

# SVILUPPO DI SERVLET

---

# Nozioni fondamentali dell'API del servlet

▶ L'API del servlet fornisce le seguenti funzionalità ai servlet:

- ▶ Metodi di callback per l'inizializzazione e l'elaborazione delle richieste
- ▶ Metodi che consentono al servlet di ottenere informazioni sulla configurazione e sull'ambiente
- ▶ Accesso alle risorse specifiche del protocollo

# Vantaggi dell'API specifica del protocollo

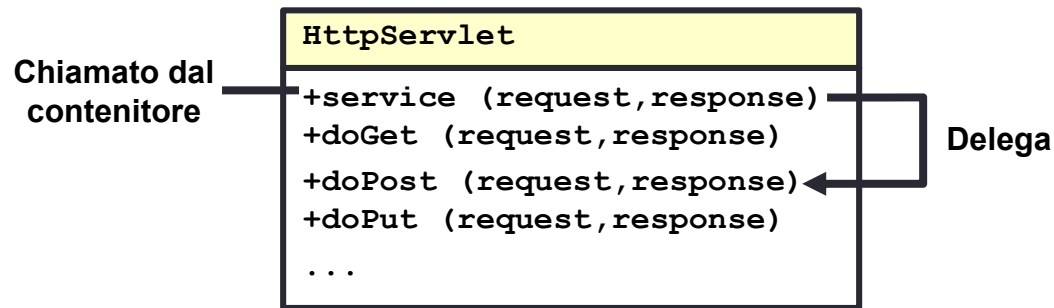
- ▶ È generalmente più conveniente lavorare con le interfacce e le classi specifiche del protocollo per una serie di motivi:
  - ▶ Le classi specifiche del protocollo offrono l'accesso agli oggetti specifici del protocollo, come l'implementazione `HttpSession`.
  - ▶ Gli argomenti di metodo e i valori restituiti sono definiti in termini di altri oggetti specifici del protocollo.
  - ▶ Le classi specifiche del protocollo offrono funzionalità di elaborazione standard per le operazioni comuni.

# Vantaggi della classe `HttpServlet`

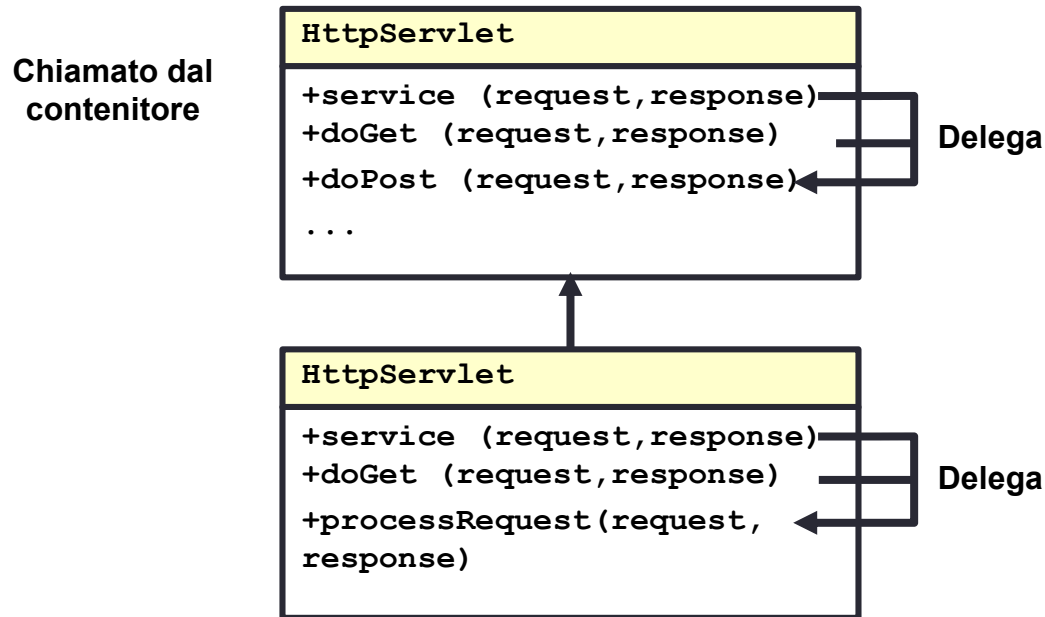
▶ I vantaggi dell'estensione della classe `HttpServlet` di base includono:

- ▶ Un metodo `init` senza argomenti semplificato che può essere sostituito per eseguire l'inizializzazione senza che sia necessario coinvolgere nel processo la classe di base
- ▶ Gestione standard dei tipi di richiesta `HTTP` che non interessano il servlet
- ▶ Argomenti relativi agli handler di richieste definiti in termini di oggetti `request` e `response` specifici di `HTTP`

# Il metodo `service`



# Metodi di gestione delle richieste



# Esempio: HelloServlet

```
@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    [...]
    private void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws IOException {
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("Hello world!");
        out.println("</html></body>");
        out.close();
    }
}
```

# Descrittori di distribuzione

▶ Un descrittore di distribuzione è un file di configurazione basato su XML. I descrittori di distribuzione delle applicazioni Web:

- ▶ Sono denominati `web.xml` e si trovano nella directory `WEB-INF`
  - ▶ Configurano informazioni sul mapping degli URL e altre impostazioni di configurazione
  - ▶ Sono facoltativi in Java EE 6. Vengono utilizzate le annotazioni per fornire informazioni di configurazione
  - ▶ Hanno la precedenza rispetto alla configurazione basata sull'annotazione
- ▶ Le annotazioni dei servlet possono essere disabilitate mediante l'attributo `metadata-complete` del tag `web-app`.

```
<web-app ... version="3.0" metadata-complete="true">
```



# Esempio di descrittore di distribuzione

- Per configurare un servlet e molti altri aspetti di un'applicazione Web, con un descrittore di distribuzione inserire un file `web.xml` nella directory `WEB-INF`.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
    <servlet-name>SomeName</servlet-name>
    <servlet-class>package.MyHttpServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SomeName</servlet-name>
    <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>
</web-app>
```

# Esempio: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">

  <servlet>
    <servlet-name>CiaoServlet</servlet-name>
    <servlet-class>it.prova.esempi.HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CiaoServlet</servlet-name>
    <url-pattern>/ciaoServlet</url-pattern>
  </servlet-mapping>

</web-app>
```

# Configurazione del servlet

▶ Senza la configurazione, un servlet non è dotato di un URL accessibile. A partire dalla specifica Servlet 3.0 (Java EE 6), vengono utilizzate le annotazioni per mappare gli URL ai servlet.

