Tema 1 (Parte 6)
Servidor dinámico Java

J. Gutiérrez

Departament d'Informàtica
Universitat de València

DAW-TS (ISAW).
Curso 14-15

Índice

Servidor dinámico Java

En la parte anterior se ha mostrado que es posible ejecutar código dinámico en el servidor lanzando un proceso independiente que es el que genera la respuesta.

Esto tiene varias desventajas:

- Lanzar un proceso es lento (comparado con ejecutar código en un hilo).
- Se puede comprometer la seguridad
- Es complicado ofrecer servicios al código dinámico (por ejemplo acceso a bases de datos, transacciones, etc).

Servidor dinámico Java

Por todo ello, el siguiente paso será poder ejecutar código dinámico Java dentro del proceso en el que se ejecuta el servidor.

Este código dinámico se ejecutará en un hilo para que se pueda atender a diferentes clientes de forma simultánea.

El principal problema es que hay que ejecutar código que no existe en el momento de la compilación del servidor.

Servidor dinámico Java

Es decir, quien desarrolle el código dinámico debe subir el código al servidor y el servidor debe ejecutarlo (crear una instancia y ejecutar un método en un hilo).

¿Cómo se puede crear una instancia de una clase si no está disponible en el servidor cuando se compila?

Esto se puede hacer usando el API Reflection.

Objetivo

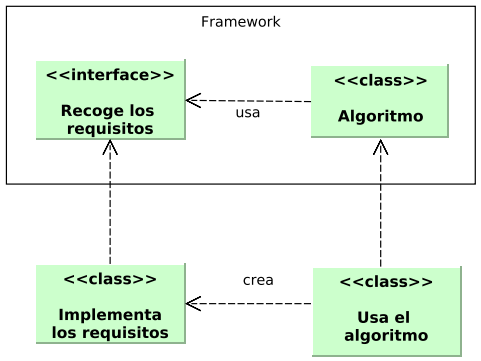
Desarrollar un framework tal que:

- El servidor pueda ejecutar código desarrollados por otros. Tendremos que ofrecer un mecanismo para que el servidor sepa qué fichero jar contiene el código y cual es el nombre de la clase que tiene el código a ejecutar.
- Facilitar el desarrollo de código dinámico (que pueda ser ejecutado por el servidor).

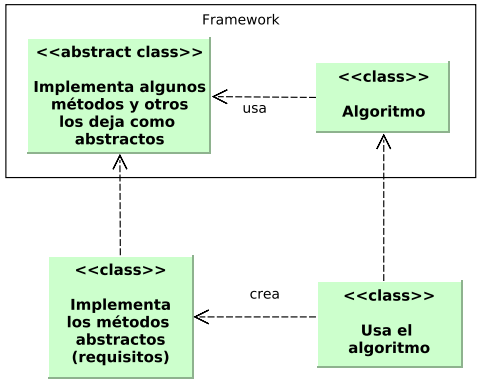
¿Qué es un framework?

Un framework lo constituyen una serie de tipos (clases o interfaces) que implementa la parte que es general para un determinado tipo de aplicación. Quien quiera usar el framework deberá ímplementar (centrarse en) la parte de código que es dependiente de su problema.

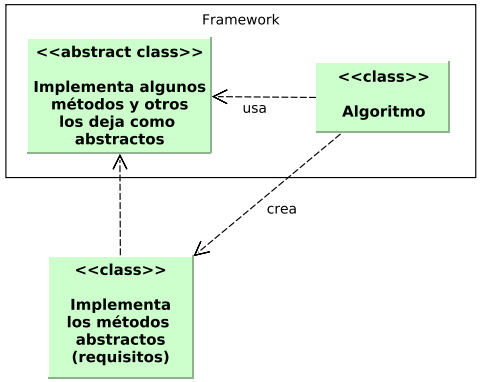
Ejemplo de framework y código que lo usa



Ejemplo de framework y código que lo usa



Ejemplo de framework y código que lo usa



Índice

Objetivo

Desarrollar el framework de tal modo que se facilite el desarrollo del código dinámico y que gestione la creación del objeto y la llamada a sus métodos (es decir, que gestione el ciclo de vida).

