Services

Ralf Lämmel
Software Languages Team
University of Koblenz-Landau
This lecture:
3 ways of architecting services

- RPC (RMI)
- WebService
- REST

In different terms: today’s lecture is about distributed programming.
Remote procedure call (RPC)

An RPC is „an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

Elevator speech

What if the objects need to be distributed in a network? How to communicate between different computers in an OOP-friendly manner? Basically, we want to continue to hold references to objects (perhaps remote objects), and send them messages (i.e., perform method calls) as before.

The **Java Remote Method Invocation** (Java RMI) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage collection. [Source: http://en.wikipedia.org/wiki/Java_remote_method_invocation]
Non-distributed programming

Caller and callee are on the same machine.

Object1 (Client)  

Method Invocation
- Arguments are evaluated.
- Caller location is pushed onto stack.
- Callee method is executed.
- Result is returned.
- Caller location resumes.

Object2 (Server)  

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Distributed programming

Caller and callee are on different machines.

Object1 (Client)

Remote Method Invocation

Client holds on a server proxy. Args + result go over the network.

Object2 (Server)

Host mymac.foo.edu

Host myserver.bar.edu

In demo code, client and server may be on the same machine, perhaps even on the same JVM, but the invocation is handled (conceptually) over the network.

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
A first look at the “hello world” of RMI

DEMO

https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaRmiSamples

See package “helloworld” of javaRmiSamples. Start the server as described in “Makefile”. Start the client by running the “Client” class. Observe how 42 was remotely incremented.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
© 2010-15, Ralf Lämmel, rlæmmel@acm.org
A more interesting scenario: a banking application to access account data remotely at a teller (ATM).

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Communication between different Java runtimes on the same machine or different machines, eventually involves the network layer of the OS.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Stubs and skeletons

- Client invokes a remote method on “stub” object.
- The stub sends the call to the server-side “skeleton”.
- That is, the stub:
  - opens a socket to the remote server,
  - marshals the method parameters,
  - forwards the data stream to the skeleton,
  - awaits and unmarshals result data stream from skeleton.
- The skeleton contains a method that:
  - receives the remote calls,
  - unmarshals the method parameters,
  - invokes the actual remote (by now, local) implementation, and
  - marshals the result and delivers it back to the stub.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Remote-method invocation

Object1 (Client)

Host mymac.foo.edu

Remote Object Stub: serves as Remote Reference

01101

true

odd(3)

Object2 (Server)

Remote Object Skeleton

Host myserver.bar.edu

Same interface as remote object

Serialized method name, parameters and return value

true

10010

odd(3)

01101

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Parameters and return values

- Remote objects – by reference
- Serializable objects – by copy
- Others – cannot be passed (exception)

Acknowledgement: Jürgen Starek has significantly contributed to this content.
How to refer to / look up remote objects?

We need a naming service! A server-side directory that associates remote objects with names (URLs).

Host myserver.bar.edu

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Naming service cont’d

Host mymac.foo.edu

lookup(“Y”)

Remote reference to server

Host myserver.bar.edu

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Designing a client/server app
(Summary)

- **Server**
  - Bind 1+ “service” objects via registry
- **Client**
  - Looks up and interacts with service objects
- For example (see “helloworld” demo):
  - Service: increment operation

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Designing a client/server app
(Detailed steps)

1. Design an interface for service (remote object).
2. Implement the service.
3. (Generate stub and skeleton classes.)
4. Implement a server to contact for binding.
5. Implement a client to invoke service.
6. Start the server.
7. Run the client.

Done automatically.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
© 2010-15, Ralf Lämmel, rlammel@acm.org
Step 1:
Design an interface for service (remote object).