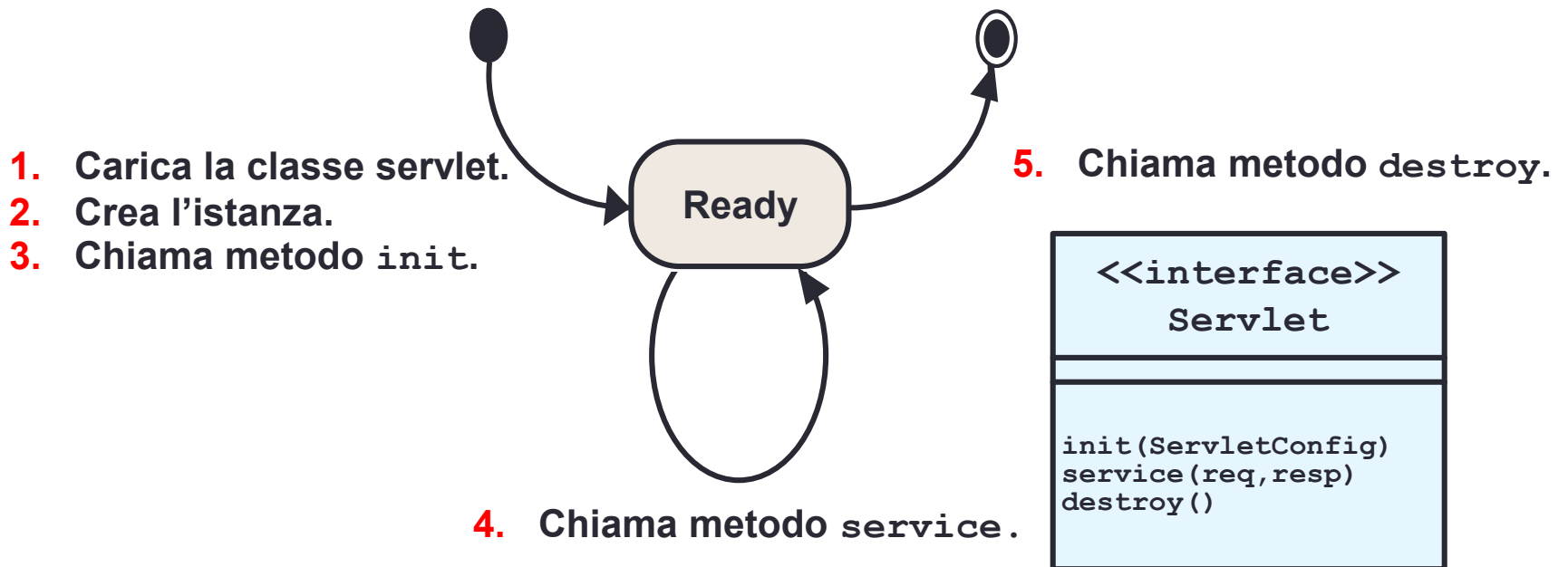
▶ Esempio di URL singolo:

```
@WebServlet("/myservlet")  
public class MyHttpServlet extends HttpServlet{  
    //...  
}
```

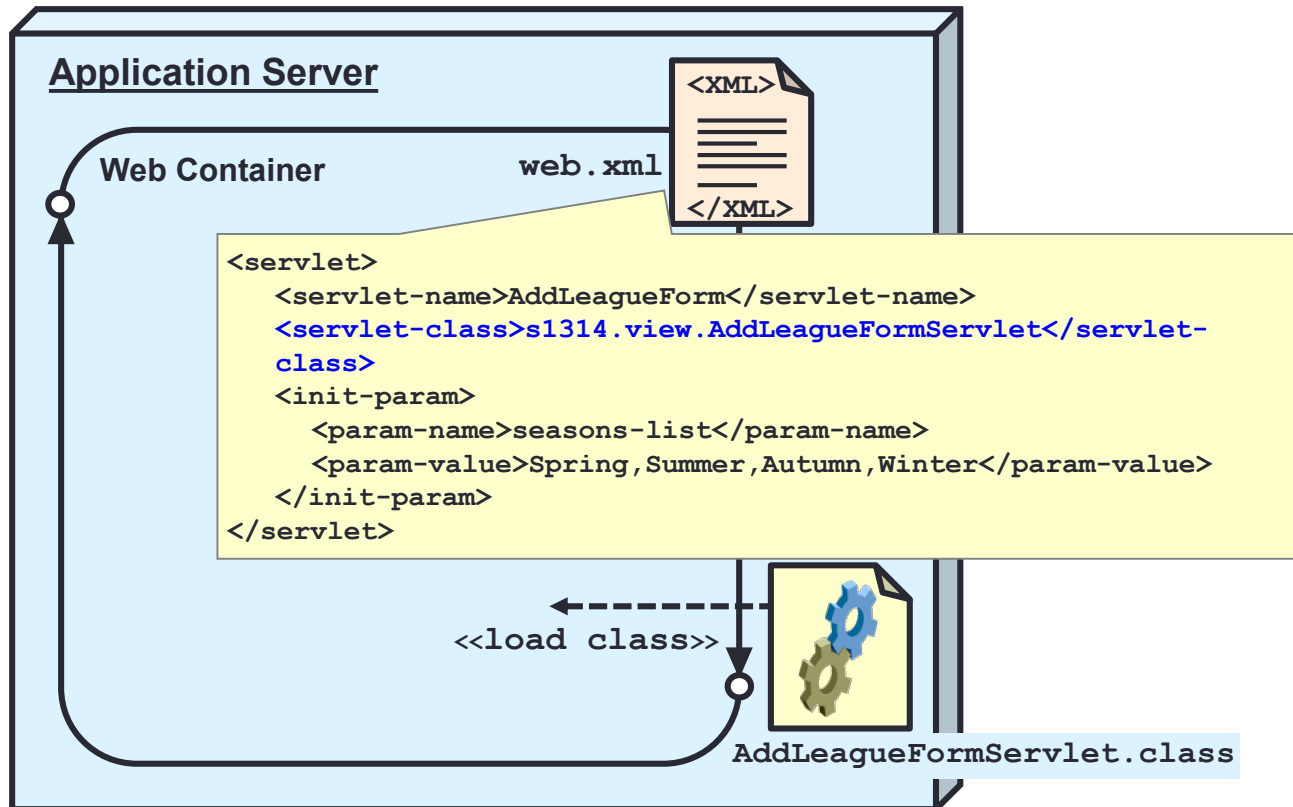
▶ Esempio di URL multiplo:

```
@WebServlet(name="SomeName", urlPatterns={"/myservlet", "/foo", "/bar"})  
public class MyHttpServlet extends HttpServlet{  
    //...  
}
```

