Distributed Computing

Lesson 20: JSON RPC

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

1. Alternatives to Web Services
2. Examples for JSON

website

Overview

- What alternatives to web services exist?
- Learn more about JSON RPC

Alternatives to Web Services

- So far you have seen

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)
- This is nothing that you can simply set up and manage in a few days

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)
- This is nothing that you can simply set up and manage in a few days
- It makes sense for a big enterprise

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)
- This is nothing that you can simply set up and manage in a few days
- It makes sense for a big enterprise
- So are there useful alternatives

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)
- This is nothing that you can simply set up and manage in a few days
- It makes sense for a big enterprise
- So are there useful alternatives when we are *not* in a huge enterprise

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)
- This is nothing that you can simply set up and manage in a few days
- It makes sense for a big enterprise
- So are there useful alternatives when we are *not* in a huge enterprise, but maybe in a smaller or growing company or when we do not have much time for setup and management?

Alternatives to Web Services

- So far you have seen: A fully-fledged SOA is powerful, but heavy-weight and slow (compared to simple RPC methods)
- This is nothing that you can simply set up and manage in a few days
- It makes sense for a big enterprise
- So are there useful alternatives when we are *not* in a huge enterprise, but maybe in a smaller or growing company or when we do not have much time for setup and management?
- There are alternatives . . . but they lack the enterprisy features and support. . .

...Stuff You Already Know

- The "older generation" of RPC/RMI frameworks

. . . Stuff You Already Know

- The "older generation" of RPC/RMI frameworks:
  - Common Object Request Broker Architecture (CORBA) [1–6]

...Stuff You Already Know

- The "older generation" of RPC/RMI frameworks:
  - Common Object Request Broker Architecture (CORBA) [1–6]
  - Distributed Component Object Model (DCOM) [7]

...Stuff You Already Know

- The "older generation" of RPC/RMI frameworks:
  - Common Object Request Broker Architecture (CORBA) [1–6]
  - Distributed Component Object Model (DCOM) [7]
  - .NET Remoting [8]

. . . Stuff You Already Know

- The "older generation" of RPC/RMI frameworks:
  - Common Object Request Broker Architecture (CORBA) [1–6]
  - Distributed Component Object Model (DCOM) [7]
  - .NET Remoting [8]
  - Java Remote Method Invocation (RMI) [6, 9, 10]

...Stuff You Already Know

- The "older generation" of RPC/RMI frameworks:
  - Common Object Request Broker Architecture (CORBA) [1–6]
  - Distributed Component Object Model (DCOM) [7]
  - .NET Remoting [8]
  - Java Remote Method Invocation (RMI) [6, 9, 10]
  - XPCOM (Cross Platform Component Object Model) [11]

XML-RPC

- XML-RPC [12]: RPC protocol using XML to encode its calls and HTTP for transport

[1] graphics from http://de.wikipedia.org/wiki/XML-RPC

XML-RPC

- XML-RPC [12]: RPC protocol using XML to encode its calls and HTTP for transport
- Simple request-response scheme[1]

[1] graphics from http://de.wikipedia.org/wiki/XML-RPC

XML-RPC

```
<?xml version="1.0"?>
 <methodCall>
   <methodName>basket.add</methodName>
   <params>
     <param>
       <value><string>iPad</string></value>
     </param>
     <param>
       <value><int>10</int></value>
     </param>
   </params>
 </methodCall>
```

Request

[1] graphics from http://de.wikipedia.org/wiki/XML-RPC

XML-RPC

```
<?xml version="1.0"?>
 <methodCall>
   <methodName>basket.add</methodName>
   <params>
     <param>
       <value><string>iPad</string></value>
     </param>
     <param>
       <value><int>10</int></value>
     </param>
   </params>
 </methodCall>
```

