Departamento de Lenguajes y Sistemas Informáticos

# Desarrollo de Aplicaciones con GWT (II)

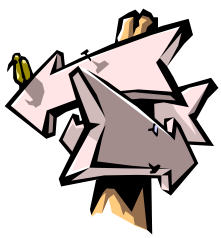
**Práctica 3**


Arquitectura e Integración del Software

Curso 2012/2013

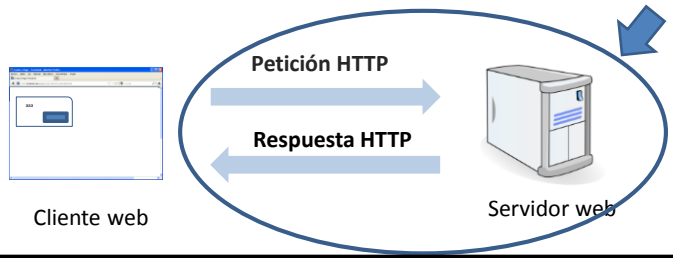
## Índice

- **Introducción**
- Comunicación Cliente-Servidor
- Enlaces



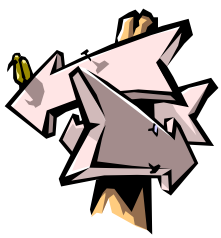
## Introducción

- En el paquete *client* hemos visto como desarrollar la interfaz de usuario, que se corresponde con la capa de presentación en la arquitectura en tres capas.
- GWT permite que la capa de negocio o la capa de acceso a recursos también se ejecute en cliente, pero normalmente la capa de acceso a recursos y negocio se despliega en servidor.
- **En la segunda parte de esta práctica, aprenderemos a comunicar nuestro cliente con el servidor mediante GWT.**




## Índice



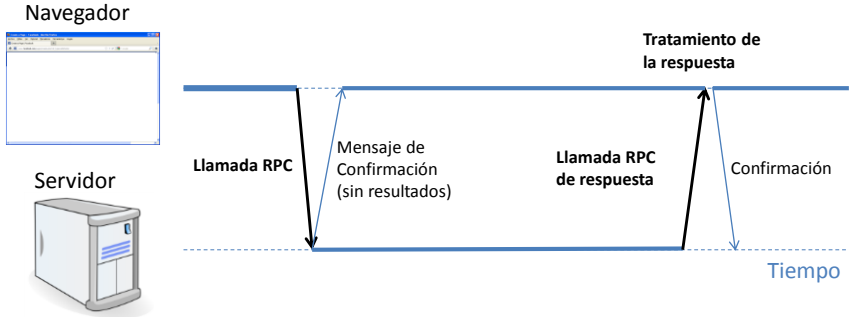


## Comunicación Cliente-Servidor
### Servlets

- En servidor, GWT utiliza tecnología Java EE.
- Java EE, además de las librerías standard de Java, provee de una nueva clase de objetos llamados servlets.
- Estos objetos proveen de funcionalidad para responder a peticiones HTTP.
- La comunicación entre Cliente y Servidor en GWT provee de invocación a estos objetos a través de un tipo de peticiones llamado GWT-RPC.

## Comunicación Cliente-Servidor
### RPC

- Las llamadas RPC son de tipo asíncrono, es decir, cuando se hace una solicitud desde el cliente al servidor, el navegador sigue su ejecución hasta que el servidor manda la respuesta a la solicitud.
- Se utiliza este tipo de peticiones para que el navegador no quede bloqueado esperando la respuesta.
- A la vez que se hace la petición de Cliente-Servidor se indica el método al que llamar para la respuesta (Servidor-Cliente).



## Comunicación Cliente-Servidor
### Uso de GWT-RPC (I)

- Primero, definimos en el paquete *client* un interfaz con los métodos que vamos a necesitar implementar en servidor.
- Este interfaz debe extender al interfaz RemoteService.

```
package es.us.eii.client;

import java.util.List;


public interface ContactService extends RemoteService {

    public void             addContact(Contact c);
    public void             updateContact(Integer i, Contact c);
    public List<Contact>    getContacts();
    public void             deleteContact(Integer i);


}
```

- Si necesitamos muchos métodos de servidor en el cliente, es conveniente separarlos en interfaces diferentes (por ejemplo, en función del tipo de información que manejan)

