**Interazione tra browser e application server**

**1. Il ruolo del browser**

Il browser è l'app principale con cui ci interfacciamo per navigare e accedere a risorse web. Quando inseriamo un indirizzo (URL) nel browser, questo può corrispondere a tre tipi di risorse:

* **Risorsa statica**: file come HTML, CSS, o PDF, serviti direttamente dall'application server.
* **Risorsa dinamica**: come le pagine JSP (Java Server Pages), generate dinamicamente dal server.
* **Servlet**: componenti Java che elaborano richieste e risposte. Le servlet e le JSP sono strettamente legate, in quanto condividono la stessa natura e hanno obiettivi simili, ovvero generare contenuti dinamici.

**2. Come il server processa le richieste**

* Quando il browser invia una richiesta (ad esempio per una pagina HTML), l'application server:
  + Determina qual è l'applicazione a cui appartiene la richiesta.
  + Cerca il file richiesto nel percorso relativo.

Se invece la risorsa è una JSP o una servlet:

* **JSP**: il server container (o servlet container, ad esempio Tomcat) riconosce il file JSP, lo compila in una classe Java (file .class) e richiama i metodi necessari per generare la risposta.
* **Servlet**: l'application server identifica e invoca direttamente la servlet associata all'URL.

**3. Differenza tra server generici e servlet container**

Non tutti i server sono in grado di elaborare JSP o servlet.

* I **servlet container** (come Tomcat o Jetty) supportano queste tecnologie, trasformando JSP in classi Java e gestendo servlet.
* I server che non sono servlet container non possono elaborare JSP o servlet.

**Protocolli e gestione delle richieste**

**4. SMTP e HTTP**

* Quando inviamo un'email tramite un'app come Outlook, viene utilizzato il protocollo **SMTP**. Questo protocollo definisce i comandi necessari per costruire e inviare un'email.
* Per le pagine web, invece, utilizziamo il protocollo **HTTP**, che definisce come inviare richieste e ricevere risposte tra il browser e il server.

**5. Errori comuni negli application server**

Un errore tipico avviene quando l'application server:

* Non riesce a trovare il **context root**, ossia la radice dell'applicazione associata all'URL.
* Non riconosce correttamente la risorsa richiesta (statica, JSP o servlet).

**Differenze tra doGet e doPost nelle servlet**

**6. Come funzionano doGet e doPost**

* **doGet**: quando inviamo un modulo tramite il metodo GET, i dati vengono aggiunti all'URL sotto forma di query string visibile, come ?chiave1=valore1&chiave2=valore2. Questo metodo è limitato nella lunghezza dei dati trasmessi.
* **doPost**: quando inviamo un modulo tramite POST, i dati vengono inclusi nel corpo della richiesta HTTP, quindi non sono visibili nell'URL. Questo metodo è adatto per trasmettere quantità maggiori di dati o informazioni sensibili.

**7. Struttura della query string**

La query string è una sequenza di coppie **chiave-valore** separate dal simbolo &. Ogni coppia associa una chiave (nome del campo del modulo) al suo valore. Ad esempio:

|  |
| --- |
| nome=Mario&cognome=Rossi  ​ |

**Sessione e gestione dello stato**

**8. Cos'è una sessione**

Una sessione permette di mantenere informazioni tra richieste successive dello stesso client.

* La sessione è simile a una **HashMap** sul server, in cui ogni client è identificato da un numero univoco (ID di sessione).
* Quando il browser invia una richiesta, include questo ID, che consente al server di recuperare i dati associati al client.

**9. Recupero della sessione**

Le sessioni possono essere utilizzate per salvare dati, come informazioni sull'utente dopo un login. Ad esempio:

|  |
| --- |
| HttpSession session = request.getSession();  session.setAttribute("username", "Mario");  ​ |

Questi dati possono essere recuperati successivamente in altre servlet:

|  |
| --- |
| String username = (String) session.getAttribute("username");  ​ |

**Formati di distribuzione delle applicazioni Java**

**10. Differenti formati**

* **JAR**: file compressi che contengono classi Java (file .class) organizzate in package.
* **WAR**: archivi per applicazioni web, contenenti risorse sia Java (servlet, JSP) che non Java (HTML, CSS).
* **EAR**: contenitori più grandi che includono sia JAR che WAR, usati per applicazioni enterprise.

