PROTOCOLLO HTTP

HTTP è un protocollo di trasporto di contenuti generici alla base del web e di tutte le applicazioni costruite su tale piattaforma. Si appoggia ad un protocollo di trasporto esistente che nella maggior parte dei casi è il TCP.
Gestisce uno scambio elementare: il client invia una richiesta e il server produce una risposta: ognuna di queste richieste sono indipendenti così come è indipendente l'utilizzo di tale protocollo rispetto al contenuto.

Sia la richiesta che la risposta sono formate da un'intestazione e da un corpo (il cui contenuto è definito nell'intestazione). Tutte le transazioni HTTP riguardano operazioni su risorse informative specificate attraverso una URL, una stringa univoca costituita da più sottoparti (schema://hostname[:port]/path):

- schema indica il protocollo tramite cui agire
- hostname[:port] è l'indirizzo in cui il server si trova in attesa di connessioni
- path indica la posizione relativa dell'oggetto all'interno del server

La prima riga di una richiesta contiene:

- L'azione da svolgere: GET (richiede l'invio della risorsa indicata dalla URL), POST (invia un insieme di informazioni alla risorsa, tipicamente un programma, indicata dalla URL e richiede il documento risultante), PUT (richiede l'aggiornamento di una risorsa sul server identificata dalla URL), PATCH (richiede l'aggiornamento parziale di una risorsa), DELETE (richiede la distruzione della risorsa indicata)
- La URL relativa dell'oggetto su cui operare
- La versione del protocollo usata dal client

Un'azione è idempotente se l'effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta ed è sicura se non genera cambiamenti nello stato interno del server.

Le risposte rappresentano il risultato della richiesta tramite un codice ed una descrizione. Il sostanziale mutare nel tempo dei tipici contenuti trasferiti ha reso il meccanisco di trasporto poco efficiente (la latenza è un fattore critico): con HTTP/2, su una singola connessione TCP vengono multiplate più transazioni, ognuna delle quali è incapsulata in un pacchetto binario che la identifica. Più richieste possono essere emesse in cascata e le risposte non devono seguire l'ordine delle richieste: il pacchetto permette di ricostruire a quale richiesta faceva riferimento.

APPLICAZIONI HTTP

Lo standard prevede quattro possibili ruoli per le applicazioni che utilizzano HTTP:

- User-agent: applicazione che origina le richieste HTTP
- (Origin) Server: applicazione che ospita le risorse trasferibili. Accetta quindi richieste e genera le corrispondenti risorse
- Proxy: intermediario scelto dallo user-agent per inoltrare le richieste al server. Può filtrare le richieste, tradurre indirizzi e servirsi di una memoria temporanea per aumentare le prestazioni
- Gateway: intermediario trasparente allo user-agent che inoltra le richieste all'origin effettivo

Esistono due tipologie di proxy: quelli trasparenti dove la risposta viene inoltrata al client senza modifiche (è comunque capace di usare una cache e possiede un sistema di filtraggio) e quelli non trasparenti, che inoltrano tutte le richieste ma hanno la possibilità di modificare le risposte.
Basic Authentication: ad ogni gruppo di risorse tra loro correlate è associato un contesto di sicurezza chiamato realm. Per ogni realm è definita una lista di utenti autorizzati, con le relative password: quando il server manda una 401-Unauthorized, specifica il realm-name ed è compito del client formulare una nuova richiesta (dopo aver chiesto all'utente user e password) con le informazioni necessarie codificate in base64 (che però è una codifica reversibile quindi OCCHIO).
Digest authentication: invece di inviare la password, è possibile inviare un dato derivato in modo irreversibile dalla password. Client e Server eseguono la derivazione in modo indipendente, cosicchè se le due coincidono, il client ha dimostrato di conoscere la password senza inviarla in chiaro; occorre difendersi da tentativi di proporre la stessa derivazione da parte di un'eventuale attaccante.

Poichè HTTP tratta tutte le richieste come tra loro indipendenti, occorre introdurre a livello applicativo un meccanismo per supportare questa modalità di lavoro. Nasce così il concetto di sessione: un insieme di richieste, proveniente dallo stessso browser e dirette allo stesso server, confinate in un dato lasso di tempo, volte ad interagire con risorse tra loro correlate. E' possibile sia creare sessioni nominali che sessioni anonime.
Il web server associa una richiesta proveniente da un dato client ad un identificatore univoco (SessionID): tale identificatore viene comunicato al client per le successive richieste. Dopo un certo periodo di inattività dell'utente, la sessione scade e ne viene perso lo stato; l'ID può essere sia generato sequenzialmente a partire da un valore dato oppure essere generato casualmente. In ogni caso, tale ID viene trasferito al client in diversi modi: passaggio esplicito nel contenuto del documento, utilizzando i cookies, riscrivendo i link o coordinando le sessioni lato client.