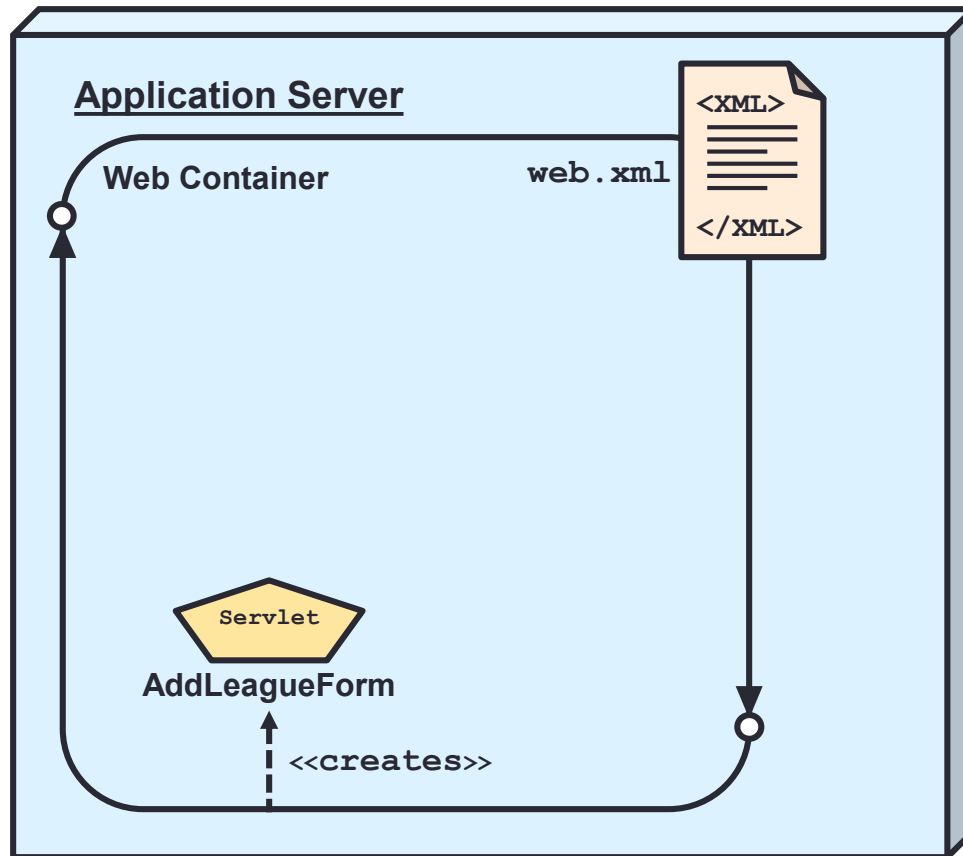
# Servlet Life Cycle: Overview



# Caricamento classe Servlet

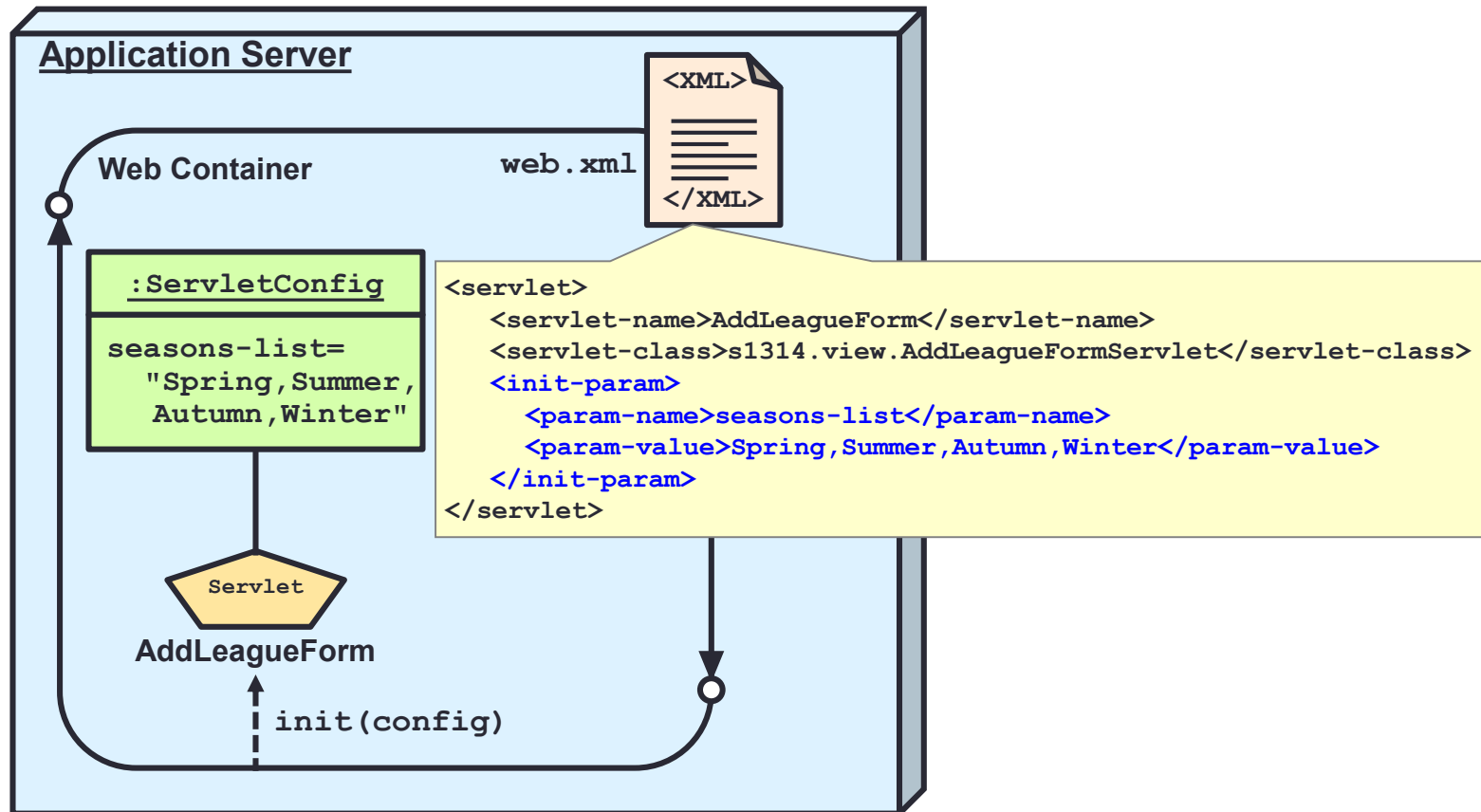


# One Instance Per Servlet Definition

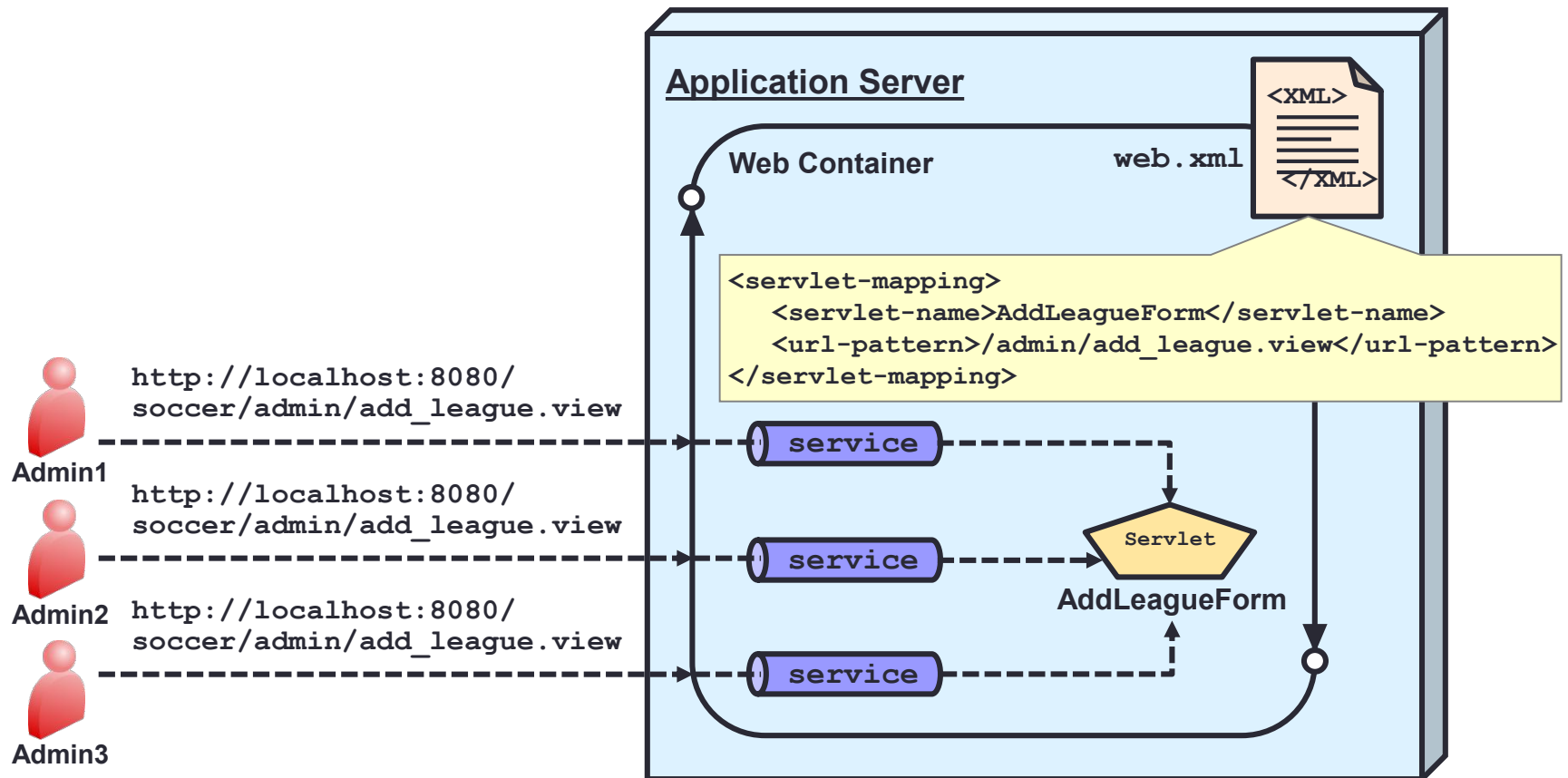


