlsoap:address
location="http://192.168.0.13:8080/SOAPExample/services/SoapClass"/>

    </wsdl:port>

</wsdl:service>
```

It defines the wsdl soap address, port number, server IP and where the service holds. These information will be used by client side later.

Accessing to an specific URL in the browser to test and see if server works well.

```
http://localhost:8080/SOAPExample/sampleSoapClassProxy/TestClient.jsp?
endpoint=http://localhost:9163/SOAPExample/services/SoapClass
```

Obviously, we noticed in this URL, service will be reached in the end point of "http://localhost:9163/SOAPExample/services/SoapClass". In another word, this is the **end point** that client will call.

Client

Client side runs on Apache 8 and Apache Axis.

In order to show the robustness and readability of the code, some classes and interface are used.

Since server uses WSDL to contain its information, then some parts of client side can be constructed from server's WSDL file.

Find server

As mentioned before, `http://localhost:9163/SOAPExample/services/SoapClass` is the end point (a default URL in server testing environment) that client will use. Therefore, we need to replace `localhost:9163` to server's actual `IP address` and `port number`. For example, `192.168.0.13:8080` is the one that server uses in my machine. Therefore, client will connect to server using end point `http://192.168.0.13:8080/SOAPExample/services/SoapClass`. Then in client, the Java class will be setted by the following code.

```
_endpoint = "http://192.168.0.13:8080/SOAPExample/services/SoapClass";
((javax.xml.rpc.Stub)soapClass)._setProperty("javax.xml.rpc.service.endpoint.ad
dress", _endpoint);
```

Calculate RTT

Because we need to calculate the RTT, client will calculate how long the request takes from sending to receiving a response. It will capture current client's system time. Milliseconds is used for calculating convenience.

```
long startTimeInMillis = System.currentTimeMillis();
```

In the next step, client will invoke the service that server provided `getServerTime()`. Client expects to get the response in terms of `String`.

```
String response = dcp.getServerTime();
```

Current client's system time is captured again to calculate its RTT.

```
long endTimeInMillis = System.currentTimeMillis();
```

By printing the response, we can clearly see the data is passed by JSON.

```
System.out.println(response); // {"timeInMillis": "1549515387238"}
```

After calculated, client's system should be changed by using the following formular.

```
long calculatedClientTime = (endTimeInMillis - startTimeInMillis) / 2 +
serverTimeInMillis;
```

`1549262779224` is the result that added half of RTT and server's system time in the form of millseconds. Client will parse it and print it to human-readable format `Wed Feb 06 20:56:27 PST 2019`. Screenshot will attached below.

Set client's system time

Lastly, client system time will be changed.

```
Calendar newClientTime = Calendar.getInstance();
newClientTime.setTimeInMillis(calculatedClientTime); // newClientTime now is
setted to calculated time
String strDateToSet = dateFormatter.format(newClientTime.getTime()); // convert
to a specific format
String command = "date " + strDateToSet; // Mac's command to change time
Process proc = Runtime.getRuntime().exec(command); // a process will be
launched to change system time in a command of `bash/shell`
```

Screenshot

Screenshots of server and client will be introduced.

Sever

When server is invoked, Eclipse's **TCP/IP Monitor** is able to show the SOAP of request and response after invoking `getServerTime()`. Obviously, they are in the format of XML because SOAP is XML-based.

Request

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body>
<getServerTime xmlns="http://com"/></soapenv:Body></soapenv:Envelope>
```