Request

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>OK</string></value>
    </param>
  </params>
</methodResponse>
```

Response

[1] graphics from http://de.wikipedia.org/wiki/XML-RPC

XML-RPC

- XML-RPC [12]: RPC protocol using XML to encode its calls and HTTP for transport
- Simple request-response scheme[1]
- Predecessor of SOAP and simpler since

[1] graphics from `http://de.wikipedia.org/wiki/XML-RPC`

XML-RPC

- XML-RPC [12]: RPC protocol using XML to encode its calls and HTTP for transport
- Simple request-response scheme[1]
- Predecessor of SOAP and simpler since
  - data is encoded only in one way (SOAP has multiple choices)

[1] graphics from `http://de.wikipedia.org/wiki/XML-RPC`

XML-RPC

- XML-RPC [12]: RPC protocol using XML to encode its calls and HTTP for transport
- Simple request-response scheme[1]
- Predecessor of SOAP and simpler since
  - data is encoded only in one way (SOAP has multiple choices)
  - does not require WSDL service descriptions

[1] graphics from `http://de.wikipedia.org/wiki/XML-RPC`

XML-RPC

- XML-RPC [12]: RPC protocol using XML to encode its calls and HTTP for transport
- Simple request-response scheme[1]
- Predecessor of SOAP and simpler since
  - data is encoded only in one way (SOAP has multiple choices)
  - does not require WSDL service descriptions
- Less heavy-weight than SOAP, many implementations

[1] graphics from `http://de.wikipedia.org/wiki/XML-RPC`

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing structured data

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ... like XML

[2] that's a terrible security risk, though

```
<CreditCard
  Provider="Visa"
  Number="1234-5678-9012-3456"
  Coverage="2e+6"
  Currency="EURO">
  <Owner
    Name="Doe"
    FirstName="John"
    Male="true"
    Age="42">
    <Partner/>
    <Hobbies>
      <Hobby>Riding</Hobby>
      <Hobby>Golf</Hobby>
      <Hobby>Reading</Hobby>
    </Hobbys>
    <Children />
  </Owner>
</CreditCard>
```

XML

image source: http://de.wikipedia.org/wiki/JavaScript_Object_Notation

[2] that's a terrible security risk, though

JSON

```
<CreditCard
  Provider="Visa"
  Number="1234-5678-9012-3456"
  Coverage="2e+6"
  Currency="EURO">
  <Owner
    Name="Doe"
    FirstName="John"
    Male="true"
    Age="42">
    <Partner/>
    <Hobbies>
      <Hobby>Riding</Hobby>
      <Hobby>Golf</Hobby>
      <Hobby>Reading</Hobby>
    </Hobbys>
    <Children />
  </Owner>
</CreditCard>
```

XML

```
{
  "Provider": "Visa",
  "Number": "1234-5678-9012-3456",
  "Coverage": 2e+6,
  "Currency”: "EURO",
  "Owner": {
    "FamilyName": "Doe",
    "FirstName": "John",
    "Male": true,
    "Hobbies": [ "Riding", "Golf", "Reading" ],
    "Age": 42,
    "Children”: [],
    "Partner": null
  }
}
```

JSON

image source: http://de.wikipedia.org/wiki/JavaScript_Object_Notation

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ... like XML

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15]

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ... like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML
- JSON-RPC [16]

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML
- JSON-RPC [16]:
  - similar to XML-RPC

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML
- JSON-RPC [16]:
  - similar to XML-RPC
  - JSON data sent via HTTP or directly TCP/IP to server

[2] that's a terrible security risk, though

```
{ "jsonrpc": "2.0", "method": "checkThisCourse", "params": ["Cs05169a"], "id": 1}
```

Request

image source: http://de.wikipedia.org/wiki/JavaScript_Object_Notation

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML
- JSON-RPC [16]:
  - similar to XML-RPC
  - JSON data sent via HTTP or directly TCP/IP to server
  - Server sends answer as JSON data

[2] that's a terrible security risk, though

JSON

```
{ "jsonrpc": "2.0", "method": "checkThisCourse", "params": ["Cs05169a"], "id": 1}
```

Request

