Client-Server Architecture

Client-Server Example

Client-Server Interaction as Function Calls

Socket Programming – Client

RPC and RMI
RMI – Server

- Create a service interface
  - Remote interface
  - Declares the methods to be remotely invoked
- Create a service implementation
  - Remote object
  - Implements the methods to be remotely invoked
- Register the service with a RMI registry so a client can find and use this service

RMI – Client

- Connect to the RMI registry
- Look up the service by name
- Invoke the service

RMI Example: AuthService

- Shared by both server and client
  - AuthService
  - User
- Server
  - AuthServiceImpl
  - AuthServiceStartup
- Client
  - AuthServiceClient

Why does User have to implement the Serializable interface? What exactly does registry.lookup() return?

How RMI Works

1. Lookup
2. Stub (proxy)
3. Method invocation
4. Parameters
5. Result
6. Return result

Cross Platform RPC

- The client and the server use different languages and/or platforms

How do we define service interface??

CORBA

- Common Object Request Broker Architecture
- Use Interface Definition Language (IDL) to describe service interface
- Provide mappings from IDL to other languages such as Java, C++, and so on.
IDL Example

```idl
module bank {
    interface BankAccount {
        void setbalance(in long acnum, in long balance) raises (ACCOUNT_ERROR);
        string queryaddress(in long acnum) raises (ACCOUNT_ERROR);
        long querybalance(in long acnum) raises (ACCOUNT_ERROR);
        exception ACCOUNT_ERROR { long errcode; string message;};
    }
};
```

Web Services

- **RPC over HTTP**
  - Client and server communicate using HTTP requests and responses

Metro

- [http://metro.java.net/](http://metro.java.net/)
- A Java web service library backed by SUN/Oracle
- Implementation of the latest Java web service specifications
- Guaranteed interoperability with .NET
- Windows Communication Foundation (WCF) web services
- Easy to use

Other Java Web Service Libraries

- **Apache Axis2**
  - [http://axis.apache.org/axis2/java/core/](http://axis.apache.org/axis2/java/core/)
- **Apache CXF**

Web Service Example: HashService

- **HashService**
  - @WebService
  - @WebMethod
- **web.xml**
- **sun-jaxws.xml**
  - `<endpoint>`

WSDL

- A language for describing web services
  - Where the service is
  - What the service does
  - How to invoke the operations of the service
- Plays a role similar to IDF in CORBA
Sample WSDL Documents

- **Amazon ECS** - http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

How Do We Describe an API

```
interface Foo {
    int bar( String, BigDecimal );
}
```

Return value  Method name  Parameters

How Do We Describe an Web Service API

```
Type
Parameters
Return values
Method name
Interface name
```

WSDL

Web Service Example: Consume HashService

- Generate client side interface and stub from WSDL using Metro's `wsimport`
- Write client code

SOAP

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
    xmlns:xsd=http://www.w3.org/1999/XMLSchema>
    <SOAP-ENV:Body>
        <ns1:doSpellingSuggestion
            xmlns:ns1="urn:GoogleSearch">
            <key xsi:type="xsd:string">britney spears</key>
            <phrase xsi:type="xsd:string">and her space</phrase>
        </ns1:doSpellingSuggestion>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A Sample SOAP Message
SOAP Encoding

- [http://schemas.xmlsoap.org/encoding](http://schemas.xmlsoap.org/encoding)
- Include all built-in data types of XML Schema Part 2: Datatypes
  - `xsi` and `xsd` namespace

Compound Values and Other Rules

```xml
<isArray xsi:type="SOAP-ENC:Array" SOAP-ENC:ArrayType="xsd:int[3]">
  <val>10</val>
  <val>20</val>
  <val>30</val>
</isArray>

<Sample>
  <Val xsi:type="xsd:int">10</Val>
  <Val xsi:type="xsd:string">Ten</Val>
</Sample>

- References, default values, custom types, complex types, custom serialization ...

A Sample SOAP RPC Response

```xml
<SOAP-ENV:Envelope
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <return xsi:type="xsd:string">Britney Spears</return>
  </doSpellingSuggestionResponse>
</SOAP-ENV:Envelope>
```

A Sample Fault Response

```xml
<SOAP-ENV:Envelope
  SOAP-ENV:encodingsStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <faultcode>SOAP-ENV:Client</faultcode>
  <faultstring>client error</faultstring>
  <message>Invalid Currency</message>
  <errorcode>1234</errorcode>
</SOAP-ENV:Fault>
</SOAP-ENV:Envelope>
```

UDDI

- Universal Description Discovery and Integration
- A registry for web services
- A web API for publishing, retrieving, and managing information in the registry
**UDDI Registries**

**Other Web Services**

- Differences between web services
  - Language support
    - Single language vs. Language independent
  - Message encoding
    - Text vs. Binary
  - Transport layer
    - HTTP vs. non-HTTP
- **RESTful Web Services**

---

**A RESTful Web Service**

List all users: /users.xml