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Service extends Remote {
    public long inc(long x)
    throws RemoteException;
}

This is a regular interface except for the special base interface and the special throws declaration.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Step 2:
Implement the service.

```java
class ServiceImpl
    extends UnicastRemoteObject
    implements Service {

    // Needed for serialization
    private static final long serialVersionUID = 6102178242852627613L;

    // Needed because of exception
    public ServiceImpl() throws RemoteException {
        super();
    }

    public long inc(long x) throws RemoteException {
        return ++x;
    }
}
```

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Step 3: Generate stub and skeleton classes.

- Use RMI compiler.
  \texttt{rmic ServiceImpl}
- Applies to Java 1 only.
- $\geq$ Java 2 uses reflection instead.

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Step 4:
Implement a server to contact for binding.

```java
import java.rmi.Naming;

public class Server {
    public Server() {
        try {
            Service s = new ServiceImpl();
            Naming.rebind("rmi://localhost/Service", s);
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }

    public static void main(String args[]) {
        new Server();
    }
}
```

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Name format for binding

\[ rmi: //<host\_name> \\
[::<name\_service\_port>] \\
/<service\_name> \]

Thus, URLs of a specific form are used for addressing remote objects.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Availability of registry

• Binding requires that the registry is locally running.
• Registry can be started programmatically as follows:

    import java.rmi.registry.LocateRegistry;
    import java.rmi.registry.Registry;
    LocateRegistry.createRegistry(Registry.REGISTRY_PORT);

• Default TCP/IP port: 1099

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Usage of “localhost”

- localhost may fail to work in binding.
- This depends on the network configuration.
- Alternatively, one can use the following idiom:

```java
InetAddress addr = InetAddress.getLocalHost();
String hostname = addr.getHostName();
Naming.rebind("rmi://"+hostname+"/Service", s);
```
Step 5: Implement a client to invoke service.

```java
import java.rmi.Naming;

public class Client {
    public static void main(String[] args) {
        try {
            Service s = (Service) Naming.lookup("rmi://localhost/Service");
            System.out.println(s.inc(41));
        } catch (Exception e) {
            ...
        }
    }
}
```

Acknowledgement: Jürgen Starek has significantly contributed to this content.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Step 6: Start the server.

JavaRmi> java helloworld.Server

One may want to run the server (and the client) with a security policy.

Server must be killed, e.g., with CTRL-C

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Step 7: Run the client.

JavaRmi> java helloworld.Client
42
Components of an RMI-based application

- Services of remote objects (interfaces)
- Implementation of services (classes)
- Proxies for client-side (classes, generated)
- A server that binds services (class)
- A client that looks up services (class)
- The RMI registry (component of RMI layer)
- Security policies for client and server (optional)

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Security of RMI

• Useful constrains:
  – Service provision (“May an app provide services?”)
  – Naming lookups (“May an IP address look up services?”)
  – Remote invocations (“May an IP address invoke ...?”)

• Use a non-default security manager:
  ```java
  System.setSecurityManager(new SecurityManager());
  ```

• Assign a security policy to an application:
  ```java
  java -Djava.security.policy=mySecurity.policy Server
  ```

Acknowledgement: Jürgen Starek has significantly contributed to this content.
An RMI-based banking app

DEMO

– A non-C/S-based version as a reference: package “banking.local”.
– The C/S-based version: package “banking.remote”.
– Think of a refactoring to derive the latter from the former.
This implementation is interesting in so far that it readies all data objects for RMI. Further, the operation “total” is provided as a service, but “cut” is not. Thus, the client must implement “cut”, which essentially means that all company, department, and employee objects end up as proxies on the client.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
Summary

• RMI easily blends with OOP.
  – A simple form of distributed programming is enabled this way.
  – Client/Server applications are enabled this way.

• RMI – semantics and concepts:
  – Use local proxy objects for access to remote objects.
  – Un-/marshal arguments and results for messages on the wire.
  – Bind objects to names (URLs) that can be used for lookup.

• RMI – programming idioms:
  – Remote objects are looked up from server-side registry.
  – Remote objects may also be returned by RMI calls.
  – Remote objects may be created by factory methods.