```
{ "jsonrpc": "2.0", "result": "not bad", "error": null, "id": 1}
```

Respose

image source: http://de.wikipedia.org/wiki/JavaScript_Object_Notation

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML
- JSON-RPC [16]:
  - similar to XML-RPC
  - JSON data sent via HTTP or directly TCP/IP to server
  - Server sends answer as JSON data
  - light-weight, many, many implementations and wide support

[2] that's a terrible security risk, though

JSON

- JavaScript Object Notation (JSON) [13, 14] is a data format for serializing and transmitting structured data over a network connection ...like XML
- Basically is JavaScript [15] $\Longrightarrow$ supported by all browsers by default $\Longrightarrow$ if service provides result in JSON, it can *directly* be used at the client side of a HTML-based web application in a browser as it can be parsed with the `eval()` function[2]
- Usually smaller than XML
- JSON-RPC [16]:
  - similar to XML-RPC
  - JSON data sent via HTTP or directly TCP/IP to server
  - Server sends answer as JSON data
  - light-weight, many, many implementations and wide support
  - by the way: GitHub API also uses JSON (though not JSON RPC)

[2] that's a terrible security risk, though

Examples

- There are different implementations for JSON RPC in many different programming languages

Examples

- There are different implementations for JSON RPC in many different programming languages
- We use JSON RPC for Java from `https://github.com/briandilley/jsonrpc4j`

Examples

- There are different implementations for JSON RPC in many different programming languages
- We use JSON RPC for Java from `https://github.com/briandilley/jsonrpc4j`
- JSON RPC server can be implemented as Java Servlet

Examples

- There are different implementations for JSON RPC in many different programming languages
- We use JSON RPC for Java from `https://github.com/briandilley/jsonrpc4j`
- JSON RPC server can be implemented as Java Servlet, which means:
  - deploy it to a servlet container

Examples

- There are different implementations for JSON RPC in many different programming languages
- We use JSON RPC for Java from `https://github.com/briandilley/jsonrpc4j`
- JSON RPC server can be implemented as Java Servlet, which means:
  - deploy it to a servlet container
  - create stand-alone JSON RPC servers (using the same procedure as in the Java Servlets and JSPs lessons)

Examples

- There are different implementations for JSON RPC in many different programming languages
- We use JSON RPC for Java from `https://github.com/briandilley/jsonrpc4j`
- JSON RPC server can be implemented as Java Servlet, which means:
  - deploy it to a servlet container
  - create stand-alone JSON RPC servers (using the same procedure as in the Java Servlets and JSPs lessons)
- JSON RPC clients can be easily constructed, too

Examples

- There are different implementations for JSON RPC in many different programming languages
- We use JSON RPC for Java from `https://github.com/briandilley/jsonrpc4j`
- JSON RPC server can be implemented as Java Servlet, which means:
  - deploy it to a servlet container
  - create stand-alone JSON RPC servers (using the same procedure as in the Java Servlets and JSPs lessons)
- JSON RPC clients can be easily constructed, too
- ...using Maven

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC
- We need again the `Complex` number bean class

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC
- We need again the `Complex` number bean class
- We specify the functionality as an interface (`CalculatorInterface`)

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC
- We need again the `Complex` number bean class
- We specify the functionality as an interface (`CalculatorInterface`)
- The above two classes will also be used in the client later

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC
- We need again the `Complex` number bean class
- We specify the functionality as an interface (`CalculatorInterface`)
- The above two classes will also be used in the client later
- We then make a Java Servlet which provides the calculator JSON RPC service

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC
- We need again the `Complex` number bean class
- We specify the functionality as an interface (`CalculatorInterface`)
- The above two classes will also be used in the client later
- We then make a Java Servlet which provides the calculator JSON RPC service
- We specify the servlet in the `web.xml` ...

Calculator Example Server

- We now take the calculator example from the web services example set and translate it to JSON RPC
- We need again the `Complex` number bean class
- We specify the functionality as an interface (`CalculatorInterface`)
- The above two classes will also be used in the client later
- We then make a Java Servlet which provides the calculator JSON RPC service
- We specify the servlet in the `web.xml` . . .
- . . . build everything with Maven. . . . . . and deploy it to the GlassFish servlet container. . .

Listing: A class for complex numbers (Complex.java).