ANNOTAZIONI JAVA

Le annotazioni permettono di inserire dei metadati che descrivono le caratteristiche di una classe. Esse possono essere usate per informare il compilatore (rilevare errori o evitare i warning), generare informazioni durante la compilazione di un pacchetto software o condizionare l'esecuzione di un programma (è possibile esaminare una classe durante l'esecuzione per verificare la presenza di annotazioni e comportarsi di conseguenza).

```
public @interface RequestForEnhancement{
        int id();
        String synopsis();
        String date();
}
```

Il simbolo @ precede la keyword interface e i metodi definiscono le proprietà dell'annotazione. Una volta definito il tipo di annotazione, questo può essere usato per annotare delle dichiarazioni. Le dichiarazioni dei metodi non devono avere parametri e non devono prevedere clausole di tipo throws, infine i tipi di ritorno devono essere tipi primitivi, String, Class, enum o array di questi.
Le meta-annotazioni servono per annotare le dichiarazioni delle annotazioni (annotazioni di annotazioni), esempi sono: @Target, @Retention, @Documented, @Inherited. Usando JPA è possibile anche configurare il comportamento delle proprie entità usando le annotazioni (come @Entity, @OneToOne, @PrimaryKey o @Column).
Con la nascita delle annotazioni è nato anche il progetto Lombok, che è un'estensione del compilatore in grado di convertire annotazioni presenti nelle classi in codice sorgente Java e quindi permette di trasformare un oggetto annotato con @Data aggiungendo getter/setter/costruttore in automatico. Le classi annotate con @Data, inoltre, hanno tutti i campi non esplicitamente final mutabili, mentre annotando con @Value tutti i campi vengono resi privati e immutabili. In conclusione, le annotazioni Java rappresentano una standardizzazione dell'approccio di annotazione del codice: non riguardano direttamente la semantica del programma ma possono semplificare notevolmente la quantità di codice che è necessario scrivere a mano senza compromettere le funzionalità.

MAVEN

Maven è uno strumento a supporto del processo di creazione e manutenzione del software. E' una versione evoluta di make, con supporto all'importazione delle dipendenze, alla creazione di moduli differenti ed all'utilizzo di plugin per la compilazione e la messa in campo. Supporta le diverse fasi del ciclo di vita di un prodotto software, e le singole applicazioni sono eseguite da plugin.
E' necessario un file POM (Project Object Model), un file XML che descrive la struttura di un progetto; il progetto e ciascun artefatto da esso utilizzato ven-

gono identificati in modo univoco tramite la notazione GAV(groupId : artifactId : version):

- groupId: identificatore arbitrario del progetto di solito derivato dal package Java
- artifactId: nome arbitrario del progetto
- version: versione del progetto

Un file POM può ereditare la propria configurazione e in tal modo facilita la gestione di progetti composti da più sotto-progetti.
Un progetto Maven può essere costituito da un insieme di moduli ognuno dei quali è contenuto in una sotto-cartella ed ha un proprio file POM; nella cartella del progetto è presente il file POM complessivo: target è la cartella di lavoro di default e src contiene tutti i file sorgente.

Per default, la creazione di un artefatto passa attraverso le seguenti fasi principali: generazione dei sorgenti e delle risorse (generate*, qui vengono eseguiti eventuali pre-processor del codice sorgente), compilazione (compile), esecuzione dei test sui moduli (test), impacchettamento (package), test di integrazione (integration-testing), installazione (install) e messa in campo (depoly). Separatamente viene considerata la fase di pulizia generale (clean, che elimina tutti i file intermedi e di uscita).
Nella maggior parte dei casi, un progetto reale si basa su un insieme di altri progetti/librerie. Per ogni elemento di tipo <dependency> occorre indicare GroupId, ArtifactId, Version, Scope; quest'ultimo <scope> definisce in quale fase del ciclo di vita del progetto la dipendenza dovrà essere considerata: compile (indica che la risorsa è necessaria per l'intero ciclo di vita), provided (indica che la risorsa è necessaria durante la fase di compilazione, ma non dovrà essere inclusa nell'artefatto finale in quanto messa a disposizione dall'ambiente di esecuzione), runtime (indica una risorsa da includere solo in fase di esecuzione) e test (indica una risorsa necessaria esclusivamente in fase di test).

