
MONASH University
MONASH
INFORMATION
TECHNOLOGY
FIT5192 Lecture 10: Web Services
GROUP
OF EIGHT
AUSTRALIA

MONASH University

Last Lecture

- Understand some of the finer aspects of Enterprise JavaBeans implementations in the Java EE platform.
- Tutorial Examples on EJBs

GROUP OF EIGHT AUSTRALIA

MONASH University

This Lecture

- Understand the role that web services can have for enterprise web applications and the web as a whole.
- Review the REST approach to web services
- Review JSON processing approaches with the Java EE 7 platform.

GROUP OF EIGHT AUSTRALIA

Web Services

Recap on Enterprise JavaBeans

- **A method of communication between two clients over the Internet using the HTTP/HTTPS protocol.**
- **Two main approaches to web services:**
  - Simple Object Access Protocol (SOAP)
  - Representational State Transfer (REST)
    - This is what we will be focusing on for this unit.
- **Message data**
  - Data is commonly transferred using text-based messages, often represented via XML or more lately, JSON.
  - Binary data is possible but less common.

Some Examples of Web Services

- **Business to Business (B2B) scenarios:**
  - Currency conversion
  - Inventory management
  - Product items and reviews
  - Real estate listings
  - …
- **Social Networking**
  - Facebook
  - Google+
  - Twitter
  - …

REST Web Services

- **Common approach to web services by using "stateless" methods of interaction, commonly via URLs.**
- **Uses HTTP methods such as GET and POST to receive and send information to a web service.**
- **Not really bound by a strict standard that SOAP requires.**
  - Very easy to setup as a result since you can make your own REST implementation.
  - Documenting the API required for almost any "Web 2.0" application since they all take different approaches.

REST Example

https://api.example.com/product/123

- **HTTP Methods:**
  - GET
    - Receive JSON format of the product with ID 123
  - PUT and POST
    - Replace or create a product with ID 123
  - DELETE
    - Remove the product with ID 123

- **GET and POST are very common implementations!**
  - Testable via the web browser!

OAuth

- **Authorisation framework commonly used with many web services to permit external applications to obtain limited information.**
  - Provides a secure method to transfer information between applications.
  - Web services implement either v1 or v2 draft.
- **Defacto authentication approach with Web 2.0 web services such as Twitter and YouTube.**
  - If you don't authenticate with the service, you don't get the data requested.
  - Web services now often require developers to get assigned a unique API key for OAuth to function.

