Appunti di “Tecnologie informatiche per il web”

Indice

Architetture **1**

HTTP **2**

HTML **3**

CGI **3**

Servlet **4**

JDBC **5**

Templating **6**

Architetture

Un’architettura è un mix di risorse hardware, software e di rete. Esse sono classificate in base a vari fattori come:

- Tipo;
- Funzionalità;
- Risorse hardware utilizzate;
- Tipologia dei collegamenti;
- Insieme di moduli software.

L’architettura più semplice è quella del mainframe: tutto la funzionalità è su un unico nodo al quale si connettono un insieme di terminali “stupidi” (solo presentazione). L’architettura mainframe è molto semplice da gestire: basta gestire un solo terminale (il mainframe) e ciò garantisce consistenza di software, dati e logica applicativa tra tutti gli utenti. Il principale svantaggio è la monoliticità del software lato mainframe che è difficilmente riutilizzabile e non permette evoluzione.

L’architettura client-server è l’evoluzione dell’architettura mainframe: ora il codice è diviso tra un server, che gestisce i dati e logica applicativa condivisa, e un client che si occupa della presentazione grafica. Il client comunica con il server tramite query SQL, mentre il server risponde con i dati. Il principale svantaggio è la necessità di un “fat client” che gestisce la presentazione e la logica di business specifica del client. Inoltre il modello del client è strettamente collegato a quello del server, rendendo difficile la propagazione di aggiornamenti, oltre che non sfruttare al massimo le connessione al database. I vantaggi sono l’uso di un linguaggio standard di query (SQL) e la presenza di numerose ottimizzazioni sia lato server (grazie al DBMS) sia lato client (cache, pre-fetch…).

Il prossimo passo è stato l’architettura a 3 tier: tra il server e il client viene aggiunto in intermediario che meglio separa client e server: centralizza le connessioni al database e maschera il modello dei dati al client permettendo più scalabilità. Rimane il problema del legame tra il codice di client e del middle-tier. Il middle-tier comunica con i client tramite diversi metodi: RPC, orientamento a oggetti (Java RMI) e Message Oriented Middleware (MOM). I MOM hanno migliorato ulteriormente l’architettura rimuovendo la sincronizzazione implicata da metodi come RPC: lo scambio di messaggi è asincrono e si basa su un sistema di domanda/risposta gestiti tramite code.

Sulla base dell’architettura three-tier basata su MOM nasce il web:

- Il client è un browser web che interpreta un linguaggio standard di presentazione (HTML) e si occupa solo della presentazione.
- Il client comunica con il middle tier tramite un protocollo standard (HTTP) basato su richieste di risorse e risposte contenenti le risorse richieste.
- Il middle-tier fornisce le risorse statiche richieste e genera dinamicamente le pagine HTML da fornire al browser in base a query sui dati al database.

Il web ha spazzato qualsiasi architettura precedente grazie al concetto di cliente universale: un solo client può accedere a qualsiasi applicazione web.

La mancanza di opzioni di interattività di HTML ha portato ad un'evoluzione della prima architettura web (detta "Pure HTML") detta delle Rich Internet Application. Le RIA permettono di ridefinire gli eventi interattivi della pagine tramite codice applicativo (FLASH, JavaScript e AJAX) eseguito dentro il browser. Adesso si ha un "fat client" che più che essere un semplice interprete ora fornisce anche una API per definire la logica di presentazione.

Le applicazioni mobili possono essere sviluppate o come applicazioni client-server (app native) o come applicazioni web (progressive web app).

Una service oriented architecture (SOA) è un'architettura che permette di creare e pubblicare applicazioni autonome con un'interfaccia standard dette servizi. La comunicazioni tra servizi avviene tramite un protocollo standard. Gli obiettivi sono il decoupling tra componenti e layer architetturali per mascherare le diverse tecnologie. La SOA garantisce scalabilità per applicazioni distribuite ed eterogenee e flessibilità grazie alla possibilità dell'aggiunta dinamica di servizi.

Il cloud computing permette l'accesso tramite rete on-demand e conveniente di risorse computazionali condivise e configurabili.

HTTP

HTTP è un protocollo a livello applicativo per lo scambio di "ipertesti", ossia documenti che contengono riferimenti ad altri documenti. Il protocollo standardizza i nomi delle risorse (URL), le richieste e le risposte. HTTP è presente il diverse versioni: 0.9, 1.0, 1.1, 2. HTTP definisce diversi concetti:

1. Client: un'applicazione che stabilisce una connessione e manda richieste
2. Server: un'applicazione che accetta connessione e risponde con risposte
3. User Agent: il client che manda una richiesta (browser, editor, web crawlers…)
4. Origin server: il server sul quale risiede una risorsa o che la crea
5. Risorsa: un oggetto o servizio identificato da URI

Il browser è un'applicazione che manda richieste HTTP e riceve, interpreta le risposte e le visualizza. I browser supportano due linguaggi di descrizione (HTML e CSS) e un linguaggio di programmazione (JavaScript). Inoltre possiedono un meccanismo di estensioni tramite un'interfaccia a plugin.