```
package calculator;

import java.io.Serializable;

public class Complex implements Serializable {
  private static final long serialVersionUID = 1L;

  private double m_a;
  private double m_b;

  public Complex() {
    super();
  }

  public Complex(final double a, final double b) {
    this();
    this.setRealPart(a);
    this.setImaginaryPart(b);
  }

  public double getRealPart() {
    return this.m_a;
  }

  public void setRealPart(final double a) {
    this.m_a = a;
  }

  public double getImaginaryPart() {
    return this.m_b;
  }

  public void setImaginaryPart(final double b) {
    this.m_b = b;
  }
}
```

Service Interface

Listing: A service interface specifying the operation the service provides (CalculatorInterface.java).

```
package calculator;

public interface CalculatorInterface {

  public abstract Complex add(final Complex x, final Complex y);

  public abstract Complex sub(final Complex x, final Complex y);

  public abstract Complex multiply(final Complex x, final Complex y);

  public abstract Complex divide(final Complex x, final Complex y);
}
```

Listing: The calculator serving implementation the JSON RPC (Calculator.java).

```
package calculator;

public class Calculator implements CalculatorInterface {
  public Calculator() {
    super();
  }

  @Override
  public Complex add(final Complex x, final Complex y) {
    return new Complex(//
        (x.getRealPart() + y.getRealPart()), //
        (x.getImaginaryPart() + y.getImaginaryPart()));
  }

  @Override
  public Complex sub(final Complex x, final Complex y) {
    return new Complex(//
        (x.getRealPart() - y.getRealPart()), //
        (x.getImaginaryPart() - y.getImaginaryPart()));
  }

  @Override
  public Complex multiply(final Complex x, final Complex y) {
    final double a1 = x.getRealPart();
    final double b1 = x.getImaginaryPart();
    final double a2 = y.getRealPart();
    final double b2 = y.getImaginaryPart();

    return new Complex(//
        ((a1 * a2) - (b1 * b2)), //
        ((a1 * b2) + (b1 * a2)));
  }

  @Override
  public Complex divide(final Complex x, final Complex y) {
    final double a1 = x.getRealPart();
    final double b1 = x.getImaginaryPart();
    final double a2 = y.getRealPart();
    final double b2 = y.getImaginaryPart();

    return new Complex(//
        (((a1 * a2) + (b1 * b2)) / ((a2 * a2) + (b2 * b2))), //
        (((a2 * b1) - (b2 * a1)) / ((a2 * a2) + (b2 * b2))));
  }
}
```

Listing: The calculator servlet serving the RPC (CalculatorServlet.java).

```
package calculator;

import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.googlecode.jsonrpc4j.JsonRpcServer;

public class CalculatorServlet extends HttpServlet {

  private Calculator service;
  private JsonRpcServer jsonRpcServer;

  @Override
  protected void doPost(final HttpServletRequest req,
      final HttpServletResponse resp) throws IOException {
    this.jsonRpcServer.handle(req, resp);
  }

  @Override
  protected void doGet(final HttpServletRequest req,
      final HttpServletResponse resp) throws IOException {
    this.jsonRpcServer.handle(req, resp);
  }

  @Override
  public void init(final ServletConfig config) {
    this.service = new Calculator();
    this.jsonRpcServer = new JsonRpcServer(new ObjectMapper(),
        this.service, CalculatorInterface.class);
  }
}
```

web.xml

Listing: The web.xml file (web.xml).

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee␣
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

  <servlet>
    <servlet-name>Calculator</servlet-name>
    <servlet-class>calculator.CalculatorServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Calculator</servlet-name>
    <url-pattern>/Calculator</url-pattern>
  </servlet-mapping>
</web-app>
```