- Dalla versione 2.4 delle specifiche, il web container crea una sola istanza del servlet.

# init Lifecycle Method

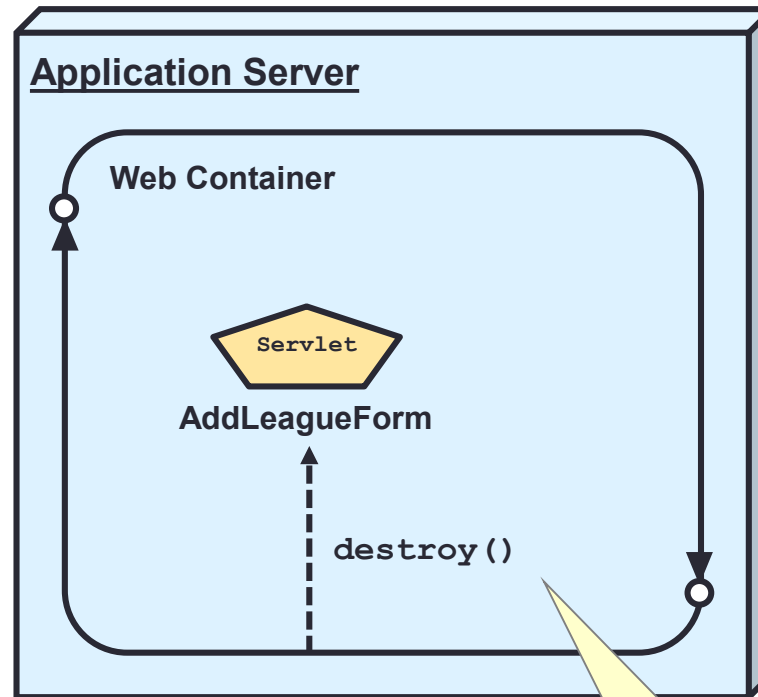


# service Lifecycle Method





# destroy Lifecycle Method



The web container can choose to destroy any servlet at any time.

