Cxf Substitution User Manual

version: 1.0

Table of Contents

About Substitution

Substitution is a project raised by Mr. GRALL in the Nantes Engineer School to solve the problem of the contravention of the Liskov substitution principle. The project is carried out by Kevin Llopart, Raphael Martignoni, Gregoire Seguin-Henry and Hao Zhang.

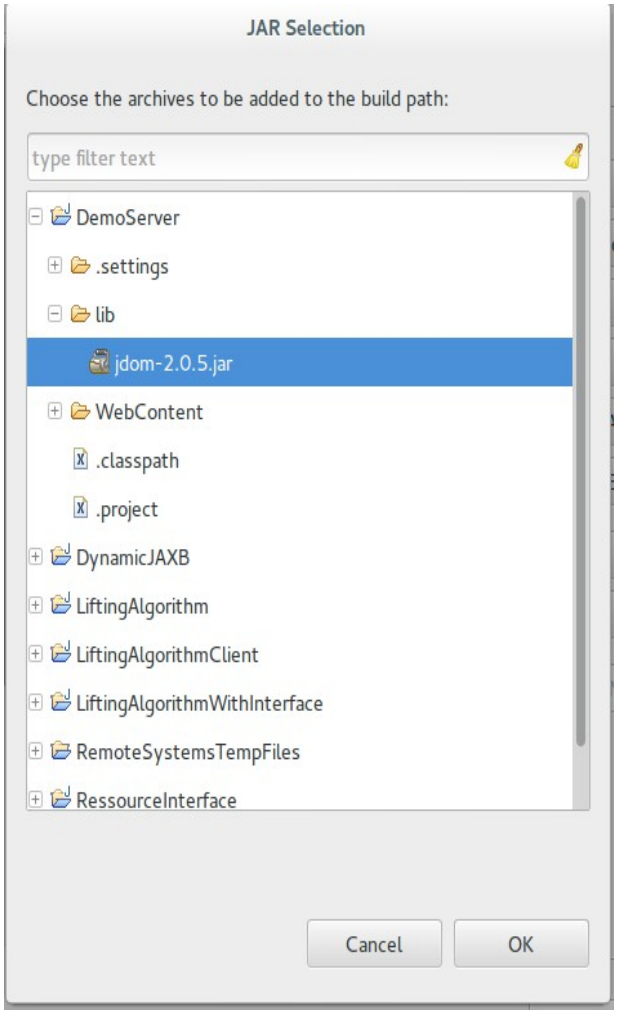
The purpose

The Substitution project is raised to solve the problem of the contravention of the Liskov substitution principle in the framework cxf.
The framework cxf is contrary to the Liskov substitution principle in two aspects:

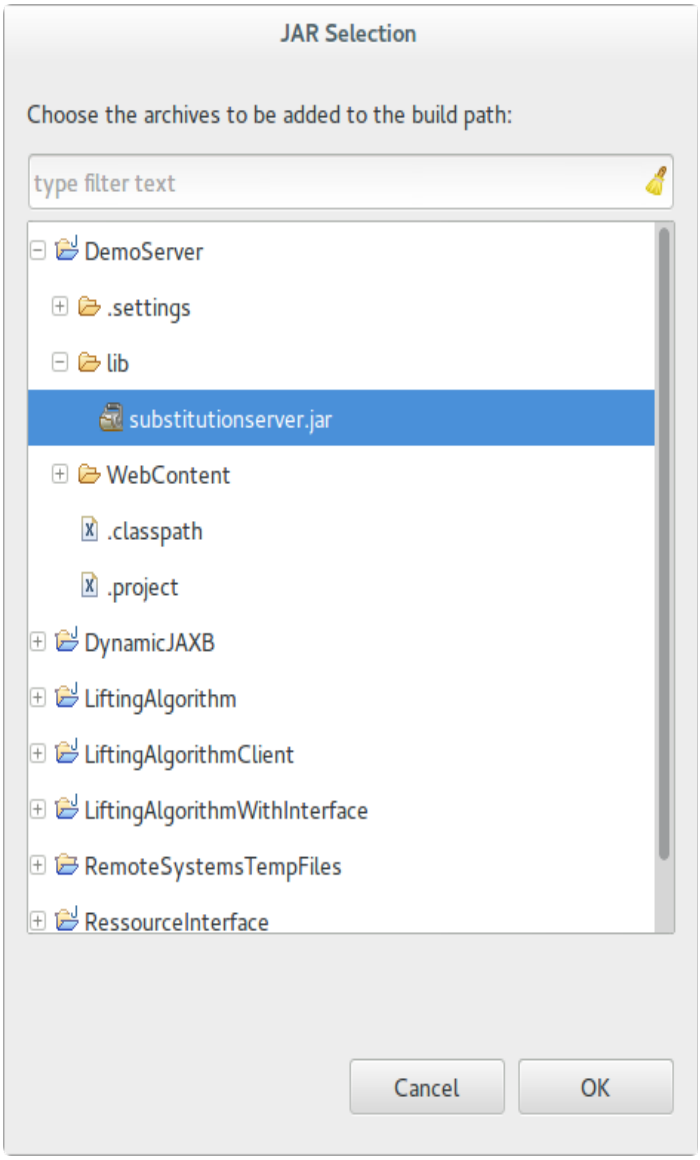
- The coupling between the service layer and the object layer.
- The problem of the inter-operation of inheritance between the client and the server.