Calculator Example Client

- Making a JSON RPC client is easier than creating web service clients with Axis

Calculator Example Client

- Making a JSON RPC client is easier than creating web service clients with Axis
- We re-use the service interface and the `Complex` class

Calculator Example Client

- Making a JSON RPC client is easier than creating web service clients with Axis
- We re-use the service interface and the `Complex` class
- A simple `TestClient` is developed

Calculator Example Client

- Making a JSON RPC client is easier than creating web service clients with Axis
- We re-use the service interface and the `Complex` class
- A simple `TestClient` is developed
- No code generation at all

Listing: The client for the calculator JSON RPC service (TestClient.java).

```
package calculatorClient;

import java.net.URL;

import com.googlecode.jsonrpc4j.JsonRpcHttpClient;
import com.googlecode.jsonrpc4j.ProxyUtil;

public class TestClient {
  public static void main(final String[] args) throws Throwable {
    CalculatorInterface calculator;
    JsonRpcHttpClient client;
    Complex m, n, res;

    client = new JsonRpcHttpClient(
        new URL("http://localhost:8080/calculatorJSONServer/Calculator"));

    calculator = ProxyUtil.createClientProxy(
        TestClient.class.getClassLoader(), CalculatorInterface.class,
        client);

    m = new Complex();
    m.setRealPart(3);
    m.setImaginaryPart(11);

    n = new Complex();
    n.setRealPart(5);
    n.setImaginaryPart(7);

    res = calculator.multiply(m, n);

    System.out.println(res.getRealPart());
    System.out.println(res.getImaginaryPart());
  }
}
```

Warehouse Stand-Alone Example Server

- We now take the warehouse example from the web services example set and translate it to JSON RPC

Warehouse Stand-Alone Example Server

- We now take the warehouse example from the web services example set and translate it to JSON RPC
- We specify the functionality as an interface (`WarehouseInterface`)

Warehouse Stand-Alone Example Server

- We now take the warehouse example from the web services example set and translate it to JSON RPC
- We specify the functionality as an interface (`WarehouseInterface`)
- We implement the functionality (`Warehouse`)

Warehouse Stand-Alone Example Server

- We now take the warehouse example from the web services example set and translate it to JSON RPC
- We specify the functionality as an interface (`WarehouseInterface`)
- We implement the functionality (`Warehouse`)
- We make a servlet exposing the functionality (`WarehouseServlet`)

Warehouse Stand-Alone Example Server

- We now take the warehouse example from the web services example set and translate it to JSON RPC
- We specify the functionality as an interface (`WarehouseInterface`)
- We implement the functionality (`Warehouse`)
- We make a servlet exposing the functionality (`WarehouseServlet`)
- We make a main class starting the stand-alone Jetty container (`Main`)

Warehouse Interface

Listing: A warehouse functionality specification (WarehouseInterface.java).

```
package warehouseServer;

public interface WarehouseInterface {

  public abstract int getStock(final String type);

  public abstract void changeStock(final String type, final int add);
}
```

Warehouse Implementation

Listing: The implementation of the warehouse service (Warehouse.java).

```
package warehouseServer;

import java.util.HashMap;
import java.util.Map;

public class Warehouse implements WarehouseInterface {
  private final Map<String, Integer> stock;

  public Warehouse() {
    super();
    this.stock = new HashMap<>();
  }

  public synchronized int getStock(final String type) {
    Integer i = stock.get(type);
    return (i == null) ? 0 : i.intValue();
  }

  public synchronized void changeStock(final String type, final int add) {
    Integer i = this.stock.get(type);
    i = Integer.valueOf(((i != null) ? i.intValue() : 0) + add);
    this.stock.put(type, i);
  }
}
```