Il server HTTP riceve e analizza le richieste HTTP. Inoltre si occupa dell'accesso alle risorse e dell'invio delle risposte HTTP oltre che ad altre funzionalità come il controllo degli accessi, esecuzione di programmi server-side, logging, monitoraggio e amministrazione, hosting virtuale, mapping URL e connessioni agli application servers. L'application server è un applicativo il cui compito è quello di facilitare l'esecuzione delle applicazioni utente lato server.

HTTP è un protocollo stateless (privo di stato): il protocollo governa un singolo round trip formato tra richiesta-riposta. Nessun dato è conservato tra due connessioni dello stesso client e/o diversi client. Tutto ciò implica che HTTP è anche sessionless. L'application server, invece, può essere stateful e preservare una sessione utente formata da più cicli richiesta-risposta. Esso, inoltre, non ha limitazioni riguardo all'interattività con altre risorse.

Alcuni altri elementi definiti dalla RFC di HTTP sono:

- Proxy: un intermediario che si comporta come server per un client e come client per un server. Questo meccanismo può essere usato per gestire l'accesso a delle risorse o per effettuare caching delle risorse.
- Gateway: un server che si comporta come intermediario per qualche server che non necessariamente comunica tramite HTTP. A differenza del proxy, il gateway riceve richieste come se fosse l'origin server della risorsa richiesta. Perciò il client potrebbe non essere a conoscenza del gateway.

I formati di richiesta e risposta sono molto semplici e si basano su:

- un metodo di richiesta (GET e POST in 1.0) con una serie di dati corollari detti header e un body.
- una risposta contente uno stato, una serie di dati corollari detti header e un body.

La versione più utilizzata è HTTP/1.1. Essa sistema i vari problemi di ottimizzazione delle risorse di rete della precedente versione e introduce numerose novità come: tunnel, chunked encoding, connessioni multi richiesta, negoziazione contenuto, cache management e nuovi metodi (ad esempio PUT, DELETE).

L'ultima versione, HTTP/2, come aveva fatto anche la 1.1, si occupa principalmente sull'ottimizzazione dell'uso delle risorse di rete e l'ottimizzazione dello scambio su TCP-IP.

La sicurezza in HTML è molto base. Le risorse sono organizzate in domain lato server (detti realm). I realm possono essere protetti. Una richiesta HTTP deve contenere un header di autorizzazione (credenziali trasmesse in chiaro, codificate con base64). Se le credenziali sono sbagliate o non ci sono viene mandata una risposta con codice 401 (Unauthorized) contenente lo header "authenticate" che istruisce il browser di mostrare il prompt per l'input delle credenziali. A causa della semplicità del meccanismo la sicurezza è spesso gestita dall'application server.

HTML

HTML (Hypertext Markup Language) è uno dei pilastri del web. È un linguaggio di markup talmente semplice da poter essere usato da tutti. Per "markup" si intende la possibilità di inframezzare dei comandi in mezzo al contenuto per descriverne il layout. HTML è un derivato di SGML (Standard Generalized Markup Language), una tecnologia per definire linguaggi di markup. Della stessa famiglia fanno parte XHTML e XML.

Di HTML vero e proprio ci sono 4 versioni. HTML 5, nonostante sia tecnicamente HTML, perde un po' lo spirito di markup di testo e diventa più una piattaforma per lo sviluppo di applicazioni web.

Il contenuto è racchiuso all'interno di tag che definisce le proprietà ad esso associate.

```
<!-- comment -->
<tag attribute="value">content</tag>
<self-closing-tag />
```

Il documento HTML è un documento ASCII con struttura generale:

```
<hmtl>
  <head> <!-- metadata -->
    <title>Titolo del documento</title>
  </head>
  <body>
    Document content
  </body>
</html>
```

Poiché HTML mischia contenuto, layout e presentazione in un solo documento, ciò ha reso difficile la separazione tra le tre. HTML 4 separa queste tre in 3 parti: HTML (contenuto/struttura), CSS (presentazione), Javascript e DOM (comportamento). Per aiutare la definizione della struttura del documento HTML 4 introduce un tag e un attributo fondamentale:

1. tag `<div>`: permette la delimitazione arbitraria di porzioni di contenuto;
2. attributo `class`: permette la classificazione di un tag HTML per permettere l'assegnamento di attributi in modo selettivo.