Acknowledgement: Jürgen Starek has significantly contributed to this content.
WebServices

A WebService is „a software system designed to support interoperable machine-to-machine interaction over a network“.  
Elevator speech

How to communicate across OS, programming language, type system, protocol? For instance, how to wire up a .NET component with a Java component? Also, how to reuse “services” across the internet/web?

See also: http://www.infoworld.com/d/architecture/infoclipz-service-oriented-architecture-soa-899

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Service-Oriented Architecture

- **Goal**: Combine and reuse
- **Units**: interoperable services
- **Features**: Loose coupling with
  - OS
  - language
  - technologies, etc.

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
SOA: How it can work ...

Service Broker

Publishes service

Finds services

Service Provider

Service

Service contract

Interact

Service Consumer

Client

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Illustration of a service 'geoclient'

https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javawsSamples/geoclient

DEMO

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
package de.unikoblenz.sle;

import net.webservicex.GlobalWeather;

/**
 * Retrieve weather forecast
 *
 * @author Pek
 */

class WhatIsTheWeather {

    public static final void main(String[] args) {
        GlobalWeatherSoap weatherService = new GlobalWeather().getGlobalWeatherSoap();
        String weather = weatherService.getWeather(
            "Koeln / Bonn", // city
            "Germany" // country
        );
        System.out.println(weather);
    }
}

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
package de.unikoblenz.sle;

import net.webservicex.Country;

/**
 * Retrieve information about Germany
 * @author pek
 */
public class WhatGMTandISOare {

    public static void main(String[] args) {
        CountrySoap countryInfo = new Country().getCountrySoap();
        String countryName = "Germany";
        String GMTzone = countryInfo.getGMTbyCountry(countryName);
        String ISOcode = countryInfo.getISOCountryCodeByCountyName(countryName);
        System.out.println(GMTzone);
        System.out.println(ISOcode);
    }
}

<terminated> WhatGMTandISOare [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_45.jdk/Contents/Home/bin/java (May 7, 2015, 8:52:10 AM)
</NewDataSet>
<NewDataSet>
    <Table>
        <Name>Germany</Name>
        <CountryCode>de</CountryCode>
    </Table>
    <Table>
        <Name>Germany</Name>
        <CountryCode>de</CountryCode>
    </Table>
</NewDataSet>
WebService as an implementation of SOA

Service Broker

Publishes

WSDL

Service Provider

Service contract

Interact

Service Consumer

Finds

WSDL

UDDI

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
WebService as an implementation of SOA

Service

Client

Java

Java

WSDL

SOAP

Service contract

XML message

XML message

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
WebService as an implementation of SOA

Java

Service

Service contract

WSDL

XML message

SOAP

.NET

Client

XML message

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
WebService as an implementation of SOA

.NET

Service

WSDL

Service contract

XML message

Java

Client

SOAP

XML message

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Technology requirements for this lecture’s project


Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Running example: **Counting words**

- **Service**: count words of a string
- **Input**: string (space separated words)
- **Output**: int for number of words

https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javawsSamples/wcservice
<message name="Wording_count">
  <part name="count" type="xs:string"/>
</message>

<message name="Wording_countResponse">
  <part name="countResponse" type="xs:int"/>
</message>

<portType name="Wording">
  <operation name="count" parameterOrder="count">
    <input message="tns:Wording_count"></input>
    <output message="tns:Wording_countResponse"></output>
  </operation>
</portType>

<binding name="WordingBinding" type="tns:Wording">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="count">...</operation>
</binding>

<service name="WordingService">
  <port binding="tns:WordingBinding" name="WordingPort">
    <soap:address location="http://127.0.0.1:8080/TextStat/Wording"/>
  </port>
</service>

...
WebServices Description Language 1.1

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Simple Object Access Protocol

- SOAP uses an internet application layer protocol as a transport protocol (SMTP, RPC, HTTP)
- Uses XML as message format
- Platform independent
- Language independent

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
The **Internet Protocol Suite**