Per ogni risorsa indicata come dipendenza, il corrispondete modulo software viene scaricato dal repository ed archiviato nella cache locale (viene scaricato inoltre anche il file POM corrispondente). Se, dall'analisi del POM, un modulo scaricato dichiaradi dipendere da altri moduli, questi vengono scaricati transitivamente (ma solo se sono "compile" e "runtime").
Tutte le operazioni svolte da Maven avvengono grazie all'attivazione di appositi plugin, ossia classi Java che estendono una classe astratta e implementano il metodo void execute(); un'apposita sezione del file POM (<build>) elenca l'insieme di plugin che saranno utilizzati nella gestione del progetto.

APPLICAZIONI JAVAEE

Le applicazione web dinamiche implementano una data logica lato server fornendo un'interfaccia utente basata su pagine html e contenuti multimediali: il livello UI viene eseguito sul client e mostra l'interfaccia utente, il livello controllo viene eseguito sul server e si occupa di implementare la logica e il livello dati viene eseguito su una base dati e si occupa di memorizzare i dati applicativi.Contrariamente alle applicazioni stand alone, quelle web sono eseguite in modo reattivo (tutte le volte che ricevono una richiesta producono una risposta specifica per la sessione in corso): solitamente, l'applicazione reagisce alla prima richiesta mostrando una "home page". Ciascuna richiesta può contenere parametri aggiuntivi (tramite form ecc..) e il livello di controllo deve: verificare la validità dei parametri ricevuti, memorizzare temporaneamente i parametri per poterli utilizzare e generare una risposta appropriata, permettendo all'utente di svolegere il proprio compito. La pagina inviata può poi essere personalizzata in base alle interazioni precedenti.

Nella navigazione pagina per pagina, a fronte di ogni richiesta viene generata una risposta che contiene una pagina HTML completa, mentre in quella progressiva a pagina singola solo la prima richiesta contiene la struttura della pagina, e le richieste successive sono effettuate con il paradigma AJAX e prevedono il trasferimento solo del frammento di struttura che deve essere modificato o di una descrizione sintetica dello stesso.
La navigazione a pagina singola è basata sull'utilizzo di codice JavaScript lato client per richiedere il frammento da aggiornare e si utilizzano le funzioni per la manipolazione del DOM per inserirle nella pagina; l'uso di librerie quali Vue, React o Angular semplifica la gestione del processo. I limiti di tale navigazione si incontrano quando ci si interfaccia con i motori di ricerca, con il tasto "indietro" dei browser e per i tempi di caricamento.

Le applicazioni JavaEE sono un insieme modulare di classi Java, messe in campo all'interno di un programma contenitore che ne gestisce il ciclo di vita e l'esecuzione: esse sono costituite da un insieme di componenti responsabili di gestire l'elaborazione di richieste HTTP e costruire le corrispondenti risposte, delegando in parte o in toto l'implementazione della logica applicativa ad uno strato di servizio che, a sua volta, può appoggiarsi ad uno strato di memorizzazione persistente. Esse sono indipendenti dallo specifico contenitore, ossia applicazioni in esecuzione all'interno di un calcolatore server. Monitorano le connessioni, applicano ad ogni richiesta un insieme di funzioni base, determinano l'applicazione cui è destinata la richiesta e la classe al suo interno responsabile di elaborarla, instanziandola se necessario e offrono un insieme di servizi standardizzati ai singoli componenti, semplificandone l'implementazione. Utilizzano un template standard per descrivere le applicazioni web ed hanno un file XML di configurazione da cui il contenitore deduce tutti i dettagli applicativi (politiche di gestione, parametri di configurazione, corrispondenza fra URL e nomi delle

classi ad esempio).

Le applicazioni JavaEE sono normalmente distribuite sotto forma di file ".war", al cui interno sono presenti sia i contenuti che la configurazione dell'applicazione e il cui nome rappresenta di default la URL di base in cui verrà pubblicata. Ci sono due sotto-cartelle obbligatorie (META-INF/MANIFEST.MF e WEB-INF/web.xml) e per installarla basta semplicemente copiare il file .war nell'apposita cartella del contenitore. Ci sono poi le componenti elementari di un'applicazione JavaEE (Servlet, filtri, Listener, pagine JSP).