CGI

La Common Gateway Interface (CGI) è un'architettura per distribuire contenuto dinamico. CGI non è né un linguaggio di programmazione né un protocollo di comunicazione, ma è un'interfaccia tra il web server and le applicazioni che definisce delle variabili di comunicazione standard. L'interfaccia è implementata tramite un insieme di variabili d'ambiente. Un programma CGI (CGI script) può essere scritto in qualsiasi linguaggio.

Per invocare un CGI script, il client specifica l'URI del programma da eseguire. Il programma deve essere deployato in una location specifica sul web server. Il server riconoscerà dall'URI che la richiesta risorsa è un eseguibile (i permessi vanno settati correttamente nel web server). Il web server, quindi, decodifica i parametri passati dal client e inizializza

le opportune variabili. Il sever quindi lancia il programma in un nuovo processo. Il programma restituisce la risposta su `stdout`: prima viene scritto il `Content-type` heading (MIME type) e in seguito il body della risposta (markup HTML nel caso di `Content-type: text/html`). Il server infine completa la risposta HTTP e la manda al client.

Il client può mandare parametri in due modi:

1. Tramite una `GET` nella query string;
2. Tramite una `POST` nel body. Le richieste `POST` possono essere mandate tramite HTML con un form.

Le variabili CGI sono divise per tipo: relative a server, richiesta e header.

Variabili relative al server:

- `SERVER_SOFTWARE=name/version`: nome e versione del server software
- `SERVER_NAME`: hostname o IP del server
- `GATEWAY_INTERFACE=CGI/version`: versione CGI supportata

Variabili relative alla richiesta (le più importanti):

- `REQUEST_METHOD`: metodo HTTP
- `PATH_INFO`: informazioni sul path
- `QUERY_STRING`: la query string
- `CONTENT_TYPE`: il `Content-type` della richiesta
- `CONTENT_LENGTH`: lunghezza del content

Gli header HTTP della richiesta sono accessibili come variabili con prefisso `HTTP_`.

CGI è un'architettura obsoleta con problemi di sicurezza e performance:

1. Quando il server riceve una richiesta viene creato un nuovo processo (resource intensive); ciò implica anche che il programma CGI non può comunicare con il web server.
2. Il processo CGI viene terminato quando il programma termina. Ciò impedisce il riuso di risorse tra chiamate consecutive.
3. Esposizione del layout fisico del server.

Servlet

Java servlet è un'altra architettura per la generazione dinamica di contenuto.

Un servlet è piccolo applicativo inseribile nel nostro application server per aggiungere funzionalità. Un servlet container è un componente dell'application server che interagisce con i servlet. Esso è responsabile della gestione del lifecycle dei servlet, del mapping delle richieste HTTP sui servlet e dell'implementazione di oggetti "utility" per semplificare lo sviluppo.

Tutti i servlet sono gestiti tramite thread separati all'interno dello stesso processo persistente sull'application server. Una singola istanza viene create per ogni servlet in memoria. Ogni singola richiesta al servlet corrisponde ad un thread. Attenzione quindi va posta ai dati salvati nel servlet: sono disponibili ad ogni richiesta e l'accesso non è thread-safe.

I servlet usano classi e interfacce contenute in:

- `javax.servlet`: contiene classe a supporto di servlet generici (protocol-independent)
- `javax.servlet.http`: aggiunge funzionalità specifica a HTTP

Tutte le servlet implementano l'interfaccia `javax.servlet.Servlet`: `javax.servlet.GenericServlet` è protocol-independent, mentre `javax.servlet.http.HttpServlet` è un servlet HTTP.

- `GenericServlet` definisce il metodo `service()` che gestisce le richieste e prende in input un oggetto richiesta e un risposta.
- `HttpServlet` definisce `doGet()` e `doPost()` per gestire richieste `GET` e `POST` rispettivamente. Il metodo `service()` gestisce il setup e il dispatch di tutti i metodi `doXXX()`. NON bisogna sovrascrivere `service()`.

Il lifecycle delle servlet è il seguente:

1. Inizializzazione: quando il server viene avviato, il metodo `init()` del server viene chiamato.

2. Gestione richieste: quando una richiesta viene catturata dal server, il metodo `service()` (e relativi) vengono invocati e gestiscono la richiesta.
3. Distruzione: quando il server viene fermato, la funzione `destroy()` viene invocata e la garbage collection viene effettuata.

Ogni web application è legata and un `ServletContext` che definisce la visione del singolo servlet dell'applicazione che lo contiene. Quindi il context permette al singolo servlet di ottenere URL a risorse e di settare attributi che altri servlet nella stessa applicazione possono accedere. Ogni context ha una root che è l'endpoint al quale è deployata l'applicazione.