Enable the substitution on the server

1. Download and import the **jdom2** module into your server project.
   You can download the **jdom2** module from the website
   http://www.jdom.org/downloads/index.html



2. import the **substitutionserver.jar** into your server project and add it into the compiling path.
   For example, in eclipse:



3. Edit your server configure file to add a new **filter bean**, in eclipse, the path is **WebContent/WEB-INF**.

   Add a filter bean definition:

```
<bean id="ClientRequestFilter" class="filter.server.ClientRequestFilter">
```

   Add a filter bean as a provider:

```
<jaxrs:providers>
      <ref bean="ClientRequestFilter" />
</jaxrs:providers>
```

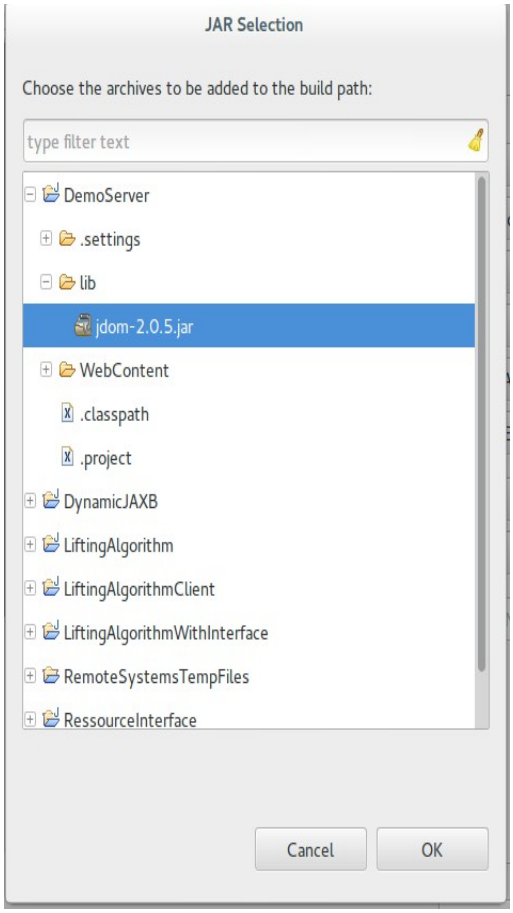
4. Add the annotation `@AllowSubstitution` to your declaration of the service methods, for example:

```
    @AllowSubstitution
    @POST
    @Path("/name")
    @Produces(MediaType.APPLICATION_XML)
    public String name(Person p);
```