Common steps with OAuth

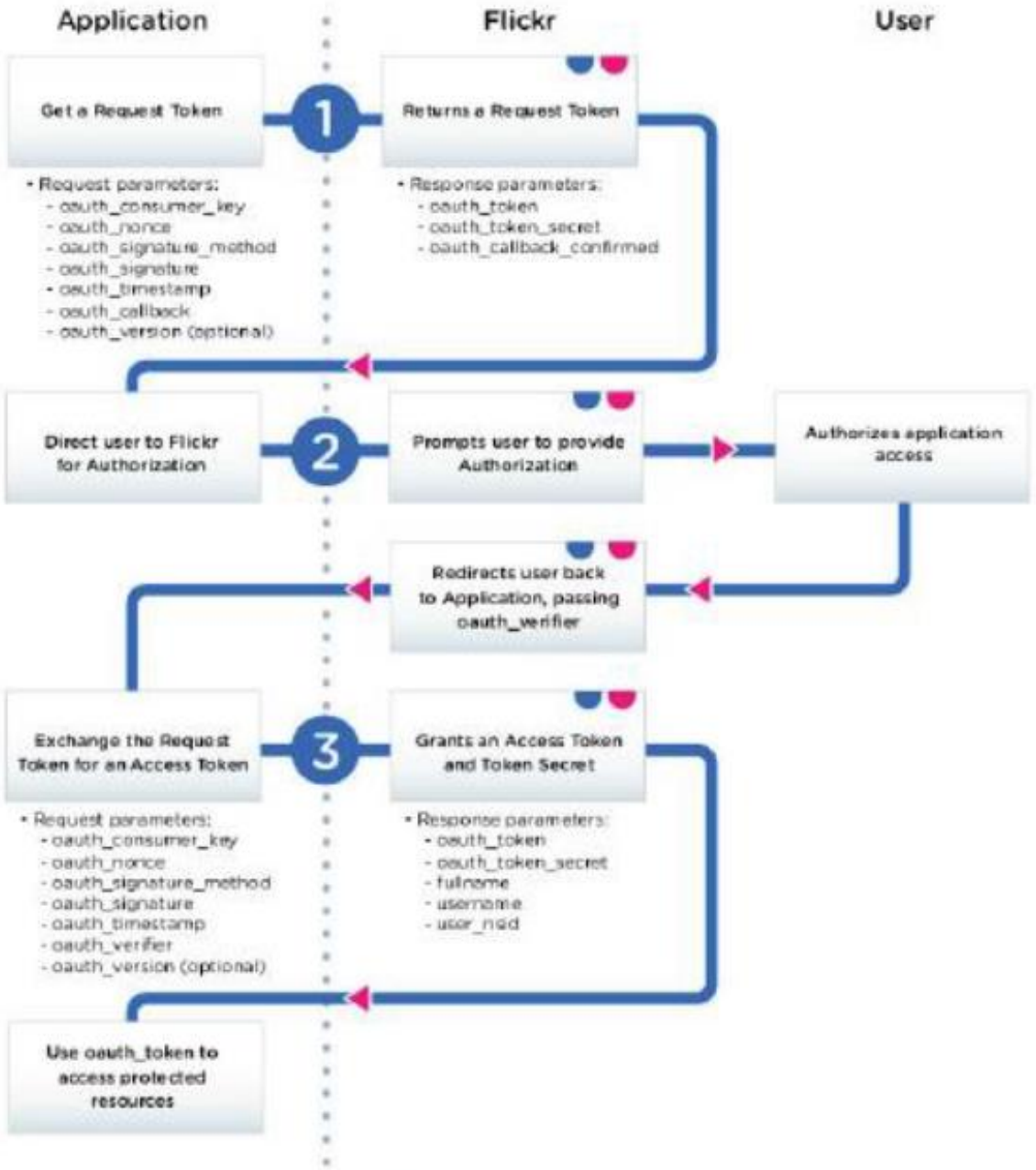
- **Communication between applications is often done in a three-step approach:**
  - **Get the request token**

    Ask the web service (using an assigned API key) for a temporary token asking for authentication.
  - **Get the user's authorisation**

    The user is then redirected to a page notifying them that the application wants access to their data. They often get a pin code if they accept.
  - **Exchange request token for access token**

    Access token used to authenticate requests

Oau



Source: http://www.flickr.com/services/api/auth.oauth.html

So why REST?

- **Significantly less development time required to deploy and consume a service:**
  – SOAP often requires extensive middleware to work correctly and can be a pain to use for mobile applications.
- **Clients can receive information from a REST web service often by simply making a normal HTTP request to a URL.**
  – We can consume information without needing libraries for simple actions.
- **Encryption is easy by just serving the web service via HTTPS**

Web Services for Enterprise Web Apps

- So why should developers such as yourself concern yourself with web services?
- **Easy to transfer information between distributed systems on the Internet**
  – We no longer have to worry about what language or platform the system is running if we use a common medium for transferring messages.
- **Easy to transfer dynamic information back to web clients such as browsers (using AJAX)**
- **Easy to work with external systems outside of the Enterprise control (E.g. Facebook, Twitter)**

REST examples

Creating a RESTful Root Resource Class

- **From JEE 7 Tutorial Chapter 29**
- **Root resource classes** are "plain old Java objects" (POJOs)
  – annotated with @Path or have at least one method annotated with @Path
  – or a **request method designator**, such as @GET, @PUT, @POST, or @DELETE.
  – **Resource methods** are methods of a resource class annotated with a request method designator

