🏁 00:39

Applicazioni Web

Abbiamo visto il protocollo HTTP ed abbiamo anche implementato alcuni server web. I web server non producono risorse solo attraverso il recupero di contenuti presenti all'interno di file memorizzate in locale, ma le risorse possono essere anche **generate dinamicamente** a seguito di una richiesta proveniente dal browser.

Un web server oltre al comportamento convenzionale che prevede di interpretare la richiesta per restituire una risorsa presente in locale, può avere anche altri comportamenti. Potrebbero infatti attivare l'esecuzione di uno script lato server che elabora l'input lato server per produrre dinamicamente un output.

Possiamo pensare al web server come una componente remota che può elaborare dei dati in input forniti dal client, e restituire una risposta (che può essere di diversi tipi).

Per far si che il server possa produrre degli output specifici possiamo usare due classi di tecnologie:

Server Script

Intendiamo degli script (anche veri e propri programmi) che vengono mandati in esecuzione quando viene ricevuta una determinata richiesta dal client, fornendo loro un input fornito sempre dal client. In java usiamo **Servlet**

Server side include

In questo caso la produzine dinamica dei contenuti è affidata a delle pagine HTML (apparentemente statiche) che però prevedono al loro interno dei frammenti di codice che saranno mandati in esecuzione sul server per produrre varianti di quelle pagine HTML in modo dinamico.

Queste pagine, quindi, preservano un **template di base** che però può cambiare dinamicamente.

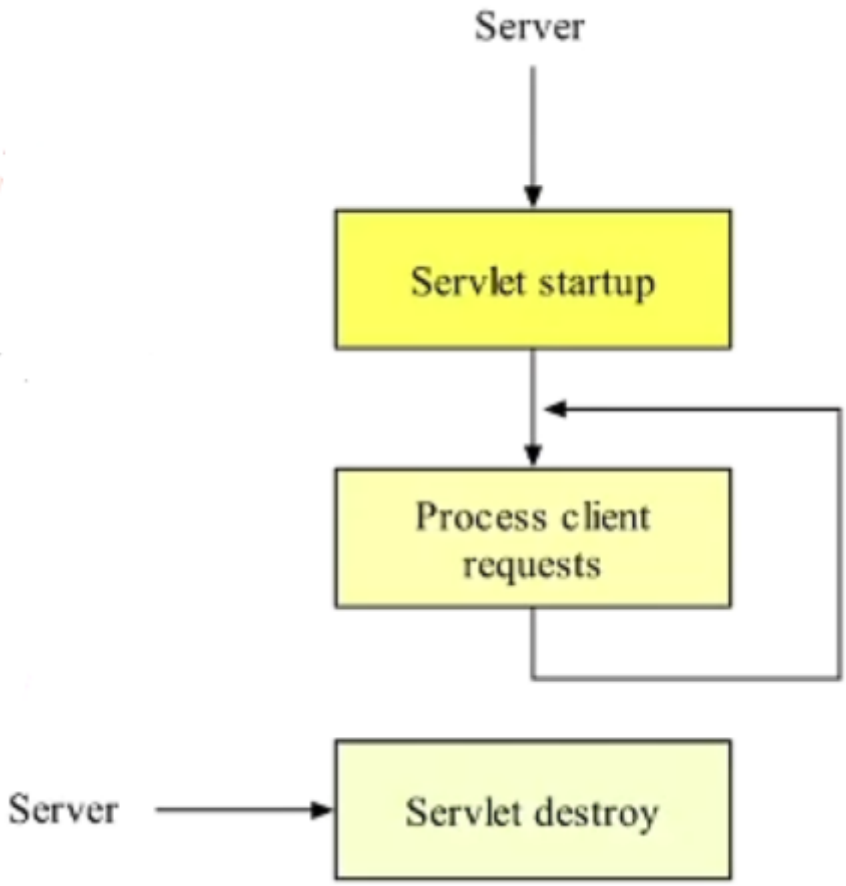
Server script - Servlet

Si tratta di un frammento di codice eseguito in java messo a disposizione attraverso una classe; questo codice viene scritto per estendere le funzionalità di un server. Le servlet non sono nate solo per lo sviluppo web, ma per estendere **tutti i server**. L'obbiettivo è di creare dei server aventi delle funzionalità di base che possono essere estesi con dei frammenti di codice scritti successivamente.

Il codice della servlet è eseguito dai **thread**, uno per ogni connessione/richiesta.

Ciclo di vita di una Servlet

Prevede 3 fasi di vita:



1. Una servlet viene caricata in memoria e viene creata un'istanza di quella classe. Viene quindi invocato un metodo **init**, che serve ad inizializzare la servlet stessa.
   La init viene invocata una sola volta (all'inizio).
2. Quando i client fanno richiesta di utilizzare le funzionalità di quel componente, viene invocato sulla servlet un altro metodo: **service**.
   Viene quindi invocato ogni volta che vi è una richiesta dal client verso il server.
3. Fase di eliminazione; il server che ospita la componente potrebbe decidere di distruggerla, invocando **destroy**. In questo metodo dobbiamo prevedere le operazioni duali previste in init, ovvero operazioni che ci permettono di dellocare in modo "pulito" la servlet.