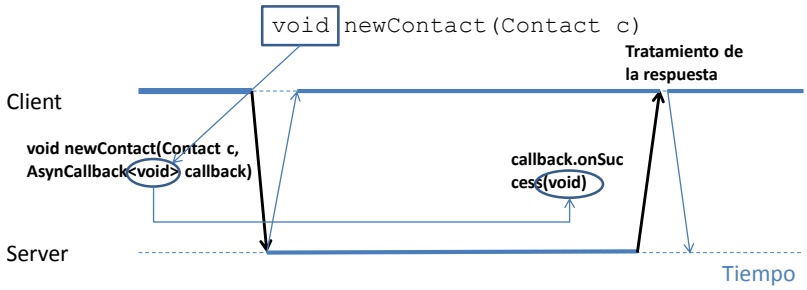
## Comunicación Cliente-Servidor
### Uso de GWT-RPC (II)

- En segundo lugar, creamos una clase en el paquete *server* que implemente la interfaz creada y extienda a *RemoteServiceServlet*.
- Habrá que implementar cada método del interfaz.
- Desde esta clase podremos importar otras clases que tengamos en el paquete server (que no sean servlets).

```
public class ContactServiceImpl extends RemoteServiceServlet implements
        ContactService {
    @Override
    public void addContact(Contact c) {
        ...
    }
    @Override
    public void updateContact(Integer i, Contact c) {
    }
    @Override
    public List<Contact> getContacts() {
        ...
    }
    @Override
    public void deleteContact(Integer i) {
        ...
    }
}
```

## Comunicación Cliente-Servidor
### Uso de GWT-RPC (III)

- En nuestras clases de *client*, no podemos usar directamente los métodos del paquete *server*, sino que usaremos un nuevo interfaz en el que por cada método del interfaz creado habrá un método con un parámetro extra de la clase *AsyncCallback*.
- La clase *AsyncCallback* se utiliza para las llamadas "de vuelta", es decir, cuando invoquemos a un método desde cliente a servidor, el servidor invocará a este método en cliente.




## Comunicación Cliente-Servidor
### Uso de GWT-RPC (IV)

- El plugin de GWT ayuda a generar automáticamente, a partir de la interfaz que extiende a *RemoteService*, la interfaz con el parámetro *AsyncCallback*.

```
package es.us.eii.client;

import java.util.List;
import es.us.eii.shared.Contact;
import com.google.gwt.user.client.rpc.RemoteService;


public interface ContactService extends RemoteService {

    public void             addContact(Contact c);
    public void             updateContact(Integer i, Contact c);
    public List<Contact>    getContacts();
    public void             deleteContact(Integer i);

}
```

```
package es.us.eii.client;

import java.util.List;

import com.google.gwt.user.client.rpc.AsyncCallback;

import es.us.eii.shared.Contact;

public interface ContactServiceAsync {

    void addContact(Contact c, AsyncCallback<Void> callback);
    void deleteContact(Integer i, AsyncCallback<Void> callback);
    void getContacts(AsyncCallback<List<Contact>> callback);
    void updateContact(Integer i, Contact c, AsyncCallback<Void> callback);

}
```

## Comunicación Cliente-Servidor
### Uso de GWT-RPC (V)

- Para llamar a los métodos implementados en *server* en nuestras clases del paquete *client* tenemos que usar un objeto del tipo de la interfaz con el parámetro AsyncCallback.
- Para crearlo, usamos el método *GWT.create(nombredelainterfaz)*

```
private final ContactServiceAsync contactService = GWT
        .create(ContactService.class);
```

- Cuando queremos llamar a un método de este objeto, tenemos que pasar como parámetro un objeto de tipo AsyncCallback. Estos objetos se pueden crear en la propia llamada, pero tienen que implementar 2 métodos:
  - **onSuccess**, que recibe como parámetro de entrada lo que devuelve la implementación en server. Este método es el que llama el servidor cuando termina de ejecutar el método que invocó el cliente previamente.
  - **onFailure**, que recibe como parámetro de entrada una excepción, cuando ha habido algún fallo en la ejecución del método en servidor.

## Comunicación Cliente-Servidor
### Uso de GWT-RPC (VI)

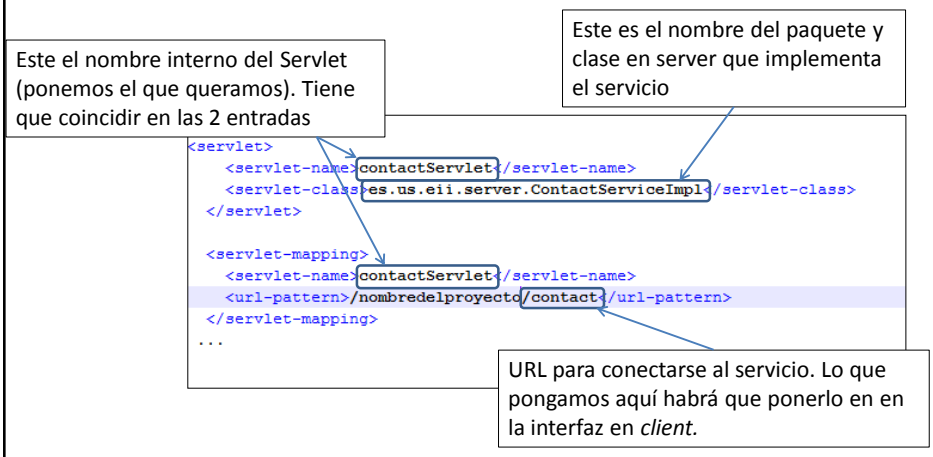
```
contactService.getContacts(new AsyncCallback<List<Contact>>(){
        public void onSuccess(List<Contact> a){
            //Aquí pondremos que hacemos en cliente cuando recibimos la respuesta del servidor
        }

        @Override
        public void onFailure(Throwable caught) {
            // TODO Auto-generated method stub

        }


    });
```