# Utilizzo delle API di richiesta e risposta

▶ Il contenitore (Web) del servlet crea un oggetto request e un oggetto response per ogni nuova richiesta. Gli oggetti request e response vengono passati al metodo `service` del servlet.

▶ L'oggetto request:

- Fornisce informazioni sulla richiesta
- Consente al servlet di ottenere informazioni sugli utenti e di passare dati ad altri componenti Web

▶ L'oggetto response offre al servlet il meccanismo che consente di generare una risposta o un codice di errore al browser.

# Oggetto request

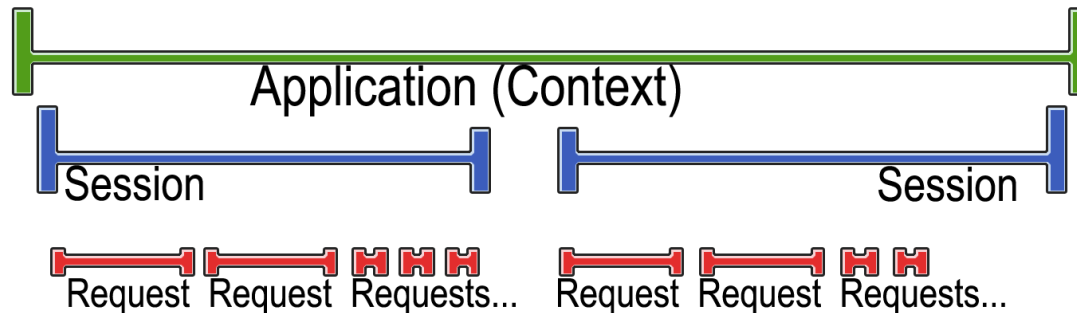
Metodi generici	Scopo
<code>getParameter</code>	Ottiene elementi dati di form
<code>getAttribute</code> e <code>setAttribute</code>	Ottiene e imposta attributi, che vengono utilizzati per passare dati tra i componenti
<code>getRequestDispatcher</code>	Ottiene un dispatcher richieste per il trasferimento del controllo a un altro componente
Metodi specifici di HTTP	Scopo
<code>getUserPrincipal</code>	L'identità dell'utente, se autenticato
<code>getCookies</code>	Ottiene l'identità dell'utente, se autenticato
<code>getSession</code>	Ottiene la sessione client

# Oggetto response

Metodi generici	Scopo
<code>getOutputStream,</code> <code>getWriter</code>	Ottiene un flusso o un processo di scrittura per l'invio di dati al browser
<code>setContentType</code>	Indica il tipo MIME del corpo della risposta
Metodi specifici di HTTP	Scopo
<code>encodeURL</code>	Aggiunge un ID di sessione a un URL
<code>addCookie</code>	Invia un cookie al browser
<code>sendError</code>	Invia un codice di errore HTTP

# Scope

Scope Name	Communication
page	Variabili locali: all'interno del metodo doGet o doPost
request	Variabili che durano la singola esecuzione di una richiesta da parte del browser, utilizzato per il passaggio di parametri tra view e servlet.
session	Variabili collegate alla sessione utente, valide tra più request, sono utilizzate per dati di sessione (login)
application	Variabili condivise tra tutti i componenti dell'applicazione



# Esempio di gestione dei dati di form e produzione di output

```
1      public void processRequest ( HttpServletRequest request,
2                                   HttpServletResponse response)
3      throws IOException {
4      response.setContentType ("text/plain");
5      PrintWriter out = response.getWriter();
6      String name = request.getParameter ("name");
7      if (name == null || name.length() == 0)
8      name = "anonymous";
9      out.println ("Hello, " + name);
10     out.close();
11 }
```

# Esempio passaggio parametri: form

```
<body>
```

```
  <div class="block">
```

```
    <form action="params" method="POST">
```

```
      <h2>Form</h2>
```

```
      Colori preferiti (valori multipli):<br/>
```

```
      <input type="checkbox" name="colori" value="blu"/> blu<br/>
```

```
      <input type="checkbox" name="colori" value="verde"/> verde<br/>
```

```
      <input type="checkbox" name="colori" value="rosso"/> rosso<br/>
```

# Esempio passaggio parametri: form

```
<p>Pensieri personali:</p>
```

```
Pensiero Uno <input type="text" name="uno"/><br/>
```

```
Pensiero Due <input type="text" name="due"/><br/>
```

```
<input type="submit" value="Submit"/>
```

```
</form>
```

```
</div>
```

```
</body>
```

```
</html>
```



# Esempio passaggio parametri: servlet

```
@WebServlet(name = "ParametersServlet", urlPatterns = {"/params"})  
public class ParametersServlet extends HttpServlet {  
  
    protected void processRequest(HttpServletRequest request,  
    HttpServletResponse response)  
        throws ServletException, IOException {  
        out.println("<tbody>");  
        Enumeration<String> paramNames =  
        request.getParameterNames();
```

# Esempio passaggio parametri: servlet

```
while (paramNames.hasMoreElements()) {  
    String paramName = paramNames.nextElement();  
    String[] paramValues =  
request.getParameterValues(paramName);  
    out.println("<tr>");  
    out.println("<td>" + paramName + "</td>");  
    out.println("<td>" + Arrays.toString(paramValues) +  
"</td>");  
    out.println("</tr>");  
}  
out.println("</tbody>");
```

# Inoltro del controllo e passaggio di dati

- ▶ Le operazioni di elaborazione e presentazione delle richieste sono separate per semplificare la gestione del software. Nessun componente esegue sia l'elaborazione che la presentazione.
- ▶ Il componente di elaborazione generalmente esegue le operazioni seguenti:
  - ▶ Svolge il proprio compito e raccoglie dati da visualizzare
  - ▶ Inserisce i dati nella richiesta
  - ▶ Trasferisce il controllo al componente di presentazione mediante un oggetto `RequestDispatcher`