**11. Creazione di servlet**

Esistono due modi per creare una servlet:

1. Utilizzando un **wizard** dell'IDE.
2. Configurandola manualmente nel file web.xml (nel folder WEB-INF), specificando il **mapping URL** (ad esempio, /miaServlet).

**Gestione degli errori**

**12. HTTP Status Codes**

Ogni risposta HTTP include uno status code che indica l'esito della richiesta. Ad esempio:

* **200 OK**: La richiesta è andata a buon fine.
* **404 Not Found**: La risorsa richiesta non è stata trovata.
* **500 Internal Server Error**: Errore interno del server.

**Cookie: gestione dello stato lato client**

**1. Cosa sono i cookie?**

I cookie sono piccoli file di testo che i server inviano ai browser, i quali li memorizzano localmente sul dispositivo dell'utente. Questi file vengono poi restituiti al server ogni volta che viene fatta una richiesta verso lo stesso dominio.

**2. Utilizzo dei cookie**

I cookie sono usati principalmente per:

* **Memorizzare informazioni di stato**: ad esempio, per sapere se un utente è loggato o ricordare preferenze.
* **Tracciare sessioni utente**: servono a identificare i client quando non si utilizzano sessioni lato server.
* **Analisi e personalizzazione**: come nel caso di dati per l'analytics o preferenze dell'utente.

**3. Differenza tra sessioni e cookie**

* Le **sessioni** mantengono lo stato lato server e usano un identificativo (ID) salvato nel client, spesso tramite cookie.
* I **cookie** memorizzano direttamente le informazioni lato client. Tuttavia, a differenza delle sessioni, i cookie possono essere letti o manipolati dal browser.

**Cookie nel contesto delle servlet**

**4. Creazione di un cookie**

In una servlet, possiamo creare e inviare un cookie al client con il seguente codice:

|  |
| --- |
| Cookie cookie = **new** Cookie("username", "Mario");  cookie.setMaxAge(3600); // Durata del cookie in secondi (1 ora)  response.addCookie(cookie);  ​ |

* **Nome**: "username" identifica il cookie.
* **Valore**: "Mario" è il valore associato.
* **Durata**: Con setMaxAge, definiamo per quanto tempo il cookie sarà valido (in secondi).

**5. Recupero di un cookie**

Quando il client invia una richiesta, i cookie vengono automaticamente inclusi. Possiamo leggerli così:

|  |
| --- |
| Cookie[] cookies = request.getCookies();  **if** (cookies != null) {  **for** (Cookie c : cookies) {  **if** ("username".equals(c.getName())) {             String username = c.getValue();             System.out.println("Username salvato: " + username);        }    }  }  ​ |

**6. Cookie e sicurezza**

Poiché i cookie sono memorizzati lato client, possono essere manipolati o intercettati. Per questo motivo:

* Evitiamo di salvare informazioni sensibili direttamente nei cookie.
* Usiamo l'attributo Secure per trasmettere cookie solo su connessioni HTTPS:

|  |
| --- |
| cookie.setSecure(true);  ​ |

* Impostiamo l'attributo HttpOnly per impedire l'accesso ai cookie tramite JavaScript (protezione contro attacchi XSS):

|  |
| --- |
| cookie.setHttpOnly(true);  ​ |

**Sessione e cookie: come lavorano insieme**

Quando utilizziamo una sessione in una servlet:

1. Il server crea un ID univoco per la sessione (spesso chiamato JSESSIONID).
2. Questo ID viene inviato al client come un cookie.
3. Il browser invia automaticamente il cookie al server con ogni richiesta successiva, permettendo al server di identificare l'utente e recuperare i dati associati alla sessione.

Esempio:

|  |
| --- |
| HttpSession session = request.getSession();  String sessionId = session.getId(); // Recupero dell'ID della sessione  ​ |

**7. Differenze tra cookie persistenti e di sessione**

* **Cookie di sessione**: vengono eliminati quando il browser è chiuso. Sono utili per mantenere lo stato durante una singola sessione.
* **Cookie persistenti**: hanno una durata specificata (setMaxAge) e rimangono memorizzati anche dopo la chiusura del browser.