Warehouse Servlet

Listing: The warehouse servlet serving the RPC (WarehouseServlet.java).

```
package warehouseServer;

import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.googlecode.jsonrpc4j.JsonRpcServer;

public class WarehouseServlet extends HttpServlet {

  private Warehouse service;
  private JsonRpcServer jsonRpcServer;

  @Override
  protected void doPost(final HttpServletRequest req,
      final HttpServletResponse resp) throws IOException {
    this.jsonRpcServer.handle(req, resp);
  }

  @Override
  protected void doGet(final HttpServletRequest req,
      final HttpServletResponse resp) throws IOException {
    this.jsonRpcServer.handle(req, resp);
  }

  @Override
  public void init(final ServletConfig config) {
    this.service = new Warehouse();
    this.jsonRpcServer = new JsonRpcServer(new ObjectMapper(),
        this.service, WarehouseInterface.class);
  }
}
```

Listing: The main class of the stand-alone JSON RPC server (Main.java).

```
package warehouseServer;

import java.io.File;    import java.io.IOException;   import java.net.URL;
import java.net.URLClassLoader;

import org.apache.tomcat.InstanceManager;   import org.apache.tomcat.SimpleInstanceManager;
import org.eclipse.jetty.annotations.ServletContainerInitializersStarter;
import org.eclipse.jetty.server.Server;     import org.eclipse.jetty.server.ServerConnector;
import org.eclipse.jetty.webapp.WebAppContext;

/**
 * The main entry point for our warehouse JSON RPC server. It instantiates
 * the embedded Jetty server and setups up our {@linkplain WarehouseServlet
 * warehouse servlet}.
 */
public class Main {
  /**
   * This is the main routine, which starts the application
   *
   * @param args
   *          the command line arguments
   * @throws Exception
   *           if it has to
   */
  @SuppressWarnings("resource")
  public static void main(final String[] args) throws Exception {
    final Server server;
    final WebAppContext context;

    server = new Server();
    try (ServerConnector connector = new ServerConnector(server)) {
      connector.setPort(8080);
      server.addConnector(connector);

      context = new WebAppContext();
      context.setContextPath("/"); //$NON-NLS-1$
      context.setAttribute("javax.servlet.context.tempdir", //$NON-NLS-1$
          Main.__getScratchDir());

      context.setResourceBase(Main.class.getResource("/webapp/") //$NON-NLS-1$
          .toURI().toASCIIString());

      context.setAttribute(InstanceManager.class.getName(),
          new SimpleInstanceManager());

      context.addBean(new ServletContainerInitializersStarter(context), true);
      context.setClassLoader(new URLClassLoader(new URL[0], Main.class.getClassLoader()));

      context.addServlet(WarehouseServlet.class, "/");//$NON-NLS-1$
      server.setHandler(context);

      server.start(); // start the server and wait for termination
      server.join();
    }
  }
```

Warehouse Example Client

- We re-use the service interface `WarehouseInterface` class

Warehouse Example Client

- We re-use the service interface `WarehouseInterface` class

Warehouse Example Client

- We re-use the service interface `WarehouseInterface` class
- A simple `WarehouseTest` is developed

Warehouse Example Client

- We re-use the service interface `WarehouseInterface` class
- A simple `WarehouseTest` is developed
- No code generation at all

