Questo capitolo introduce le applicazioni web, che solitamente utilizzano le tecnologie JavaServer Faces e/o Java Servlet.

Un’applicazione web è un’estensione dinamica di un server web o di un’applicazione.

Le applicazioni web sono dei seguenti tipi:

* **Orientate alla presentazione:** genera pagine web interattive contenenti vari tipi di linguaggi di mark-up (HTML, XHTML, XML, etc) e contenuti dinamici in risposta a determinate richieste.
* **Orientate al servizio:** Implementano l’endpoint di un servizio web. Le applicazioni orientate alla presentazione sono solitamente client di questo tipo di applicazioni.

## 6.1 Applicazioni Web

Nella piattaforma Java EE i componenti web offrono l’estensione delle funzionalità dinamiche per il server web.

I componenti web possono essere Java Servlets, pagine web impementate tramite la tecnologia JavaServer Faces, endpoints di servizi web o pagine JSP.

La figura sottostante mostra l’interazione chec’è tra un web client e l’applicazione web che usa una servlet.



1 Il client invia una richiesta HTTP al server web.

Il server che implementa le tecnologie Java Servlet e JavaServer Pages converte la richiesta in un oggetto HTTPServletRequest.

2 Questo oggetto viene consegnato al componente web

3 che può interagire con i componenti JavaBeans

4 o il database per generare contenuti dinamici.

5 Il componente web può quindi generare un oggetto HTTPServletResponse o può girare la richiesta a un altro componente web.

6 Se crea l’oggetto HTTPServletResponse il server web lo converte in una risposta HTTP e lo ritorna al client.

Le servlet sono classi Java che processano dinamicamente le richieste e costruiscono le risposte.

Le tecnologie JavaServer Faces e Facelets sono usate per costruire applicazioni web interattive (possono essere usati anche framework a questo scopo).

Anche se le servlet e le JavaServer Faces e Facelets possono essere usate per fare cose simili ognuna è più performante per alcuni tipi di utilizzo.

Le servlet sono migliori per applicazioni orientate ai servizi e funzioni di controllo di un’applicazione orientata alla presentazione.

Le pagine JavaServer Faces e Facelets sono più appropriate per generare mark-up testuale, come XHTML, e sono solitamente usate per applicazioni orientate alla presentazione.

I componenti web sono supportati dal web container.

Un web container offre servizi quali gestione delle richieste, sicurezza, concurrency e gestione del ciclo vitale.

Un web container da inoltre modo ai componenti web di accedere a API quali naming, transazione e email.

Alcuni aspetti del comportamento di un’applicazione web possono essere configurati quando l’applicazione è installata (o importata) nel web container.

Le informazioni di configurazione possono essere specificate usando le annotazioni o essere manutenute in un file XML chiamato DD (che abbiamo visto nel documento precedente).

## 6.2 Ciclo di vita delle applicazioni web

Un’applicazione web è composta da componenti web, file di risorse statiche e classi di aiuto/librerie.

Il web container offre molti servizi di supporto che accrescono le capacità dei componenti web e li rende più semplici da sviluppare anche se, visto che un’applicazione web deve rispondere di questi servizi , il processo per creare e far girare un’applicazione web è diverso da quello delle classi Java standalone tradizionali e può essere indicizzato come segue:

1. Sviluppo del codice dei componenti web
2. Sviluppo del deployment descriptor dell’applicazione web (se necessario)
3. Compilazione dei componenti dell’applicazione web e delle classi di aiuto richiamate dai componenti
4. Facoltativamente, raggruppare l’applicazione in un’unità importabile
5. Importare l’applicazione nel web container
6. Accedere a un URL che richiama l’applicazione web

## ESEMPIO: hello1 (vedi esercizio hello1 su NetBeans)

1 Apriamo il file *index.xml* che è la pagina principale di default delle applicazioni Facelets.

In una tipica appicazione Facelets le pagine web sono create in XHTML.

In questa applicazione la pagina usa un semplice mark-up di tag per visualizzare un form con un’immagine, un header, un campo e due bottoni di comando.

2 Apriamo il file response.XHTML che contiene il codice della pagina di risposta.

3 Apriamo Hello.java che offre i metodi getter e setter.

Se si usa il nome di default per la classe bean si può usare l’annotazione *@Model* invece di specificare *@Named* e @*RequestScoped.*

L’annotazione *@Model*  è detta stereotype perché incapsula altre annotazioni.

4 Apriamo il web.xml che contiene alcuni elementi indispensabili per un’applicazione che utilizzi le Facelets.

Ciò che segue viene creato in automatico quando si usa NetBeans IDE per creare un’applicazione:

* Un parametro context che specifica l’allestimento del progetto.