Response

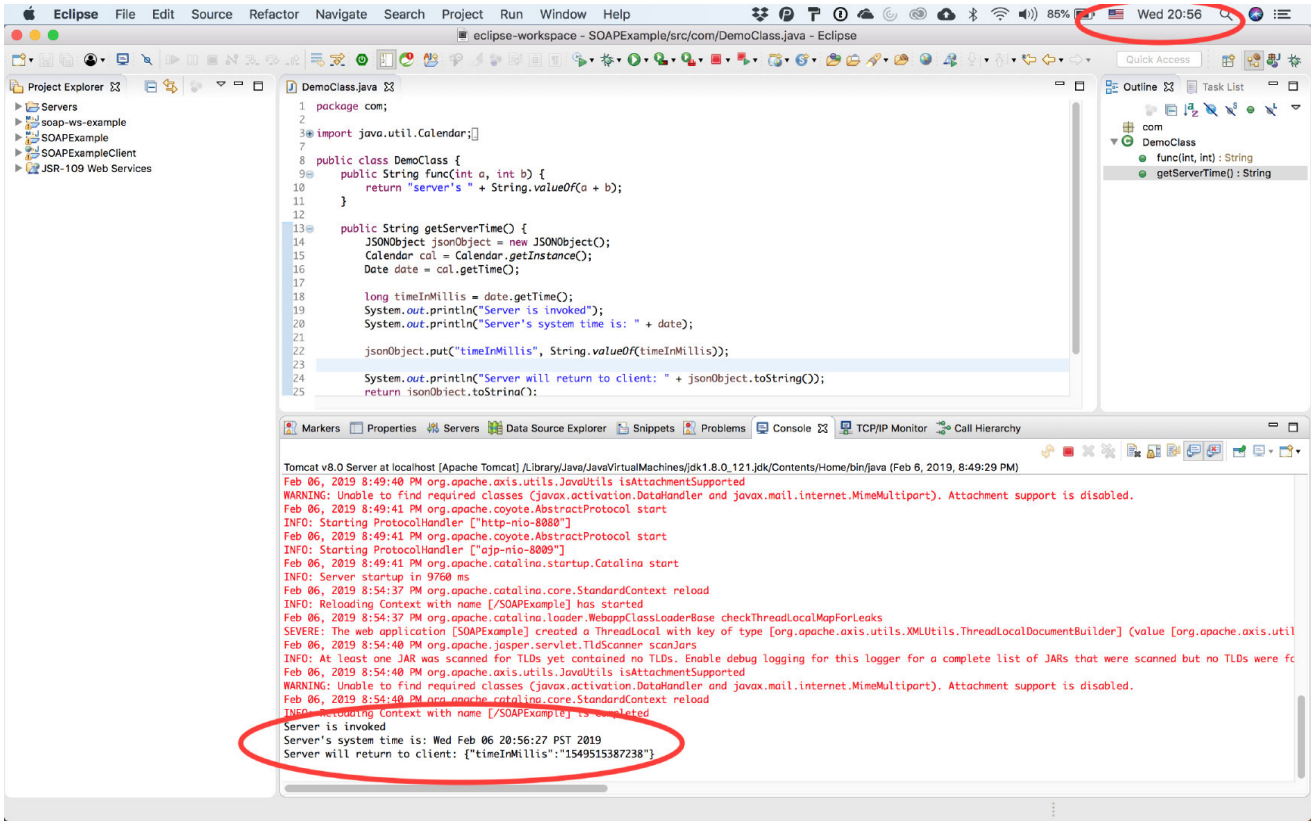
```
<?xml version="1.0" encoding="utf-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body>
<getServerTimeResponse xmlns="http://com"><getServerTimeReturn>
{&amp;quot;timeInMillis&amp;quot;:&amp;quot;1549515387238&amp;quot;}</getServerTimeReturn>
</getServerTimeResponse></soapenv:Body></soapenv:Envelope>
```

Terminal output

This is the output in the terminal of server.

```
INFO: Reloading Context with name [/SOAPExample] is completed
Server is invoked
Server's system time is: Wed Feb 06 20:56:27 PST 2019
Server will return to client: {"timeInMillis":"1549515387238"}
```