SERVLET API

Un Servlet è una classe Java responsabile di gestire le richieste ricevute da un contenitore, modellando il paradigma richiesta/elaborazione/risposta tipico delle applicazioni lato server. Il contenitore ne gestisce completamente il ciclo di vita. Le sue API sono un insieme di interfacce e classi che definiscono, a basso livello, l'interazione tra codice applicativo lato server e contenitore. javax.servlet contiene le classi e le interfacce che definiscono il comportamento generico di un servlet, javax.servlet.http contiene le classi e le interfacce che modellano il comportamento di richieste/risposte trasportate attraverso il protocollo HTTP. ServletRequest rappresenta la richiesta del client e ServletResponse rappresenta la risposta del servlet al client (estese da HTTPServletRequest e HTTPServletResponse).

Il contenitore crea un'unica istanza del servlet e ne invoca il metodo init(), e finchè esso non ritorna il contenitore non invoca nessun altro metodo. Ad ogni richiesta, il contenitore invoca il metodo service() utilizzando un thread differente e infine, a proprio piacimento, il contenitore decide di invocare il metodo destroy() per rimuovere il servlet.
L'annotazione WebServlet permette di indicare la configurazione: il valore di default indica l'elenco di URL che vengono gestite dall'istanza del servlet, urlPatterns è equivalente al valore di default, initParams indica eventuali parametri di inizializzazione, loadOnStartup indica la politica di instanza, asyncSupported indica se la richiesta può essere gestita in modalità asincrona.

Metodi di HTTPServletRequest: getParameter(String name), ServletInputStream getInputStream(), BufferedReader getReader(), String getMethod(), Enumeration getParameterNames(), String getHeader(String headerName), void setAttribute(String name, Object value), Object getAttribute(String name)

Metodi di HTTPServletResponse: void setContentType(String contentType), void setStatusCode(int statusCode), void addCookie(Cookie c), void addHeader(String name, String value).

Metodi di HTTPSession: void setAttribute(String name, Object value), Object getAttribute(String name), void removeAttribute(String name), String getId(),

boolean isNew(), void invalidate()

Il ServletContext è un'interfaccia che definisce le interazioni tra servlet e suo contenitore. Nel caso di contenitori basati su una singola virtual machine, ogni applicazione web ha un unico contesto, mentre nel caso di più VM ci sono tante istanze di questa interfaccia quante sono le macchine virtuali attive.
In ogni caso, permette di creare/aggiungere/rimuovere servlet/filtri/listener al contenitore ed offre un meccanismo per risalire al tipo di contenuto di un file presente nel file system locale; inoltre, è in grado di fornire informazioni sul contenitore, sulle API supportate e sulle politiche in essere per la gestione dell'applicazione.

Quali sono i vantaggi di un Servlet? Di sicuro sono veloci, efficienti, scalabili, flessibili e persistenti; in più, il contenitore garantisce la separazione fisica tra applicazioni. Il problema è che è complicato modificare il codice HTML di risposta, la presentazione e la logica sono fortemente accoppiate e l'architettura rischia di diventare complessa se si prendono in considerazione le caratteristiche più avanzate.

FILTRI

Un filtro è un componente java che trasforma le richieste inviate ad un servlet o le risposte da esso generate. Ciascun filtro riceve le richieste destinate ad un (o un insieme di) URL e decide, in base alla propria logica, se inoltrare la richiesta al destinatario, modificare uno o più parametri o reindirizzare la richiesta verso un indirizzo differente. Lo stesso funzionamento vale anche per la risposta, e ci possono essere più filtri collegati in cascata, il tutto gestito dal file web.xml.
Come nel caso dei Servlet, anche per i filtri è possibile indicare il set di URL su cui devono essere applicati in due modi alternativi: tramite l'annotazione @WebFilter o inserendo delle sezioni nel file web.xml. Parametri:

- urlPatterns – indica la lista delle espressioni regolari che descrivono l'insieme delle URL su cui il filtro si applica
- servletNames – indica l'insieme dei servlet su cui deve essere applicato il filtro
- initParams – i parametri di configurazione del filtro
- filterName – il nome simbolico associato al filtro
- dispatcherType – indica a quale fase della gestione della richiesta debba essere applicato il filtro
- asynchSupported – indica se il filtro può operare in modalità asincrona