- NOTA: Si queremos usar objetos creados fuera de los métodos *onSuccess* y *onFailure*, deben ser de tipo **final**.

## Comunicación Cliente-Servidor
### Uso de GWT-RPC (VII)

- La implementación de la interfaz en el servidor debe ser accesible mediante HTTP a través de una URL para poder invocarla desde el cliente.
- Para especificar la URL a la que hay que conectarse para ejecutar la implementación de *server*, tenemos que hacerlo en el fichero Carpeta war->WEB-INFO->web.xml
- En dicho fichero, tendremos que meter una entrada *<servlet>* y otra *<servlet-mapping>* por cada implementación de servicio

## Comunicación Cliente-Servidor
### Uso de GWT-RPC (VII)



## Comunicación Cliente-Servidor
### Uso de GWT-RPC (VIII)

- La URL indicada en el fichero web.xml hay que indicársela a nuestra interfaz en *client* para que las clases de cliente sepan la URL que implementa el servicio.

```
...
servlet>
   <servlet-name>contactServlet</servlet-name>
   <servlet-class>es.us.eii.server.ContactServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>contactServlet</servlet-name>
   <url-pattern>/nombredelproyecto/contact</url-pattern>
</servlet-mapping>
...
```

```
package es.us.eii.client;

import java.util.List;
import es.us.eii.shared.Contact;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;


@RemoteServiceRelativePath("contact")
public interface ContactService extends RemoteService {

    public void                 addContact(Contact c);
                                updateContact(Integer i, Contact c);
                            >   getContacts();
                                deleteContact(Integer i);
```

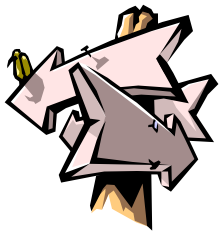
Habrá que añadir esta línea a nuestra interfaz en *client*. El valor concreto de la URL depende de lo que hayamos puesto en el fichero web.xml

## Comunicación Cliente-Servidor
### Paso de objetos entre cliente y servidor con GWT-RPC

- Las clases que vayamos a utilizar tanto en cliente como en servidor hay que definirlas en el paquete *shared*.
- Si además, vamos a usar objetos de clases propisa como parámetro de entrada y salida de nuestros servicios (es decir, vamos a usar estos objetos en clases *client* y *server*), tenemos que asegurarnos que ese objeto es serializable (para que su conversión desde Javascript a http y después a Java y viceversa, sea posible).
- Los tipos primitivos (String, int, float, …) son serializables. Los arrays y colecciones de tipos serializables. Nuestras propias clases tendrán que implementar la interfaz Serializable y tener un constructor vacío para asegurar que los objetos son serializables.

## Índice

- Introducción
- Comunicación Cliente-Servidor
- **Enlaces**



## Enlaces

- **Tutoriales (MUY RECOMENDADOS)**

**Tutorial de uso de GWT-RPC con el proyecto GWT. Stockwatcher**
https://developers.google.com/web-toolkit/doc/latest/tutorial/RPC

**Tutorial visual de GWT-RPC**
http://www.youtube.com/watch?v=jMTF2ZzAjHM

## Disclaimer and Terms of Use

All material displayed on this presentation is for teaching and personal use only.

Many of the images that have been used in the presentation are Royalty Free images taken from http://www.everystockphoto.com/. Other images have been sourced directly from the Public domain, from where in most cases it is unclear whether copyright has been explicitly claimed. Our intention is not to infringe any artist's copyright, whether written or visual. We do not claim ownership of any image that has been freely obtained from the public domain. In the event that we have freely obtained an image or quotation that has been placed in the public domain and in doing so have inadvertently used a copyrighted image without the copyright holder's express permission we ask that the copyright holder writes to us directly, upon which we will contact the copyright holder to request full written permission to use the quote or images.