Warehouse Client

Listing: The client for the warehouse JSON RPC service (WarehouseTest.java).

```
package warehouseClient;

import java.net.URL;

import com.googlecode.jsonrpc4j.JsonRpcHttpClient;
import com.googlecode.jsonrpc4j.ProxyUtil;

public class WarehouseTest {
  public static void main(String[] args) throws Exception {
    JsonRpcHttpClient client;
    WarehouseInterface warehouse;

    client = new JsonRpcHttpClient(new URL("http://localhost:8080/"));

    warehouse = ProxyUtil.createClientProxy(
        WarehouseTest.class.getClassLoader(), WarehouseInterface.class,
        client);

    System.out.println("#␣of␣cars:␣" + warehouse.getStock("cars"));
    System.out.println("#␣of␣cats:␣" + warehouse.getStock("cats"));
    warehouse.changeStock("cars", 100);
    warehouse.changeStock("cats", 120);
    System.out.println("#␣of␣cars:␣" + warehouse.getStock("cars"));
    System.out.println("#␣of␣cats:␣" + warehouse.getStock("cats"));
    warehouse.changeStock("cats", -12);
    System.out.println("#␣of␣cars:␣" + warehouse.getStock("cars"));
    System.out.println("#␣of␣cats:␣" + warehouse.getStock("cats"));
  }
}
```

Summary

- There are several simple alternatives to web services
- They lack the broad tool and framework support that web services have, but they may be faster and simpler to use
- JSON RPC is one example of such an alternative

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. David Chappell. The trouble with corba, May 1998. URL http://www.davidchappell.com/articles/article_Trouble_CORBA.html.
2. Robert Orfali, Dan Harkey, and Daniel J. Edwards. *Client/Server Survival Guide*. Wiley Computer Publishing. New York, NY, USA: John Wiley & Sons Ltd., 1999. ISBN 0471316156 and 9780471316152. URL http://books.google.de/books?id=fBteTDjMwScC.
3. Plano, TX, USA: Electronic Data Systems Corporation (EDS), Palo Alto, CA, USA: Hewlett-Packard Company, Cambridge, MA, USA: IBM Corporation, Cambridge Scientific, Mobile, AL, USA: ICON Computing, Andover, MA, USA: I-Logix Inc., Santa Clara, CA, USA: IntelliCorp, Redmond, WA, USA: Microsoft Corporation, Needham, MA, USA: Object Management Group (OMG), Kanata, ON, Canada: ObjecTime Limited, Redwood Shores, CA, USA: Oracle Corporation, Houston, TX, USA: Platinum Technology, Inc., Quincy, MA, USA: Ptech Inc., Cupertino, CA, USA: Rational Software Corporation, London, UK: Reich Technologies, Paris, France: SOFTEAM, Dallas, TX, USA: Sterling Software, Oslo, Østlandet, Norway: Taskon A/S, and Blue Bell, PA, United States: Unisys. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 – Part 1: CORBA Interfaces*, volume formal/2011-11-01. Needham, MA, USA: Object Management Group (OMG), November 2011. URL http://www.omg.org/spec/CORBA/3.2/Interfaces/PDF/.
4. Plano, TX, USA: Electronic Data Systems Corporation (EDS), Palo Alto, CA, USA: Hewlett-Packard Company, Cambridge, MA, USA: IBM Corporation, Cambridge Scientific, Mobile, AL, USA: ICON Computing, Andover, MA, USA: I-Logix Inc., Santa Clara, CA, USA: IntelliCorp, Redmond, WA, USA: Microsoft Corporation, Needham, MA, USA: Object Management Group (OMG), Kanata, ON, Canada: ObjecTime Limited, Redwood Shores, CA, USA: Oracle Corporation, Houston, TX, USA: Platinum Technology, Inc., Quincy, MA, USA: Ptech Inc., Cupertino, CA, USA: Rational Software Corporation, London, UK: Reich Technologies, Paris, France: SOFTEAM, Dallas, TX, USA: Sterling Software, Oslo, Østlandet, Norway: Taskon A/S, and Blue Bell, PA, United States: Unisys. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 – Part 2: CORBA Interoperability*, volume formal/2011-11-02. Needham, MA, USA: Object Management Group (OMG), November 2011. URL http://www.omg.org/spec/CORBA/3.2/Interoperability/PDF.