# L'interfaccia `RequestDispatcher`

- ▶ Esistono due metodi `getRequestDispatcher("URI")` che restituiscono un'implementazione `RequestDispatcher`.
- ▶ Il metodo `ServletRequest`, che accetta percorsi relativi o percorsi che iniziano con il simbolo a `"/"`

```
RequestDispatcher requestDispatcher =  
request.getRequestDispatcher("relativeURI");
```

- ▶ Il metodo `ServletContext`, che deve iniziare con il simbolo `"/"`

```
RequestDispatcher requestDispatcher  
=    getServletContext().getRequestDispatcher  
    ("/ServletName");
```

# L'interfaccia `RequestDispatcher`

► Se un servlet non è dotato di un URI in quanto non è stato configurato per includere un tag `url-pattern`, è possibile recuperare un `RequestDispatcher` per il servlet dal contesto di runtime assegnando il nome del servlet di destinazione.

```
RequestDispatcher requestDispatcher =  
  
    getServletContext().getNamedDispatcher("ServletName");
```

## La destinazione `RequestDispatcher` e la radice di contesto

- ▶ L'argomento di `getRequestDispatcher` è un URI, tuttavia viene interpretato dal contenitore Web con riferimento al contesto dell'applicazione corrente. L'URI:
  - ▶ Deve iniziare con una barra (/) o essere relativo rispetto alla pagina corrente
  - ▶ *Non deve contenere una radice di contesto o essere un URI completo*
- ▶ Nell'applicazione di esempio bank il servlet ottiene il componente JSP al quale trasferirà il controllo utilizzando l'istruzione seguente:

```
getRequestDispatcher  
("/showCustomerDetails.jsp");
```

# I metodi `forward` e `include`

▶ L'interfaccia `RequestDispatcher` rende disponibili due metodi per il trasferimento del controllo da un servlet (il componente chiamante) a un componente di destinazione:

- ▶ `RequestDispatcher.forward`: generalmente utilizzato dai controller
- ▶ `RequestDispatcher.include`: generalmente utilizzato dalle viste

▶ Tra questi metodi, `forward` è leggermente più veloce ma non è in grado di unire l'output di un componente in quello di un altro.

# Trasferimento di dati nell'oggetto request

- ▶ L'oggetto request può trasportare dati tra componenti:
  - ▶ Nel componente chiamante:

```
CustomerData customerData = // get customer data  
request.setAttribute("customerData", customerData);  
requestDispatcher.forward(request, response);
```

- ▶ Se il componente di destinazione è un servlet:

```
CustomerData customerData = (CustomerData)  
request.getAttribute("customerData");
```

- ▶ Se il componente di destinazione è un componente JSP:

```
<jsp:useBean id="customerData"  
class="Bank.CustomerData" scope="request"/>
```

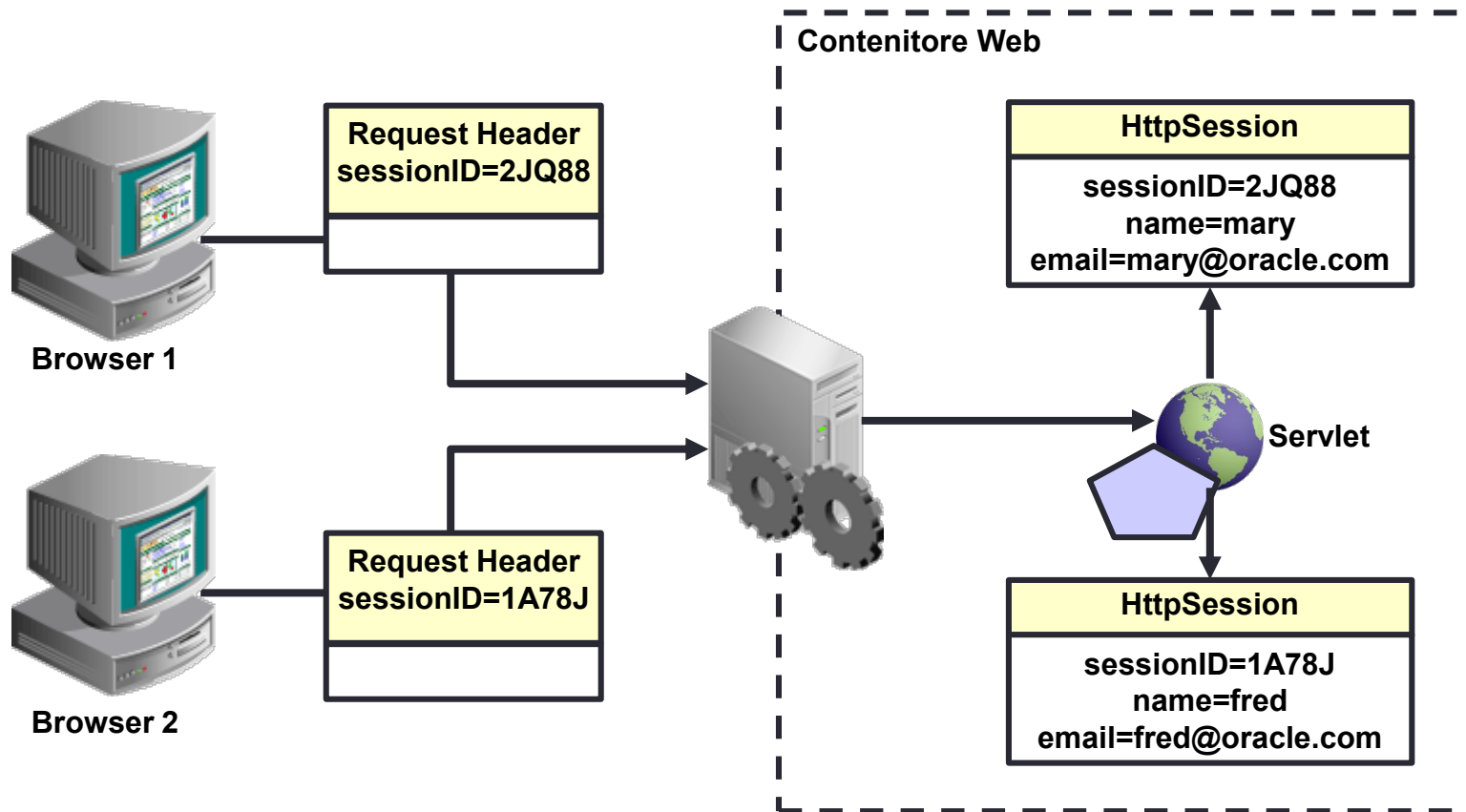


# Utilizzo dell'API di gestione delle sessioni

▶ Il modello di gestione delle sessioni della piattaforma Java EE nel livello Web si basa sull'interfaccia `HttpSession`. Un servlet può completare le azioni indicate di seguito.

