universität innsbruck

Department of Computer Science

STI INNSBRUCK

703128 PS/2 Web Services

REST Services

Zaenal Akbar

Friday, 2017-12-15

Outline

- REST Services
  - Overview
  - Principles
  - Common Errors
- Exercise

What is REST?

- Set of architectural principles used for design of distributed systems
- Focus on a system's resources
- A predominant Web service design model in the last few years – displaced SOAP and WSDL-based interface design because it's a considerably simpler style to use

Principles of REST

1. Use HTTP methods explicitly
2. Be stateless
3. Expose directory structure-like URIs
4. Transfer XML, JavaScript Object Notation (JSON), or both

1. Use HTTP methods explicitly

- We have HTTP protocol on the Web, why not use it?
- CRUD (create, read, update, delete) operations and HTTP methods
  - To create a resource on the server, use **POST**
  - To retrieve a resource, use **GET**
  - To change the state of a resource or to update it, use **PUT**
  - To remove or delete a resource, use **DELETE**

HTTP Usage

- Use of HTTP methods for intended purposes
- The request URI in an HTTP GET request identifies
  - One specific resource
  - Or the query string in a request URI
- Inconsistent use of HTTP GET
  - State-changing operation over HTTP GET
  - Web caching tools (crawlers) and search engines could make server-side changes unintentionally

Use of HTTP POST

Wrong:

```
GET /adduser?name=Robert HTTP/1.1
```

Right:

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
    <name>Robert</name>
</user>
```

Use of HTTP PUT

Wrong:

```
GET /updateuser?name=Robert&newname=Bob HTTP/1.1
```

Right:

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
    <name>Bob</name>
</user>
```

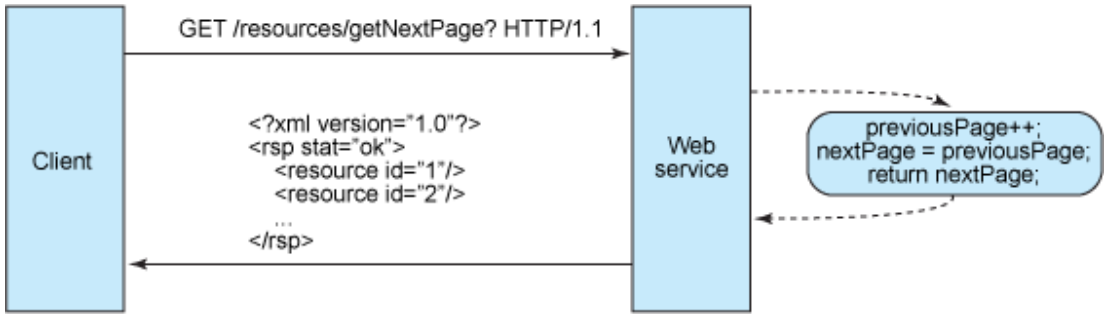
2. Be Stateless (1)

Scalability and performance are needed on the Web

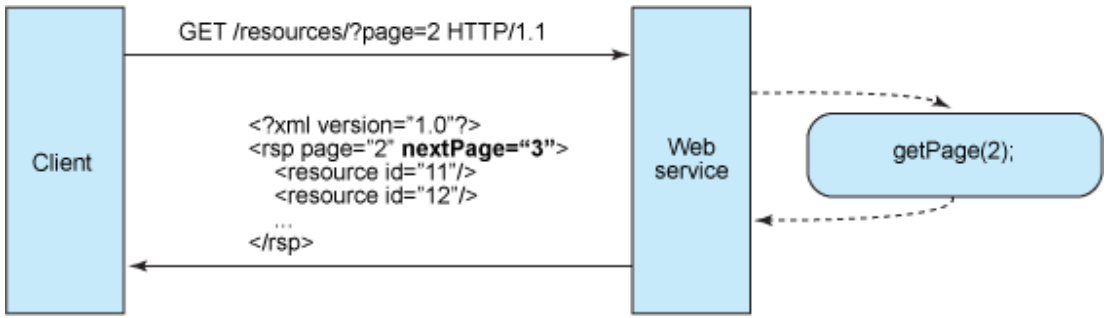
- Network topologies need requests to be self-contained for optimal load-balancing, routing, etc.
- A complete, independent request doesn't require the server, while processing the request, to retrieve any kind of application context or state
- Statelessness simplifies the design and implementation of server-side components – no need to synchronize session data

2. Be Stateless (2)

- Stateful request



- Stateless request



3. Expose directory structure-like URIs

- The structure of a URI should be straightforward, predictable, and easily understood.

- This type of URI is hierarchical, rooted at a single path, and branching from it are sub-paths that expose the service's main areas

  `http://www.myservice.org/discussion/topics/{topic}`

URI Guidelines

1. Hide the server-side scripting technology file extensions (.jsp, .php, .asp).
2. Keep everything lowercase.
3. Substitute spaces with hyphens or underscores (one or the other).
4. Avoid query strings as much as you can.
5. Instead of using the “404 Not Found” code if the request URI is for a partial path, always provide a default page or resource as a response.
6. URIs should also be static so that when the resource changes or the implementation of the service changes, the link stays the same. This allows bookmarking.
7. It is also important that the relationship between resources that is encoded in the URIs remains independent of the way the relationships are represented where they are stored.

4. Transfer XML, JSON, or both

- A resource representation typically reflects the current state of a resource, and its attributes, at the time a client application requests it
- The relationships between data model objects (resources) should be reflected in the way they are represented for transfer to a client application
- Give client applications the ability to request a specific content type that is best suited for them, construct your service so that it makes use of the built-in “HTTP Accept” header

Common REST errors

1. Tunneling everything through GET

   You are using HTTP GET method for “writing” while your URIs do not identify a resource but some operation you would like to perform on the resource

2. Ignoring response codes

   HTTP protocol has an extensive set of application-level response codes that you can use to describe various results of your API calls

3. Ignoring caching
   - HTTP defines a powerful caching mechanism that include “ETag” (“If-None-Match”), “If-Modified-Since” header, and “304 Not Modified” response code
   - Allow clients and servers to negotiate a fresh copy or through caching or proxy servers of a resource

Common REST errors

4. Ignoring hypermedia

   If your API calls send representations that do not contain any links, you are most likely breaking the REST principle called Hypermedia as the Engine of Application State (HATEOAS)

5. Ignoring MIME types
   - If resources returned by API calls only have a single representation, you are probably only able to serve a limited number of clients that can understand the representation
   - You should use HTTP's content negotiation (supports various resource representations such as XML, JSON or YAML)

Useful Links

1. RESTful Web services: The basics, `http://www.ibm.com/developerworks/webservices/library/ws-restful/`
2. Create RESTful Web services with Java technology, `http://www.ibm.com/developerworks/library/wa-jaxrs/`
3. Building RESTful Web Services with JAX-RS, `http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html`
4. JAX-RS, `http://cxf.apache.org/docs/jax-rs.html`

Task

Assignment:

1. Download the worksheet from the course webpage
2. Complete the exercises in the worksheet
3. Upload it to the OLAT before the next session

Thank You