**Application Layer**
- BGP
- DHCP
- DNS
- FTP
- GTP
- HTTP
- IMAP
- IRC
- LDAP
- Megaco
- MGCP
- NNTP
- NTP
- POP
- RIP
- RPC
- RTP
- RTSP
- SDP
- SIP
- SMTP
- SNMP
- SOAP
- SSH
- Telnet
- TLS/SSL
- XMPP

**Transport Layer**
- TCP
- UDP
- DCCP
- SCTP
- RSVP
- ECN

**Internet Layer**
- IP (IPv4, IPv6)
- ICMP
- ICMPv6
- IGMP
- IPsec

**Link Layer**
- ARP/InARP
- NDP
- OSPF
- Tunnels (L2TP)
- PPP
- Media Access Control (Ethernet, DSL, ISDN, FDDI)

http://en.wikipedia.org/wiki/Application_Layer

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek. © 2010-15, Ralf Lämmel, rlammel@acm.org
A sample SOAP message

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body
xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>

http://en.wikipedia.org/wiki/SOAP

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
SOAP messages for the running example

Request

```xml
<envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

 cuerpo>
  <ns2:count xmlns:NS2="http://sle.unikoblenz.de/"
    Hello .NET
  </ns2:count>
</S:Body>
</S:Envelop>
```

Response

```xml
<envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

 cuerpo>
  <ns1:countResponse xmlns:NS1="http://sle.unikoblenz.de/"
    2
  </ns1:countResponse>
</S:Body>
</S:Envelope>
```
Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Java API for XML Web Services (JAX-WS)

- Annotations for WebServices creation
- Part of Java SE 6
- Part of Java EE 5 platform

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
JAX-WS (Server side)

- Write simple WebService.
- Deploy it locally or remotely.
- Fetch WSDL.
WebService code

Generated WSDL

@WebService
public class Wording {
    @WebMethod
    public int count (String text) {
        // TODO this is just a stub
        return 42;
    }
}

<message name="Wording_count">
    <part name="count" type="xs:string"/>
</message>

<message name="Wording_countResponse">
    <part name="countResponse" type="xs:int"/>
</message>

<portType name="Wording">
    <operation name="count">
        <input message="tns:Wording_count"/>
        <output message="tns:Wording_countResponse"/>
    </operation>
</portType>

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
## `@javax.jws.WebService`

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the <code>wsdl:portType</code></td>
<td>The unqualified name of the Java class or interface</td>
</tr>
<tr>
<td>targetNamespace</td>
<td>XML namespace of WSDL and some XML elements generated from this web service</td>
<td>The namespace mapped from the package name containing the web service</td>
</tr>
<tr>
<td>serviceName</td>
<td>The Service name of the web service: <code>wsdl:service</code></td>
<td>The unqualified name of the Java class or interface + &quot;Service&quot;</td>
</tr>
<tr>
<td>endpointInterface</td>
<td>Qualified name of SEI. This allows separation of interface contract from implementation.</td>
<td>None – If not specified, the endpoint implementation class is used to generate the contract.</td>
</tr>
<tr>
<td>portName</td>
<td>The <code>wsdl:portName</code></td>
<td>The <code>WebService.name + &quot;Port&quot;</code></td>
</tr>
</tbody>
</table>

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.

© 2010-15, Ralf Lämmel, rlaemmel@acm.org
### @javax.jws.WebMethod

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>operationName</td>
<td>The name of the wsdl:operation matching this method.</td>
<td>The name of the Java method</td>
</tr>
<tr>
<td>action</td>
<td>XML namespace of WSDL and some XML elements generated from this web service</td>
<td>&quot;&quot; (empty string)</td>
</tr>
<tr>
<td>exclude</td>
<td>Used to exclude a method from the WebService.</td>
<td>false</td>
</tr>
</tbody>
</table>
### @javax.jws.WebParam