*<context-param>*

*<param-name>javax.faces.PROJECT\_STAGE</param-name>*

*<param-value>Development</param-value>*

*</context-param>*

Il parametro di contesto offre le informazioni di configurazione necessarie per un’applicazione web.

Un’applicazione può definire i propri parametri di contesto.

Inoltre le tecnologie JavaServer Faces e Java Servlet definiscono i parametri di contesto che l’applicazione può utilizzare.

* Un elemento servlet e il proprio elemento mapping che specifica il FacesServlet.

Tutti i file con suffisso .xhtml saranno confrontati:

*<servlet>*

*<servlet-name>Faces Servlet</servlet-name>*

*<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>*

*<load-on-startup>1</load-on-startup>*

*</servlet>*

*<servlet-mapping>*

*<servlet-name>Faces Servlet</servlet-name>*

*<url-pattern>\*.xhtml</url-pattern>*

*</servlet-mapping>*

* Un elemento welcome-file-list che specifica la posizione della pagina di arrivo:

*<welcome-file-list>*

*<welcome-file>index.xhtml</welcome-file>*

*</welcome-file-list>*

## Introduzione agli Scopes

Nella classe Hello.java le annotazioni *@Named* e *@RequestScoped* identificano la classe come un managed bean che utilizza lo scope della richiesta. Lo Scope definisce la persistenza dei dati e la loro condivisione.

Gli scope più utilizzati nelle applicazioni JavaServer Faces sono i seguenti:

* **Request (***@RequestScoped***):** Persistenza limitata a una singola richiesta HTTP nell’applicazione web.
* **Session (***@SessionScoped***):** La persistenza si ha per richieste HTTP multiple nell’applicazione web e serve quindi quando un’applicazione deve rispondere a molte richieste mantenendo i dati.
* **Application (***@ApplicationScoped***):** La persistenza dei dati c’è per tutte le interazioni dell’utente con l’applicazione web.

## Packaging and Deploying the hello1 Web Module

Un modulo web deve essere impacchettato dentro un WAR in alcuni casi e se si vuole distribuire il modulo web. Si può impacchettare il modulo web in un file WAR usando indipendentemente sia Maven che IDE.

Il tutorial che segue mostra come costruire, impacchettare e distribuire l’applicazione di esempio hello1.

## Costruire e impacchettare il modulo web hello1 usando NetBeans

1 Far partire il GlassFish Server

2 Open Project

3 Fare build, questo comando esporta del progetto sul server

## Vedere i moduli web esportati

GlassFish Server offre due modi di vedere i moduli web esportati: l’Administration Console e il comando asadmin.

Si può inoltre usare Netbeans IDE.

## Con l’Administration Console

1 Aprire l’URL <http://localhost:4848/> nel browser

2 Selezionare il nodo dell’applicazione

I moduli appaiono nella tabella Deployed Applications

## Con il comando asadmin

Usare il comando *asadmin list-applications*

## Con NetBeans IDE

1 Nella Service tab espandere il nodo Servers e il nodo GlassFish Server.

1. Espandere il nodo Applications per vedere i moduli esportati

## 6.3.4 Far girare il modulo hello1 esportato

Inserire nel browser il seguente URL: [*http://localhost:8080/hello1/*](http://localhost:8080/hello1/)

## 6.3.4.1 Dynamic Reloading of Deployed Modules

Se il dynamic reloading è abilitato non c’è bisogno di ri-esportare l’applicazione o il modulo quando si effettuano modifiche al codice o ai descrittori ma basta copiare le classi o pagine modificate nella directory di esportazione dell’applicazione o modulo.

La deployment directory di un modulo web denominato *context-root*  sarà *domain-dir/applications/context-root*.

Il server controlla periodicamente se ci sono modifiche e effettua il redeploy dell’applicazione in maniera automatica e dinamica.

Questa funzionalità è utile in un ambiente di sviluppo perché permette di testare velocemente le modifiche al codice.

Il reload dinamico non è raccomandato invece per l’ambiente di produzione perché può abbassare le performance e, quando si effettua il reload, la sessione viene invalidata e il client deve ricominciarla.

Nel server GlassFish il reload dinamico è abilitato di default.

## Undeploying del modulo web hello1

Si può effettuare l’undeploying di un modulo web e di altri tipi di applicazioni enterprise usando NetBeans IDE o Maven.