```xml
<users>
  <user>
    <id>1</id>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
    <email>jdoe@localhost</email>
  </user>
</users>
```

**RESTful Web Services**

- Web applications for programs
  - Generate responses in formats to be read by machines (i.e. XML and JSON) rather than by humans (i.e. HTML)
- Satisfy the REST constraints
  - The *stateless* constraint in particular

---

**REST**

- REpresentational State Transfer
- Introduced by Roy Fielding in his Ph.D. dissertation on network-base software architecture
- Describes the common characteristics of *scalable*, *maintainable*, and *efficient* distributed software systems

**The REST Constraints**

- Client and server
- Stateless
- Support caching
- Uniformly accessible
- Layered
- *(Optional)* support code-on-demand
RESTful Web Service Example

User Management
- List
- Get
- Add
- Update
- Delete

Create a RESTful Web Service

- Identify resources and operations
- Determine resource representation, i.e. data exchange format between the service and the clients
- Design URL and request mapping
- Implement the operations

Resource Representation

- Data format should be easily “understandable” by all programming languages
  - XML
    - Already widely in use as a platform independent data exchange format
    - XML parsers are readily available
  - JSON
    - Much more concise than XML
    - Can be used directly in JavaScript

JSON

- JavaScript Object Notation
- http://json.org/
  - E.g.
    ```
    [{
      "id": 1,
      "firstName": "John",
      "lastName": "Doe",
      "email": "jdoe@localhost"
    }]
    ```

URL Design and Request Mapping Conventions (1)

- Operation: get a user
- URL
  - /user/{id} or
  - /user/get?id={id}

Path variable based design is usually preferred to request parameter based design.

URL Design and Request Mapping Conventions (2)

- Operation: get a user
- Choose which data format to use
- Solution:
  - /user/{id}.{format}
  - Check the Accept request header

Checking Accept header is preferred in theory, but the URL based solution is more convenient in practice, e.g. https://dev.twitter.com/docs/api/1.1
URL Design and Request mapping conventions (3)

- Map HTTP Request Methods to CRUD operations
  - POST (or PUT) ➔ Create
  - GET ➔ Retrieve
  - PUT (or POST) ➔ Update
  - DELETE ➔ Delete

Request Mapping Example

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get a user</td>
<td>GET /user/1 HTTP 1.1</td>
</tr>
<tr>
<td>Delete a user</td>
<td>DELETE /user/1 HTTP 1.1</td>
</tr>
<tr>
<td>Update a user</td>
<td>PUT /user/1 HTTP 1.1</td>
</tr>
<tr>
<td></td>
<td>{ &quot;id&quot;:1,</td>
</tr>
<tr>
<td></td>
<td>&quot;firstName&quot;:&quot;John&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;lastName&quot;:&quot;Doe&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;email&quot;:&quot;jdoe@localhost&quot;}</td>
</tr>
</tbody>
</table>

Service Implementation – Know Your Libraries

- Map HTTP requests to service operations
  - Modern webapp framework like Spring
  - Jersey - https://jersey.java.net/
- Convert between objects and XML/JSON
  - Jackson - http://jackson.codehaus.org/
  - Gson - http://code.google.com/p/google-gson/

Service Implementation Example: Get A User

- Generate response directly
  - HTTP Response
  - Content type: application/json
- Generate response using a JSP
- Generate response using Spring’s MappingJackson2JsonView

Jackson Support in Spring

- Dependency
  - com.fasterxml.jackson.core:jackson-databind
- Additional view resolver
  - BeanNameViewResolver
- Additional view
  - MappingJackson2JsonView

Using Multiple View Resolvers in Spring

- View resolution order
  - Order of the resolver beans, or
  - Based on the order property of the beans
- InternalResourceViewResolver should always be the last
Service Implementation
Example: Update A User

- @RequestBody and @ResponseBody in Spring
- ObjectMapper in Jackson

Advantages of RESTful Web Services (vs. SOAP)

- Do not depend on complex specifications and libraries
  - Easy to implement services
  - Easy to consume services
- Data exchange format is much more concise
  - Easy to generate and debug data
  - More efficient to transfer data
- Take full advantage of infrastructure support for HTTP, e.g. caching

Access RESTful Web Service in Desktop and Mobile Apps

- Apache HttpClient
  - Available on Android
  - Example: restws.client.Client

Summary

- RPC and RMI
- CORBA
  - IDL
- SOAP, WSDL, UDDI
  - Create and consume SOAP web services using Metro
- RESTful web services

Further Readings

- Java Web Services Up and Running by Martin Kalin
- RESTful Java Web Services by Jose Sandoval
- The Rise and Fall of CORBA by Michi Henning