Server's big picture



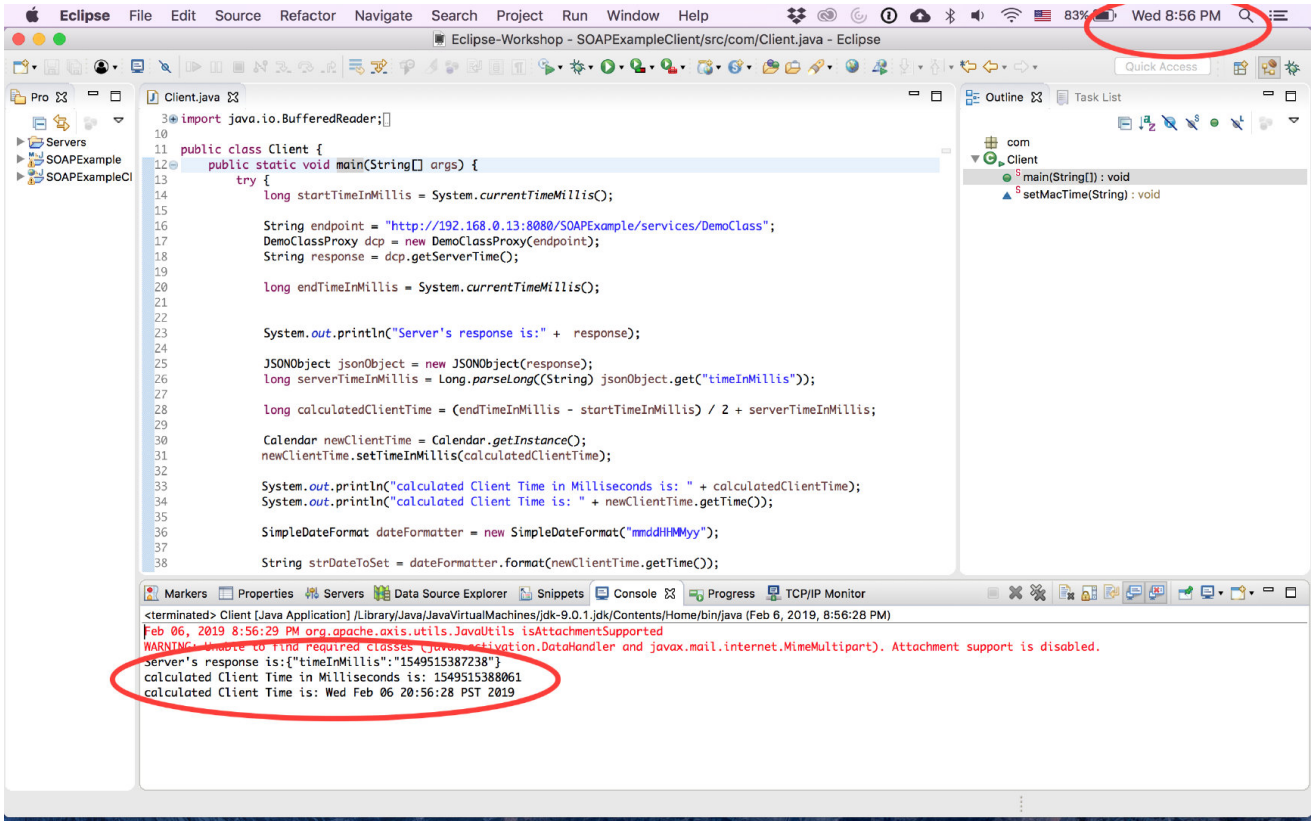
Client

Client sends request to server firstly, followed by the calculated client's system time.

Terminal output

```
Server's response is:{"timeInMillis":"1549515387238"}
calculated Client Time in Milliseconds is: 1549515388061
calculated Client Time is: Wed Feb 06 20:56:28 PST 2019
```

Client's big picture


Eclipse File Edit Source Refactor Navigate Search Project Run Window Help
Wed 8:56 PM
Eclipse-Workshop - SOAPExampleClient/src/com/Client.java - Eclipse
Quick Access
Client.java
Outline
Task List
Servers
SOAPExample
com
Client
main(String[]) : void
setMacTime(String) : void
Markers Properties Servers Data Source Explorer Snippets Console Progress TCP/IP Monitor
<terminated> Client [Java Application] /Library/Java/JavaVirtualMachines/jdk-9.0.1.jdk/Contents/Home/bin/java (Feb 6, 2019, 8:56:28 PM)
Feb 06, 2019 8:56:29 PM org.apache.axis.utils.JavaUtils isAttachmentSupported
Server's response is:{"timeInMillis":"1549515387238"}
calculated Client Time in Milliseconds is: 15495153880061
calculated Client Time is: Wed Feb 06 20:56:28 PST 2019