Bibliography II

5. Plano, TX, USA: Electronic Data Systems Corporation (EDS), Palo Alto, CA, USA: Hewlett-Packard Company, Cambridge, MA, USA: IBM Corporation, Cambridge Scientific, Mobile, AL, USA: ICON Computing, Andover, MA, USA: I-Logix Inc., Santa Clara, CA, USA: IntelliCorp, Redmond, WA, USA: Microsoft Corporation, Needham, MA, USA: Object Management Group (OMG), Kanata, ON, Canada: ObjecTime Limited, Redwood Shores, CA, USA: Oracle Corporation, Houston, TX, USA: Platinum Technology, Inc., Quincy, MA, USA: Ptech Inc., Cupertino, CA, USA: Rational Software Corporation, London, UK: Reich Technologies, Paris, France: SOFTEAM, Dallas, TX, USA: Sterling Software, Oslo, Østlandet, Norway: Taskon A/S, and Blue Bell, PA, United States: Unisys. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.2 – Part 3: CORBA Component Model*, volume formal/2011-11-03. Needham, MA, USA: Object Management Group (OMG), November 2011. URL `http://www.omg.org/spec/CORBA/3.2/Components/PDF`.
6. Josef Stepisnik. *Distributed Object-Oriented Architectures: Sockets, Java RMI and CORBA*. Hamburg, Germany: Diplomica Verlag GmbH, 2007. ISBN 3836650339 and 9783836650335. URL `http://books.google.de/books?id=qNGTzYdJt18C`.
7. *[MS-DCOM]: Distributed Component Model (DCOM) Remote Protocol Specification*, volume v20111214. Redmond, WA, USA: Microsoft Corporation, December 14, 2011. URL `http://download.microsoft.com/download/a/e/6/ae6e4142-aa58-45c6-8dcf-a657e5900cd3/%5BMS-DCOM%5D.pdf`.
8. Dino Esposito. .net remoting – design and develop seamless distributed applications for the common language runtime. *Microsoft Developer Network – MSDN Magazin*, 2(10), October 2002. URL `http://msdn.microsoft.com/en-us/magazine/cc188927.aspx`.
9. Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the java system. In Douglas C. Schmidt and Doug Lea, editors, *Proceedings of the USENIX 1996 Conference on Object-Oriented Technologies (COOTS)*, Toronto, ON, Canada, 1996. URL `http://pdos.csail.mit.edu/6.824/papers/waldo-rmi.pdf`.
10. William Grosso. *Java RMI*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2011. ISBN 1449315356 and 9781449315351. URL `http://books.google.de/books?id=TeK5uL2dWwQC`.
11. Nigel McFarland. *Rapid Application Development With Mozilla*. Bruce Perens' Open Source Series. Upper Saddle River, NJ, USA: Prentice Hall Professional, 2004. ISBN 0131423436 and 9780131423435. URL `http://books.google.de/books?id=gKeRXPkSAVMC`.
12. Simon St. Laurent, Joe Johnston, and Edd Dumbill. *Programming Web Services with XML-RPC*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2001. ISBN 0596001193 and 9780596001193. URL `http://books.google.de/books?id=l4OnvlrjWL0C`.

Bibliography III

13. Tim Bray and USA: Google Inc. Mountain View, CA, editors. *The JavaScript Object Notation (JSON) Data Interchange Format*, volume 7159 of *Request for Comments (RFC)*. Network Working Group, March 2014. URL `http://tools.ietf.org/html/rfc7159`.
14. *The JSON Data Interchange Format*, volume ECMA-404. Geneva, Switzerland: ECMA International, 1 edition, October 2013. URL `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`.
15. *ISO/IEC 16262:2011(E): Information Technology – Programming Languages, their Environments and System Software Interfaces – ECMAScript Language Specification*. Geneva, Switzerland: International Organization for Standardization (ISO), June 15, 2011. URL `http://standards.iso.org/ittf/PubliclyAvailableStandards/c055755_ISO_IEC_16262_2011(E).zip`.
16. Matt Morley. Json-rpc: A light weight remote procedure call protocol. it is designed to be simple!, 2014. URL `http://www.jsonrpc.org/`.