Algo así como:

```
class CodigoDinamico extends ClaseQueProporcionaElFramework{
   public void ifGet(ClaseDelFrameworkConElementosDeLaPeticion requestThings,
                     ClaseDelFrameworkConElementosDeLaRespuesta responseThings){
      // ¿Qué elementos debería poder obtener de requestThings?

      // ¿Qué elementos debería poder obtener de responseThings?

      // Código para escribir la respuesta
   }
   public void ifPost((ClaseDelFrameworkConElementosDeLaPeticion requestThings,
                     ClaseDelFrameworkConElementosDeLaRespuesta responseThings){
   ...
   }
   ...
}
```

Además, el servidor gestionará el ciclo de vida de este objeto y llamará a los métodos de esta clase. Por tanto será responsable pasar los objetos que necesiten.

Fichero con los mappings

Supongamos que compilamos esta clase y la almacenamos en un fichero jar (por ejemplo `aplicacion.jar`). Entonces el fichero con los mappings podría contener la siguiente línea:

```
aplicacion;/opt/web/dynamic/aplicacion.jar CodigoDinamico
```


El primer elemento es la URL, el segundo elemento contiene a su vez dos elementos: la localización del fichero jar (con el código compilado) y el nombre de la clase que extiende a la clase que proporciona el framework.

Supongamos que el servidor lee este fichero y obtiene el último de los elementos (el nombre de la clase). Esto es de tipo `String`. ¿Cómo podemos crear un objeto de este tipo si lo que tenemos es su nombre?.

Para esto se puede usar el API *reflection*.

Clase que ofrece el framework para almacenar elementos de la petición

```
public class ThingsAboutRequest {
   private HashMap<String, String> params;
   private HashMap<String, String> headers;
   private InputStream in;

   public ThingsAboutRequest(HashMap<String, String> h,
         HashMap<String, String> p, InputStream in) {
      params = p;
      headers = h;
      this.in = in;
   }
   public String getParam(String c) {
      return params.get(c);
   }
   public String getHeader(String c) {
      return headers.get(c);
   }
   public Set<String> getParamNames() {
      return params.keySet();
   }
   public Set<String> getPresentHeaders() {
      return headers.keySet();
   }
   public InputStream getInputStream() {
      return in;
   }
}
```


Clase que ofrece el framework para almacenar elementos de la petición

```
public class ThingsAboutResponse {
   private OutputStream out;
   private PrintWriter pw;
   private HashMap<String, String> headers;

   public ThingsAboutResponse(OutputStream o) {
      out = o;
      pw = new PrintWriter(out);
      headers = new HashMap<String, String>();
   }

   public OutputStream getOutputStream() {
      return out;
   }
   public void flushResponseHeaders() {
      for (String h : headers.keySet()) {
         pw.print(h);
         pw.print(": ");
         pw.println(headers.get(h));
      }
      pw.println("");
      pw.flush();
   }
   public void setResponseHeader(String h, String v) {
      headers.put(h, v);
   }
}
```

Clase del framework que debe extender quien desarrolle código dinámico I

```
public class ResponseClass {
   private Socket canal;
   private String method;
   private String resource;

   void setSocket(Socket s) {
      canal = s;
   }

   void setMethod(String m) {
      method = m;
   }

   void setResource(String r) {
      resource = r;
   }

   public void ifGet(ThingsAboutRequest req, ThingsAboutResponse resp)
         throws Exception {
   }

   public void ifPost(ThingsAboutRequest req, ThingsAboutResponse resp)
         throws Exception {
   }

   public void dealWithCall() throws Exception {

      InputStream in = canal.getInputStream();
      OutputStream out = canal.getOutputStream();
```


Clase del framework que debe extender quien desarrolle código dinámico II

```
      HashMap<String, String> headers = UtilsHTTP.getHeaders(in);
      HashMap<String, String> params;

      if (method.equals("GET"))
         params = UtilsHTTP.getParamsGet(resource);
      else
         params = UtilsHTTP.parseBody(UtilsHTTP.getBody(headers, in));

      ThingsAboutRequest req = new ThingsAboutRequest(headers, params, in);
      ThingsAboutResponse resp = new ThingsAboutResponse(out);

      PrintWriter pw = new PrintWriter(out);
      UtilsHTTP.writeResponseLineOK(pw);

      if (method != null) {
         if (method.equals("GET")) {
            ifGet(req, resp);
         } else if (method.equals("POST")) {
            ifPost(req, resp);
         }
      }

   }
}
```