Helloworld RESTful service (1)

```
package javaeetutorial.hello;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
/**
* Root resource (exposed at "helloworld" path)
*/
……..
```

Helloworld RESTful service (2)

```
……
@Path("helloworld")
public class HelloWorld {
@Context
private UriInfo context;
/** Creates a new instance of HelloWorld */
public HelloWorld() { }
/**
* Retrieves representation of an instance of
helloWorld.HelloWorld
* @return an instance of java.lang.String */
……
```

Helloworld RESTful service (3)

```
@GET
@Produces("text/html")
    public String getHtml() {
        return "<html
    lang=\"en\"><body><h1>Hello,World!!</h1></body></html>"
    ;
    }
}
```

Helloworld RESTful service (Put Method)

- We Can add a put method:

```
@PUT
@Consumes("text/html")
   public void putHtml(String content) {
      //// Store a resource on the server
}
```

The @Path Annotation

- @Path annotation
  – identifies the URI path template to which the resource responds
  – is specified at the class or method level of a resource
  – relative to the base URI of the server on which the resource is deployed
    - the context root of the application
    - URL pattern to which the JAX-RS runtime responds

URI Path Templates (1)

- URIs with variables embedded within the URI syntax
  – Substituted at runtime
  – Variables are denoted by braces ({ and })
- E.g.: @Path("/users/{username}")
  – User is prompted to type his or her name
  – Configured JAX-RS web service respond to requests
  – For example, if user name "Galileo," the web service responds to the following URL:
    - http://example.com/users/Galileo

URI Path Templates (2)

- @PathParam annotation used to access variable value
- E.g.:

```
@Path("/users/{username}")
public class UserResource {
   @GET
   @Produces("text/xml")
   public String getUser(@PathParam("username")
   String userName) {
   ...
   }
}
```

Query Parameters

- @QueryParam used to extract query parameters from the Query component of the request URL

E.g.

```
@Path("smooth")
@GET
public Response smooth(
    @DefaultValue("2") @QueryParam("step") int step,
    @DefaultValue("true") @QueryParam("min-m") boolean hasMin,
    @DefaultValue("true") @QueryParam("max-m") boolean hasMax,
    @DefaultValue("true") @QueryParam("last-m") boolean hasLast,
    ) {
    ...
    }
```

Form Parameters

- Useful for extracting information sent by POST in HTML forms.

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void post(@FormParam("name") String name) {
    // Store the message
}
```

Overview of the RESTful Client API

- **Creating a Basic Client Request Using the Client API**
  - Obtain an instance of the javax.ws.rs.client.Client interface.
  - Configure the Client instance with a target.
  - Create a request based on the target.
  - Invoke the request.

Example RESTful Client API (GET)

- Method invocations can be chained together
  – to configure and submit a request to a REST resource

```
Client client = ClientBuilder.newClient();
String name = client.target("http://example.com/webapi/hello")
    .request(MediaType.TEXT_PLAIN)
    .get(String.class);
```

Example RESTful Client API (POST)

- Method invocations can be chained together
  – to configure and submit a request to a REST resource

```
Client client = ClientBuilder.newClient();
Form form = new Form();
form.param("name", name);

String name = client.target("http://example.com/webapi/hello")
.request(MediaType.APPLICATION_FORM_URLENCODED_TYPE)
.post(Entity.entity(form,
MediaType.APPLICATION_FORM_URLENCODED_TYPE));
```

JSON processing approaches

- Using JSON is similar, we just need to change the media type to JSON

```
@GET
@Produces(MediaType.APPLICATION_JSON)
```

Summary

- Understand the role that web services can have for enterprise web applications and the web as a whole.
- Review the REST approach to web services
- Review JSON processing approaches with the Java EE 7 platform.

Next Week

- Java Enterprise Edition Security

MONASH University

See you in the Studio !

Readings

MONASH University

- **Part VI *Web Services* in The Java EE 7 Tutorial**
  Chapter 27 to 31 cover the various web services technologies