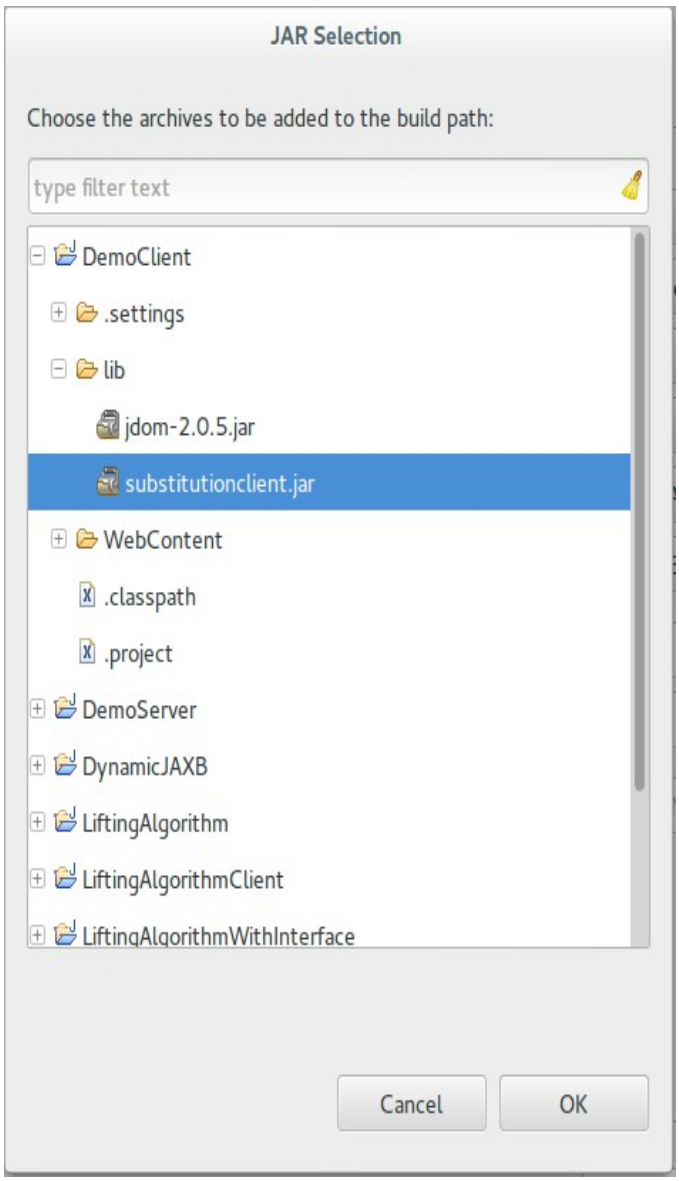
Now, the substitution method is enabled on the server side.

Enable the substitution on the client

1. Download and import the **jdom2** module into your client project.
   You can download the **jdom2** module from the website
   http://www.jdom.org/downloads/index.html



2. import the **substitutionclient.jar** into your client project and add it into the compiling path.
   For example, in eclipse:



3. When you create the server point with the client factory, add a list of the interceptors as following:

```
List<Object> filtres = new LinkedList<>();
filtres.add(new LiftingInterceptor());
Service service = JAXRSClientFactory.create(
                        "http://localhost:8080/DemoServer",
                        Service.class, filtres);
```

Now, the substitution method is enabled on the client side.

Enable the function interface as resource

This is used on the server.
For example, we want to define a resource Person.

1. Define a interface Person as following:

```
public interface Person {

	@GET
	@Path("id")
	public int getId();

	@PUT
	@Path("id")
	public void setId(int value);

	@GET
	@Path("lastname")
	public String getLastname();

	@PUT
	@Path("lastname")
	public void setLastname(String value);

	@GET
	@Path("firstname")
	public String getFirstname();

	@PUT
	@Path("firstname")
	public void setFirstname(String value);

	@GET
	@Path("")
	@Produces(MediaType.APPLICATION_XML)
	public Person getRepresentation();

}
```

2. Define a class PersonImpl to implement the interface Person as following:

```
@XmlRootElement(name="personimpl")
public class PersonImpl implements Person {
	protected int id;
	protected String lastname;
	protected String firstname;

	@Override
	public int getId(){
		return id;
	}

	@Override
	public void setId(int value){
		id = value;
	}
```

```
	@Override
	public String getLastname() {
		return lastname;
	}

	@Override
	public void setLastname(String value) {
		this.lastname = value;
	}

	@Override
	public String getFirstname() {
		return firstname;
	}

	@Override
	public void setFirstname(String value) {
		this.firstname = value;
	}

	@Override
	public Person getRepresentation() {
		return this;
	}

}
```

3. Define a class PersonAdapter as following:

```
public class PersonAdapter extends XmlAdapter<PersonImpl, Person>{

	@Override
	public PersonImpl marshal(Person arg0) throws Exception {
		return (PersonImpl)arg0;
	}

	@Override
	public Person unmarshal(PersonImpl arg0) throws Exception {
		return arg0;
	}

}
```

4. Create a **package-info.java** file in your resource model directory, if it doesn't exist. The add the following into the file.

```
@XmlJavaTypeAdapters({
	@XmlJavaTypeAdapter(PersonAdapter.class)
})
```

5. Add the annotation `@XmlJavaTypeAdapter(PersonAdapter.class)` in front of the parameter or the return type in the declaration of the server method, as following:

```
public String getName(@XmlJavaTypeAdapter(PersonAdapter.class)Person p);
```

This tells JAXB to use **PersonImpl** instead of **Person** when marshalling or unmarshalling.

History

V1.0
author: Hao ZHANG
date: February 17th 2014
changes:
The first version, adding demonstration to demonstrate how to use substitution in the server and/or the client and how to use the interface as resource.