Di solito il path del context rispecchia una cartella fisica salvata sul server. Una applicazione può essere deployata in modalità 'unpacked', dove ogni file e directory esiste separatamente, o 'packed'. Quando deployata come packed, l'applicazione è un singolo file `WAR` e il nome del file è di default la root del context.

Le risorse accessibili dall'applicazione segue una struttura standard:

- La root contiene l'HTML/JSP, JS, CSS, immagini organizzati in subdirectory
- La cartella `WEB-INF` contiene risorse non accessibili dai client:
  - `web.xml`: file che descrive i servlet e vari componenti
  - `classes`: contiene i file `class` generati da java
  - `lib`: directory contenenti `JAR` richiesti dall'applicazione

A differenza di CGI, Java servlet non espone l'effettiva location degli script, ma invece fa un mapping tra l'URL della richiesta e un URL pattern dichiarato dalle server (ovviamente ogni servlet deve avere un pattern unico).

```
                              servlet path
                                  V
https://www.myserver.com/catalog/garden/abc
                             ^            ^
                   context path         PathInfo
```

Le regole di definizione dei mapping sono le seguenti:

1. Una stringa che inizia con `/` e finisce con `/*` è usato come mapping per il path della richiesta
2. Una stringa che inizia con `*` è usata come mapping per le estensioni di una risorsa nell'URL della richiesta
3. Una string vuota mappa al context root
4. Una stringa che contiene solo `/` indica il servlet di default dell'applicazione
5. Qualsiasi altra stringa è usata per un match completo

Il server matchera l'URI tranne context path e query string della richiesta ai servlet mapping come segue:

1. Se un match esatto viene trovato, si seleziona quel servlet;
2. Si percorre il percorso directory per directory e si seleziona il servlet con il match più lungo;
3. Se l'ultimo segmento dell'url contiene un'estensione e un servlet che gestisce quell'estensione esiste, viene selezionato;
4. Si usa la servlet di default.

Il mapping `/*` ha la proprietà di sovrascrivere tutti i mapping e gestire tutte le richieste.

JDBC

JDBC è un modo per connettere un'applicazione ad un database. La API di JDBC è composta dai seguenti componenti:

1. `DriverManager`: colui che gestisce il caricamento dei driver e la creazione della connessione al DB.
2. `Driver`: interfaccia che gestisce la comunicazione con il DB
3. `Connection`: connessione al DB
4. `Statement`: query al DB
5. `ResultSet`: risultato di una query al DB
6. `SQLException`: gestisce gli eventuali errori lanciati dal DB

Tutti i componenti sono contenuti in `java.sql` e `javax.sql`.

Il workflow base di JDBC in una servlet è: connessione al DB, preparazione ed esecuzione della query, processo del risultato, disconnessione dal database. Poiché la connessione può essere usata da più servlet, essa è una risorsa condivisa.

I parametri della connessione vengono definiti a livello di applicazione, tramite context parameters. Una soluzione più avanzata è usare una connection pool. Tutte le risorse allocate nel `init()`, vanno deallocate nel `destroy()`!

Le query da mandare al DB vengono preparate tramite un `PreparedStatement`. Si chiama così poiché sono 'preparate':

1. Viene prima mandato uno "statement template" al DB;
2. Il DB compila il template e lo salva senza eseguirlo;
3. In futuro, l'applicazione effettuerà il binding dei parametri del template con i valori effettivi e la query è eseguita.

Una query preparata può essere eseguita ripetutamente senza essere ricompilata, aumentando notevolmente le prestazioni del piano di accesso dati.

Attenzione particolare va posta sul binding dei parametri del template: ATTENZIONE ALLA SQL INJECTION! Sempre effettuare sanitizing delle query.

In fase di modifica, è opportuno, prima di eseguire la modifica, disabilitare la feature di `autoCommit` e effettuare il commit delle modifiche manualmente, per evitare che operazioni che potrebbero fallire lascino la base di dati in uno stato inconsistente. Di default, infatti, ogni query è gestita in una transazioni a sé stante, disabilitare questa feature ci permette di costruire a mano una transazione, tramite `commit()` e `rollback()`.

Templating

Fino ad adesso le servlet hanno dovuto scrivere tutto il loro contenuto a mano, incluso l'HTML. Includere HTML nel codice delle servlet rende molto difficile e brutto il codice interno delle servlet, oltre che mischiare presentazione e comportamento.

La soluzione è ribaltare la situazione: inseriremo del codice all'interno di un template. Il codice viene eseguito dal server e il risultato dell'esecuzione viene iniettato dentro un template. Il concetto di template è quello di una pseudo-pagina dove viene inserito tutto il contenuto statico e nel quale viene iniettato il codice effettivo.