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>wsdl:part / local name of the XML element</td>
<td>&quot;argX&quot; (X starts with 0)</td>
</tr>
<tr>
<td>targetNamespace</td>
<td>The XML namespace of the XML element for the parameter</td>
<td>The targetNamespace for the web service</td>
</tr>
</tbody>
</table>

### @javax.jws.WebResult

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>wsdl:part / local name of the XML element</td>
<td>return ... or ..._response</td>
</tr>
<tr>
<td>targetNamespace</td>
<td>The XML namespace of the XML element for the parameter</td>
<td>The targetNamespace for the web service</td>
</tr>
</tbody>
</table>

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
JAX-WS (Client side)

- Generate dynamic proxy.
- Write client code.
- Test WebService locally or remotely.
JAX-WS (Client side)

```bash
> $JAVA_6_HOME/bin/wsimport URL -keep
```

<table>
<thead>
<tr>
<th>Generated file</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count.java</td>
<td>Wrapper for request message</td>
</tr>
<tr>
<td>CountResponse.java</td>
<td>Wrapper for response message</td>
</tr>
<tr>
<td>ObjectFactory.java</td>
<td>JAXB XML Registry</td>
</tr>
<tr>
<td>package-info.java</td>
<td>JAXB package annotations</td>
</tr>
<tr>
<td>Wording.java</td>
<td>Service Endpoint Interface</td>
</tr>
<tr>
<td>WordingService.java</td>
<td>WSDL service</td>
</tr>
</tbody>
</table>

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
@WebServiceClient(
    name = "WordingService",
    targetNamespace = "http://sle.unikoblenz.de/",
)
public class WordingService extends Service {

    public WordingService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public WordingService() {
        super(WORDINGSERVICE_WSDL_LOCATION,
            new QName("http://sle.unikoblenz.de/", "WordingService"));
    }

    @WebEndpoint(name = "WordingPort")
    public Wording getWordingPort() {
        return (Wording) super.getPort(
            new QName("http://sle.unikoblenz.de/", "WordingPort"),
            Wording.class);
    }

    }

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Dynamic proxy implementing SEI

JAXB use for WSDL⇔Java mapping

Service Endpoint method

@WebService(
  name = "Wording",
  targetNamespace = "http://sle.unikoblenz.de/")
@XmlSeeAlso({ObjectFactory.class})

public interface Wording {

  @WebResult(targetNamespace = "")
  @RequestWrapper(localName = "count",
  targetNamespace = "http://sle.unikoblenz.de/",
  className = "de.unikoblenz.sle.Count")
  @ResponseWrapper(localName = "countResponse",
  targetNamespace = "http://sle.unikoblenz.de/",
  className = "de.unikoblenz.sle.CountResponse")
  @WebMethod
  public int count(
    @WebParam(name = "arg0", targetNamespace = "")
    String arg0);

  }

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
JAX-WS (Client side)

...  

WSDL

Client code

public class Client {
    public static void main(String[] args){
        Wording wording = new WordingService().getWordingPort();
        System.out.println(wording.count("Count this!"));
    }
}

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
Illustration of the wc service/client

DEMO

Start server; see package *wcserver*
Start client; see package *wcclient*
Omitted topics

- WS-*, additional specifications (e.g. WS-Security)
- Orchestration (WS-BPEL)
- Choreography (WS-CDL)
- ...

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Further reading

- http://www.soapprinciples.com
- http://www.w3.org/TR/ws-arch
- Understanding SOA with Web Services, by Eric Newcomer, Greg Lomow
- http://www.w3.org/TR/wsd1
- http://www.w3.org/TR/soap12-part1
- https://jax-ws.dev.java.net/
- https://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html

Acknowledgement: Much of this content has been originally authored by Ekaterina Pek.
© 2010-15, Ralf Lämmel, rlaemmel@acm.org
Representational State Transfer (REST)

REST is „a software architecture style consisting of guidelines and best practices for creating scalable web services“. [Source: http://en.wikipedia.org/wiki/Representational_state_transfer]
The API notion