Interfaccia servlet

```
public interface Servlet{
    public void init(ServletConfig conf) throw ServletException;
    public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException;
    public void destroy();

    public ServletConfig getServletConfig();
    public String getServletInfo();
}
```

ServletRequest

Metodo dell'interfaccia ServletRequest:

```
public interface ServletRequest{
    ServletInputStream getInputStream() throws IOException; // recuperiamo lo
stream per leggere la richiesta (char)
    BufferedReader getReader() throws IOException;

    String getParameter(STring name);
    Enumeration getParameterNames();
    String[] getParameterValues(String name);
    String getCharacterEncoding();
    int getContentLength();
    ...
}
```

Questi metodi sono ispirati all'HTTP: ad esempio **getProtocol()** possiamo recuperare a partire di un messaggio di richiesta il protocollo utilizzato.

ServletResponse

```
public interface ServletResponse{
    ServletOutputStream getOutputStream() throws IOException;   // possiamo
rispondere al client
    PrintWriter getWriter() throws IOExcpetion;

   ...
}
```

Queste interfacce sono da rivedere nel concetto specifico di HTTP, perchè sebbene possiamo usarle con ogni tipo di protocollo, si sono diffuse maggiormente con HTTP.

ServletConfig

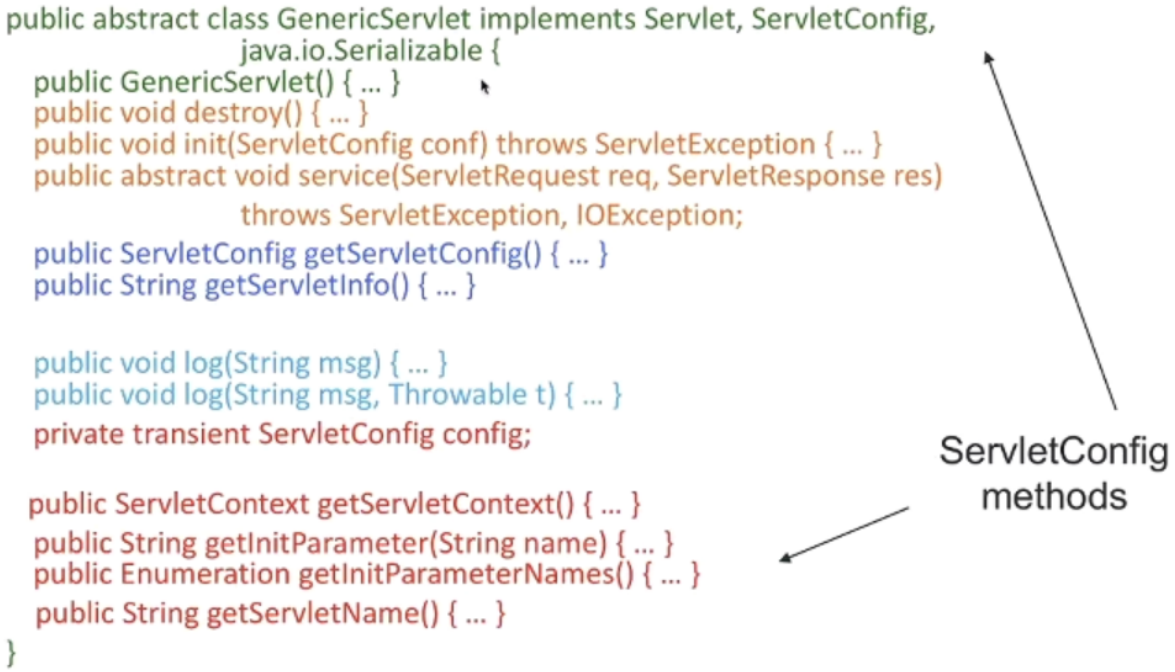
Con servletConfig recuperiamo informazioni sull'ambiente in cui la servlet esegue:

```
public interface ServletConfig{
    public ServletContext getServletContext();
    public String getInitParameter(String name);
    public Enumeration getInitParameterNames();
    public String getServletName();
}
```

> Ritorneremo su questa interfaccia

Generica Servlet

A partire dalle interfacce precedenti possiamo tirar su una servlet generica: Questa classe si chiama **GenericServlet** ed implementa le interfacce Servlet e ServletConfig:



Web Based Servlet

A partire da GenericServlet abbiamo una specializzazione: **HttpServlet**. Estende GenericServlet con l'obbiettivo di mettere a disposizione una classe di componenti che può essere impiegata per lo sviluppo web; la differenza con GenericServlet, è che in questa nuova classe abbiamo diversi **metodi di servizio**. Ognuno dei metodi java:

- **doGet(...), doPost(...), doPut(...), doDelete(...), doTrace(...), doHead(...), doOptions(...), service()**

Viene eseguito quando sarà ricevuto un messaggio di richiesta HTTP con un particolare metodo della request line.

HttpServlet

Estende GenericServlet e prevede tanti metodi addizionali (di servizio):

```
public abstract class HttpServlet extends GenericServlet implements
        java.io.Serializable {
    public void service(ServletRequest req, ServletResponse res) { ... }
    protected void service(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException { ... };
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException { ... };
    ...
}
```

Fine lezione 30