Clase del framework que usa a la clase anterior I

```
public class ThreadDynamic implements Runnable {
   private Socket canal;
   private String clase;
   private String request;

   public ThreadDynamic(Socket s, String cl, String req) {
      canal = s;
      clase = cl;
      request = req;
   }

   public void run() {
      try {
         Class<?> c = Class.forName(clase);
         Constructor<?> con = c.getConstructor(new Class<?>[] {});
         ResponseClass rc = (ResponseClass) con.newInstance(new Object[] {});
         rc.setMethod(UtilsHTTP.getMethod(request));
         rc.setResource(UtilsHTTP.getResource(request));
         rc.setSocket(canal);
         rc.dealWithCall();
         canal.close();
         rc = null;
      } catch (Exception ex) {
         ex.printStackTrace();
         try {
            UtilsHTTP.writeResponseServerError(new PrintWriter(canal
```


Clase del framework que usa a la clase anterior II

```
                  .getOutputStream()));
         } catch (Exception ex2) {
         }
      }
   }
}
```

Añadir ficheros jar desde el código

En el `main` se procesa el fichero con los *mappings* dinámicos y se obtienen los ficheros `jar` donde están las clases que extienden a la clase `ResponseClass`.

Estos ficheros `jar` hay que añadirlos al `classpath` desde el código Java.

Es decir, no podemos hacerlo de esta forma:

```
java -classpath fich1.jar:fich2.jar:...:fichN.jar;. Servidor
```

Para ello se puede utilizar la clase `JarPathUtils` que tiene métodos para añadir al `classpath` un fichero `jar`.

Índice

El código cliente consiste en una clase que extiende a la clase `ResponseClass`, proporcionada por el *framework*.

En esta clase se implementan los métodos `ifGet` o `ifPost` o ambos.

Ejemplo de código cliente

```
//import ...

public class Response1 extends ResponseClass {
   public void ifGet(ThingsAboutRequest req, ThingsAboutResponse resp)
         throws Exception {
      OutputStream out = resp.getOutputStream();
      PrintWriter pw = new PrintWriter(out);

      resp.setResponseHeader("Content-Type", "text/html; charset=utf-8");
      resp.flushResponseHeaders();

      pw.println("<html>");
      pw.println("<body>");
      pw.println("<h1> Respuesta </h1>");

      Set<String> hdrs = req.getPresentHeaders();

      pw.println("<h2> Campos de cabecera de la peticion </h2>");

      for (String s : hdrs)
         pw.println(s + ": " + req.getHeader(s) + "</br>");

      pw.println("</body>");
      pw.println("</html>");
      pw.flush();
      pw.close();

   }
}
```

Fichero de mappings

Una vez realizado este código de debe generar un fichero jar, y se debe añadir una línea al fichero de los mappings indicando la URL, el fichero jar y el nombre de la clase que extiende a `ResponseClass`.

Índice

Mejoras

Implementar el patrón singleton

En el código proporcionado se crea un objeto por cada petición. Si hay muchas peticiones estamos generando muchos objetos que provocarán que se lance el recolector de basura en el servidor y esto provocará que las peticiones que lleguen en ese momento tarden más en ser atendidas. En lugar de crear un objeto por cada petición podríamos tener un objeto por cada mapping y reutilizarlo. **Esto se deja como tarea**.

Mejoras

Usar anotaciones o incluir un fichero XML

En lugar de añadir líneas al fichero dynamic_mappings.txt se podría pensar en que el propio fichero jar generado llevara esta información. Por ejemplo se podría incluir un fichero en formato XML con esa información. O mejor todavía, se podrían definir anotaciones que permitieran definir el *mapping* en el propio código fuente.

Acceso a servicios gestionados por el servidor

Se podría proporcionar un objeto común a todas las aplicaciones dinámicas donde colocar servicios que gestiona el servidor. Esos servicios podrían ser: conexiones a bases de datos, conexiones a colas de mensajes, etc. De este modo desde el código cliente obtendríamos una referencia a ese objeto y podríamos solicitar estos servicios.

Despliegue de aplicaciones en servidor remoto

Realizar una aplicación cliente/servidor que permitiera desplegar una aplicación en un servidor remoto.

Código estático

Que el fichero jar además de contener código dinámico también pudiera contener contenido estático (páginas HTML, código JavaScript, etc).