Ci sono due sezioni nel file web.xml: "filter" descrive un filtro mentre "filter-mapping" esprime la corrispondenza tra URL e filtri da utilizzare. Nel caso in cui ad una data URL corrispondano due o più filtri, questi sono applicati nell'ordine in cui le rispettive sezioni filter-mapping compaiono nel file web.xml

JSP - JAVA SERVER PAGES

Documenti di testo che descrivono come elaborare una richiesta per produrre una risposta in base ai parametri ricevuti. JSP è un modello di programmazione basata sul linguaggio Java e usato per creare dinamicamente contenuti web. Le pagine JSP contengono testo, frammenti di codice e direttive per l'ambiente di esecuzione e vengono create e gestite nel contenitore. Esse generano contenuto dinamico che viene visualizzato in un browser generico (nonostante il client non veda mai la pagina JSP iniziale, ma solo i risultati della sua esecuzione) grazie ai frammenti di codice Java, per cui semplici oggetti (JavaBeans) possono essere utilizzati per incapsulare data storage o altre operazioni di tipo business.
Che vantaggi porta l'utilizzo di JSP? Si può modificare il contenuto delle pagine web senza cambiare il codice HTML di layout e vi è quindi una separazione tra i contenuti dinamici e quelli statici; inoltre, i JavaBeans e i tag personalizzati permettono il riutilizzo di codice e l'utilizzo di semantiche dichiarative. L'unico problema è che il programmatore deve occuparsi di progettare ed implementare una macchina a stati opportuna per gestire l'interazione con l'utente.

1. Il contenitore riceve la richiesta della pagina JSP (direttamente o tramite server HTTP)
2. Il file viene identificato e trasformato in classi sorgenti Java (tramite pageCompiler)
3. Il compilatore Java compila la classe prodotta
4. La richiesta ricevuta e i relativi parametri vengono inviati all'oggetto generato (servlet), che la elabora
5. Il risultato viene inviato al client

Nel caso in cui venissero ricevute più richieste per la stessa risorsa, vengono omessi i passi 2 e 3

Le pagine JSP sono composte da tag standard HTML e tag JSP:

- Commenti

  ```
  <%-- comments.. --%>
  ```

- Direttive

  ```
  <%@ page import="java.util.*" %>
  ```

- Scriptlet

  ```
  <% if( session.isNew() ){ <p> Welcome </p> } %>
  <%! some declaration %>
  <%= expression %>
  ```

Permettono al programmatore di accedere e memorizzare informazioni relative all'elaborazione corrente (request = HttpServletRequest, response = HttpServletResponse, out = JspWriter)

Per redirigere la pagina JSP ad una determinata URL, per separare l'elaborazione della richiesta dalla visualizzazione della risposta: <jsp:forward page="URL" />
Inserimento di una pagina identificata da una URL durante l'esecuzione: <jsp:include page="URL" flush="true" />

La pagina JSP viene trasformata in classe java

```
public void _jspService(
        HttpServletRequest req,
        HttpServletResponse res) throws IOException, ServletException;
```

Il codice di _jspService è composto da: inizializzazione e gestione degli errori e codice contenuto nello scriptlet e dalle richieste di stampa di testo sull'oggetto "out" incapsulate tra due scriptlet.

LISTENER

L'insieme dei componenti ospitati all'interno di un contenitore JavaEE è completamente gestito da questo: ne governa il ciclo di vita, ne definisce le politiche di attivazione, controlla l'uso delle risorse e della concorrenza. Il contenitore offre un insieme di meccanismi volti a permettere ad un'applicazione di reagire opportunamente ai cambiamenti degli stati dei singoli componenti. I listener permettono inizializzazione e distruzione del WebContext, modificare attributi del WebContext, creazione, modifica e distruzione di sessioni, ricezione e modifica dei parametri di una richiesta da parte del contenitore.

Si dichiara un ascoltatore degli eventi legati ad un contesto web attraverso l'annotazione @WebListener: nel momento in cui viene deployata l'applicazione, tutte le sue classi sono scandagliate alla ricerca di eventuali ascoltatori (creando un'istanza di tali classi e associando la tipo di evento che essi dichiarano di ascoltare) e quando si verifica un evento del tipo richiesto si invoca il metodo opportuno (nel contesto del thread in cui si è verificato l'evento.