- API — Application Programming Interface
  - An interface for functionality that is reusable by programs
- Web API
  - An interface provided over the Web (using HTTP)
Intermezzo: read data on the network instead of the file system
Read XML from the file system

http://101companies.org/wiki/Contribution:jdom

```java
package org.softlang.company.features;

import java.io.File;
import java.io.IOException;
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

public class Parsing {

    public static Document parseCompany(String file) throws IOException, JDOMException {
        return new SAXBuilder().build(new File(file));
    }
}
```

Use SAX-based event processing to construct DOM objects from file content

Use file input stream to process input
Read XML from the file system

http://101companies.org/wiki/Contribution:jdom

```java
package org.softlang.company.features;

import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import java.net.URL;

public class Parsing {

    public static Document parseCompany(String url) throws IOException, JDOMException {
        return new SAXBuilder().build(new URL(url).openStream());
    }
}
```

Use the very same approach except for the kind of stream

Use „HTTP GET“ stream
More pointers for comparison


The difference between file versus network stream is minimal both in the case of XML and JSON.
Aspects of a RESTful web service API

• A base URI such as http://example.com/resources/
• An internet media type such as JSON
• Standard HTTP methods: GET, PUT, DELETE, POST
• Links to reference state such as http://.../resources/1
Constraints of REST — Representational state transfer

• „Client-server“: data on server, UI on client, …
• „Stateless“: no client context being stored on server
• „Cacheable“: responses define themselves as cacheable

REST is an architectural style — not a protocol
The following table shows how the HTTP methods are typically used to implement a RESTful API.

<table>
<thead>
<tr>
<th>Resource</th>
<th>GET</th>
<th>PUT</th>
<th>POST</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection URI, such as <a href="http://example.com/resources">http://example.com/resources</a></td>
<td>List the URIs and perhaps other details of the collection's members.</td>
<td>Replace the entire collection with another collection.</td>
<td>Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.</td>
<td>Delete the entire collection.</td>
</tr>
<tr>
<td>Element URI, such as <a href="http://example.com/resources/item17">http://example.com/resources/item17</a></td>
<td>Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.</td>
<td>Replace the addressed member of the collection, or if it doesn't exist, create it.</td>
<td>Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.</td>
<td>Delete the addressed member of the collection.</td>
</tr>
</tbody>
</table>
A Web API for the „Polls“ App
An API for the Polls app

- **index**: Return all polls
  ```json
  {
  "1": "What's the coolest language?“, 
  "2": "Where do you go tomorrow?"
  }
  ```
- **detail**: Return choices with numbers of votes for a poll
  ```json
  {
  "1": {"votes": 5, "choice": „Cobol"}, 
  "2": {"votes": 90, "choice": "Python"}, …
  }
  ```
- **vote**: Count a vote towards a choice of a poll
  ```
  POST
  ```
An API client

A client using the API of the Django-based Polls app at https://github.com/rlaemmel/mysite

There are these scripts for accessing the API:

- index.py: list all polls
- detail.py: list details for a given poll
- vote.py: vote on a given choice of a given poll
- bot.py: continuously vote to maintain a given choice of a given poll as the most popular one

This example is used in a software development lecture to explain basics of REST.
$ python detail.py 1
{5: {u'votes': 1, u'choice': u'Nowhere'}, 6: {u'votes': 10, u'choice': u'Microsoft'}, 7: {u'votes': 11, u'choice': u'Google'}}

$ python bot.py 1 2
Starting bot for poll 1 and choice 2 ...
The question for poll ID 1 is "What's the coolest language?".
The choice for choice ID 2 is "Python" with current popularity 90.
Popularity challenged by "Haskell" with current popularity 91.
Vote!
The resulting popularity of "Python" is 91.
Popularity challenged by "Haskell" with current popularity 91.
Vote!
The resulting popularity of "Python" is 92.
Popularity maintained.
Popularity maintained.
Popularity maintained.
^C
Finishing
End of Lecture