- ▶ Stabilire se è stata appena creata una sessione
- ▶ Aggiungere un elemento denominato alla sessione
- ▶ Recuperare un elemento denominato dalla sessione
- ▶ Chiudere la sessione

# Modello di gestione delle sessioni del livello Web Java EE Platform



# Sessione e autenticazione

- ▶ I servizi di autenticazione e autorizzazione vengono forniti dalla specifica Servlet. Gli sviluppatori possono implementare una soluzione programmatica personalizzata.
  - ▶ Soluzione programmatica: dopo che un utente è stato autenticato, l'ID e lo stato di autenticazione dell'utente diventano generalmente parte della sessione.
  - ▶ Specifica del servlet: il servlet può utilizzare il metodo `request.getUserPrincipal`, anziché la sessione, per ottenere l'identificativo di login.

# Associazione di sessioni

► Per ogni richiesta il server deve essere in grado di identificare il browser specifico per selezionare l'oggetto session. Questa associazione delle sessioni viene eseguita utilizzando i cookie o la riscrittura degli URL.

<b>Tecnica di associazione delle sessioni</b>	<b>Vantaggi</b>	<b>Svantaggi</b>
<b>Cookie</b>	Il contenitore legge e scrive i cookie, pertanto non è necessario effettuare alcuna azione aggiuntiva.	Non tutti i browser supportano i cookie.
<b>Riscrittura URI</b>	La tecnica di riscrittura URI funziona senza il supporto dei cookie.	È necessario che l'ID di sessione venga aggiunto a ogni URL visualizzato dal browser.

# Timeout di sessione

- ▶ Il contenitore Web esegue il timeout delle sessioni dopo un periodo di inattività. L'applicazione Web:
  - ▶ Deve essere sviluppata per la normale gestione di questa situazione, generalmente mediante la reinizializzazione della sessione o il reindirizzamento a una pagina di login
  - ▶ Può impostare il timeout per l'intera applicazione nel descrittore `web.xml` oppure a livello di programmazione nelle singole sessioni

# Durata della sessione

► Il metodo `request.getSession` restituisce sempre una sessione. Il metodo `HttpSession.isNew` restituisce `true` in una delle situazioni seguenti:

- In caso di una nuova sessione con un nuovo browser.
- Se la sessione del browser corrente ha raggiunto il timeout prima di questa richiesta.

► Per chiudere subito una sessione, chiamare il rispettivo metodo `invalidate`:

```
1 if ("logout".equals(request.getParameter("action"))) {  
2     session.invalidate();  
3 }
```

# Recupero di un oggetto session

```
1          // Get a session object for the current client, creating
2          // a new session if necessary
3          HttpSession session = request.getSession();
4
5          // If this is a new session, initialize it
6          if (session.isNew()) {
7              // Initialize the session attributes
8              // to their start-of-session values
9              session.setAttribute ("account", new Account());
10             // ... other initialization
11         }
12
13         // Get this client's 'account' object
14         Account account = (Account) session.getAttribute("account");
```

# JDBC API

## ► Coonettività con I database:

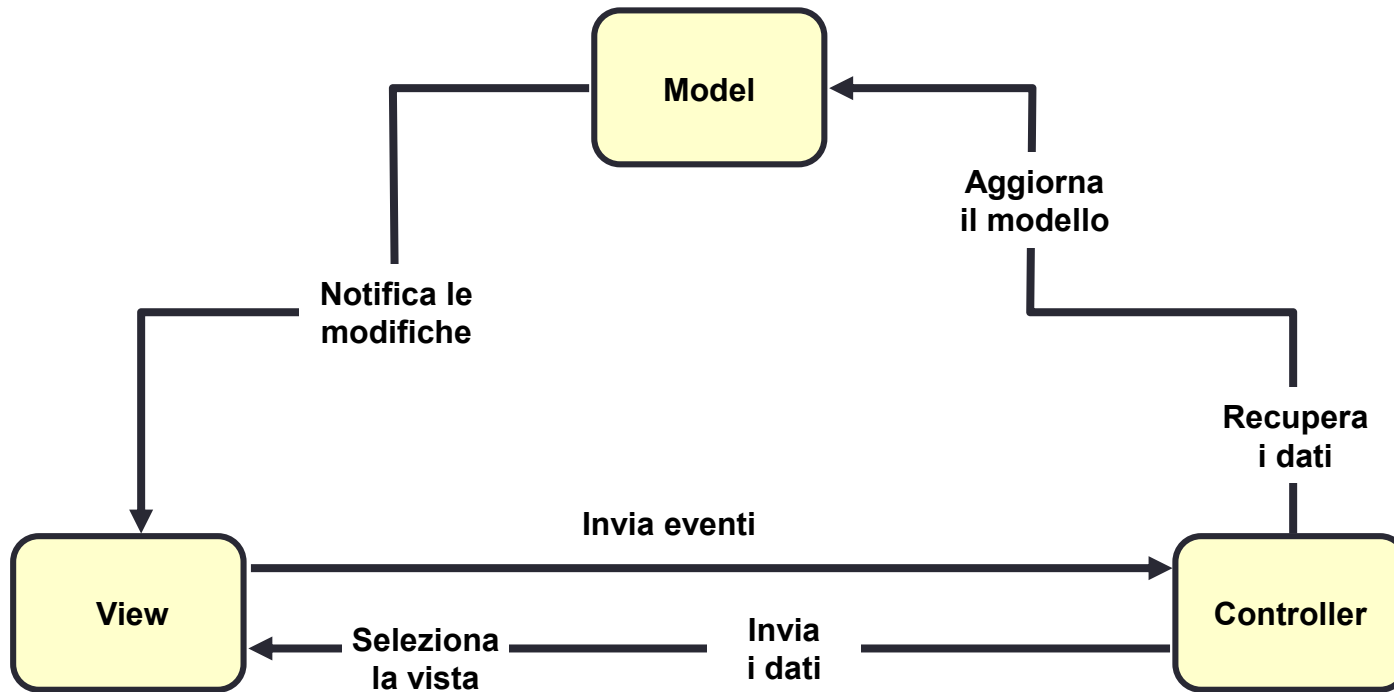
```
Connection connection = null;
Statement statement = null;
ResultSet rs = null;
try {
    connection = DriverManager.getConnection(URL, USER, PASS);
    statement = connection.createStatement();
    rs = statement.executeQuery("SELECT * FROM QuizCheck");
    while (rs.next()) {
        ...
    }
} catch (Exception e) {
    ...
} finally {
    //Close resources
}
```