IL PARADIGMA MVC

Le applicazioni web, lato server, sono costruite mediante un approccio stratificato. Ciascuno stato interagisce unicamente con gli strati adiacenti e limita le dipendenze reciproche definendo i punti di contatto sotto forma di interfacce. Questo approccio enfatizza la modularità della soluzione ed il principio della separazione delle responsabilità.

Il Data Access Layer si occupa del trasferimento delle singole entità da e verso la sorgente dei dati: offre metodi per il caricamento, il salvataggio, la cancellazione e la modifica di singoli oggetti referenziati attraverso un identificatore univoco (ID) e può anche offrire metodi per cercare oggetti sulla base del loro contenuto e per riperire tutte le entità connesse tramite una certa relazione ad un'altra entità data. Spesso è costituito da un insieme di oggetti DAO (Data Access Object) che possono appoggiarsi ad un ORM (Object Relational Mapping) o implementare le proprie logiche direttamente.

Il Domain Layer rappresenta il modello logico dei dati cui deve essere applicata la logica di business dell'applicazione: questo costituisce un'astrazione dei concetti e delle relazioni tra essi esistenti all'interno del dominio applicativo, e si appoggia allo strato di accesso dati per reperire/rendere persistenti le informazioni che lo alimentano.

Il Service Layer definisce il contratto tra l'utente e l'applicazione: offre metodi che corrispondono alle azioni logiche che l'utente può svolgere con l'applicazione. Inoltre, coordina le attività e delega i vari compiti agli oggetti del dominio applicativo: il suo stato riflette la progressione dei compiti da svolgere, piuttosto che lo stato di ciascun compito.

Il Presentation Layer ha sostanzialmente due compiti principali: trasformare le richieste provenienti dall'utente in operazioni sullo strato di servizio e incapsulare i risultati all'interno delle risposte che dovranno essere visualizzate sul dispositivo dell'utente. Data la natura multi-user delle applicazioni web, questo strato può interagire con il meccanismo delle sessioni.

La sequenza degli eventi non è nota a priori: i comandi arrivano in successione nel tempo e il programma reagisce agli eventi esterni adottando un opportuno comportamento in base alla storia passata. Quindi, occorre "dare un senso" a questa sequenza interpretando ogni evento nell'ottica della funzione che il programma intende svolgere: bisogna, ad esempio, distinguere i dati inseriti dai comandi, verificare correttezza e congruenza dei dati, eseguire comandi ecc..
Per evitare che i programmi diventino rapidamente illeggibili, occorre separarli in un insieme di componenti ed uno degli approcci più adottati è quello detto Model-View-Controller.
La vista è una rappresentazione visuale del modello. Il controllore intercetta le richieste entranti, estrae i parametri, invoca lo strato di servizio e sceglie la vista da presentare, che verrà popolata con i dati ottenuti dal modello.

MODELLO

Un'applicazione è uno strumento informatico che consente ad un utente di elaborare informazioni: esse devono essere contenute e sono soggette ad un insieme di vincoli logici. Per gestire tali informazioni si utilizza un apposito oggetto che viene detto "modello". Esso non si occupa affatto dell'interfaccia ma solo

dei dati che vengono incapsulati nei suoi attributi su cui l'applicazione opera. Può essere acceduto da uno strato di servizio. E' realizzato tramite un inseme di oggetti Java memorizzati all'interno della sessione o dell'applicazione, o resi persistenti in file o in una base dati.
Il modello deve essere il punto di partenza nello sviluppo di un progetto.

VISTA

Una vista è un oggetto specializzato nel mostrare quanto contenuto nel modello. La vista non modifica in alcun modo il modello e, a parità di modello, possono esistere viste diverse, ciascuna dedicata a mostrarne una parte o a presentare le informazioni in modo diverso. Le viste sono realizzate tramite "scheletri" HTML/XML/JSON

CONTROLLORE

Oggetto che mette in comunicazione le azioni eseguite sull'interfaccia offerta dalla vista con le operazioni offerte dal modello. Il controllore costituisce il nucleo dell'interazione ed è rappresentato da codice che identifica i parametri ricevuti, aggiorna il modello attraverso le interfacce di servizio e sceglie la vista opportuna

Per ogni tipo di richiesta viene definita una URL opportuna: qui viene mappato il controllore responsabile della sua gestione (spesso implementato come servlet) e per ogni vista viene preparata una opportuna implementazione. Per ogni URL supportata, occorre definire un controllore con i relativi metodi.