# Pattern Java EE

- ▶ I pattern offrono una soluzione standard ai problemi di programmazione più noti.
- ▶ Il catalogo di pattern Java EE:
  - Consente a uno sviluppatore di creare applicazioni con tecnologia Java EE scalabili, affidabili e a elevate prestazioni
  - Presuppone l'utilizzo del linguaggio di programmazione Java e della piattaforma con tecnologia Java EE
  - In molti casi è strettamente correlato ai pattern Gang of Four (GoF)

# Architettura MVC tradizionale



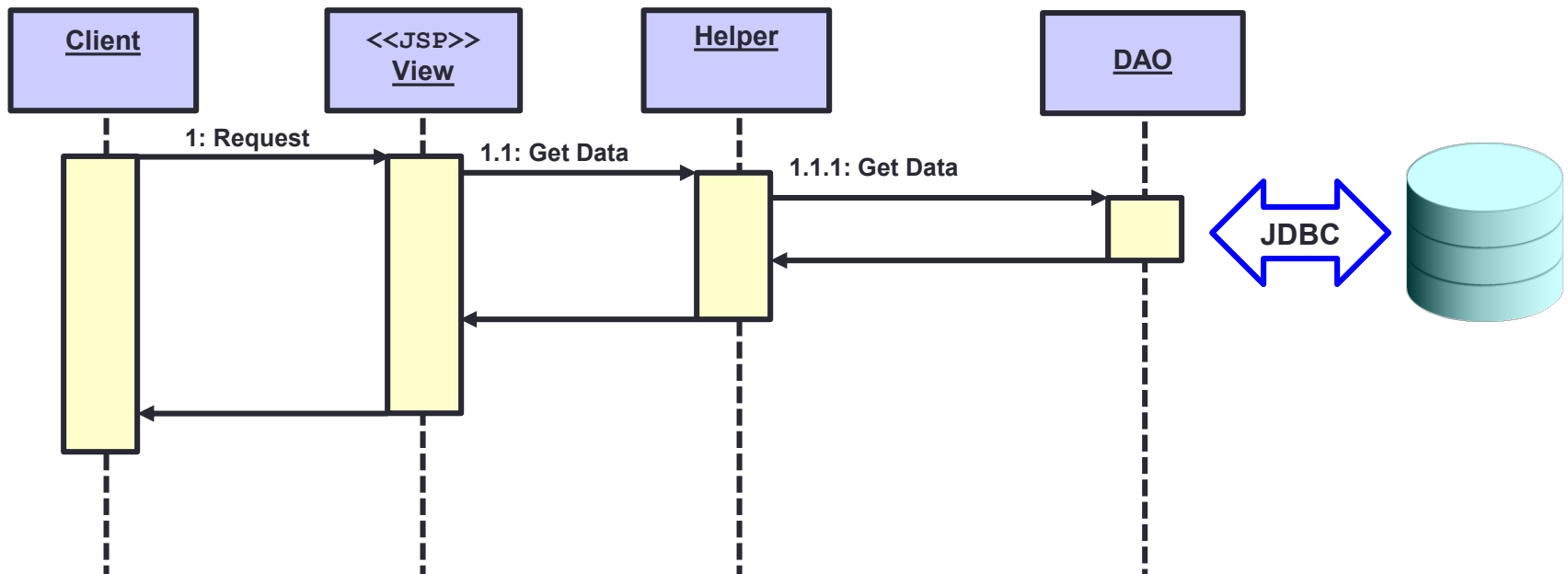
# Data Access Object Pattern (DAO)

- ▶ Il pattern DAO semplifica la gestione delle comunicazioni con il DB.

```
public List updateUtente(int id) throws SQLException {  
    // update mediante JDBC  
    Connection connection = DriverManager.getConnection(URL, USER, PASSWORD);  
    String sql = "UPDATE utente set attivo=true WHERE id=?";  
    PreparedStatement statement = connection.prepareStatement(sql);  
    statement.setInt(1, id);  
    result = statement.executeUpdate();  
    return result  
}  
  
public List getListaUtenti(String managerName) throws SQLException {  
    // select mediante JDBC  
    [...]  
}
```

# View Helper Pattern

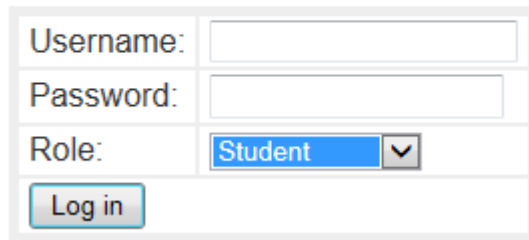
- Una view dovrebbe preoccuparsi solo della visualizzazione dei dati.
- Si crea una classe Helper che si occupa di recuperare il dato e di passarlo alla view occupandosi delle trasformazioni necessarie.



# Esercitazione

- ▶ Creare una pagina di login creando un form collegato alla action /login.do con una dropdown con una serie di ruoli (Student, Admin, Instructor)

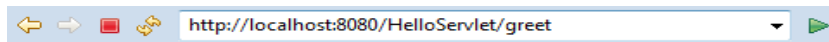
## Log in



A login form with the following fields:

- Username:
- Password:
- Role:  (dropdown menu)
- Log in button

- ▶ Creare la servlet loginServlet collegata al pattern login.do che controlla username e password (opzionale: collegandosi ad un db) e reindirizza l'utente (`response.sendRedirect(...)`)  
su una servlet di saluto o una pagina html di errore. Lo username deve essere memorizzato in sessione nell'attributo «user-name»



**Hello Fausto**

You are a logged in as a student

**Login Failed**

Try password: 123456

[Go back](#)

# Esercitazione

- ▶ La servlet greet deve controllare che in sessione non sia presente l'attributo «user-name» prima di visualizzare la pagina: nel caso non sia valorizzato deve reindirizzare sulla pagina di login

**Grazie**