## Usando NetBeans IDE

1 Nel Services tab espandere il nodo Servers e quello del GlassFish Server.

2 Espandere il nodo Applications.

1. Tasto dx sul modulo hello1 e selezionare **Undeploy**.
2. Tornare sulla Projects tab e selezionare **clean**  dopo aver schiacciato il tasto dx sul progetto

## Usando Maven

1 Nella finestra del terminale andare su:

*tut-install/examples/web/jsf/hello1/*

2 Inserire il comando *mvn cargo:undeploy*

Inserire il comando *mvn clean*

## Un modulo Web che utilizza la tecnologia delle Java Servlet: l’esempio hello2

L’applicazione hello2 è un modulo web che utilizza la tecnologia Java Servlet per mostrare il greeting e la risposta.

## 6.4.1 Mappare gli URL’s su un componente web

Quando riceve una richiesta il web container deve determinare quale componente web debba gestirla, fa ciò mappando il percorso URL contenuto nella richiesta a un componente web o un’applicazione.

L’URL path contiene la context root e, facoltativamente, un URL pattern:

[*http://host:port/context-root[/url-pattern*](http://host:port/context-root%5b/url-pattern)*]*

URL pattern viene settato tramite l’annotazione *@WebServlet* nel file sorgente della servlet.

Ad esempio il file *GreetingServlet.java* dell’applicazione hello2 contiene le seguenti annotazioni che specificano l’URL pattern come */greeting:*

*@WebServlet("/greeting")*

*public class GreetingServlet extends HttpServlet {*

Tale annotazione indica che l’URL pattern */greeting* segue il context root e quindi si può accedere al server esportato localmente tramite il seguente URL:

*http://localhost:8080/hello2/greeting*

Per accedere al server usando unicamente la context root si deve specificare “/” come URL pattern.

## 6.4.2 Disamina del modulo web hello2

L’applicazione hello2 è praticamente identica a hello1 ma è implementata tramite la tecnologia delle Java Servlet invece di quella delle JavaServer Faces.

1 Ora andiamo a prendere in considerazione il file GreetingServlet.java:

Questa servlet effettua l’override del metodo *doGet* implementando il metodo GET di HTTP.

La servlet mostra una semplice greeting form in cui il bottone *submit,* come in hello1, specifica una pagina di risposta per l’azione di inserimento.

2 Andiamo ad esaminare il file *ResponseServlet:*

Anche questa servlet effettua l’override del metodo *doGet* mostrando solo la risposta.

## 6.4.3 Far partire l’esempio hello2

## 6.4.3.1 Usando NetBeans IDE

1 Aprire il progetto e, col tasto dx, selezionare l’opzione **build** per impacchettare e esportare il progetto

2 Aprire in seguente URL nel browser [*http://localhost:8080/hello2/greeting*](http://localhost:8080/hello2/greeting) che specifica la context root seguita dall’URL pattern.

L’applicazione è uguale a hello1 con la differenza che clicckando su submit la risposta appare nella stessa pagina al posto che in una separata.

## 6.4.3.2 Usando Maven

1 Nella finestra del terminale andare su: *tut-install/examples/web/servlet/hello2/*

2 Inserire il comando *mvn install* che crea il file WAR nella directory *tut-install/examples/web/hello2/target/* e lo esporta

3 Aprire nel browser l’URL [*http://localhost:8080/hello2/greeting*](http://localhost:8080/hello2/greeting)che specifica il context root seguito dall’URL pattern.

## Configurare applicazioni web

Questo capitolo descrive i compiti da svolgere per configurare le applicazioni web:

* Settare i parametri di contesto
* Dichiarare i welcome files
* Mappare gli errori su error screens
* Dichiarare le referenze alle risorse

## 6.5.1 Setting Context Parameters

I componenti web in un modulo web condividono un oggetto che rappresenta il loro contesto di applicazione (application context).Si possono fornire parametri di contesto al context o parametri di inizializzazione a una servlet.

I parametri di contesto sono disponibili per tutta l’applicazione, quelli di inizializzazione sono illustrati nel capitolo 17.4.

## 6.5.1.1 Aggiungere un parametro di contesto usando NetBeans IDE

Questi passaggi vengono generalmente applicati alle applicazioni web anche se non li useremo negli esempi di questo capitolo.

1 Aprire il progetto

2 Aprire il **web.xml** dal WEB-INF (se il progetto non lo ha crearne uno)

1. Cliccare su **General** nella finestra editor

4 Espandere il nodo **Context Parameters** e cliccare su **Add**

1. Nel box di dialogo di **Add Context Parameters** inserire il nome dell’oggetto context (**Parameter Name**)
2. Nel campo **Parameter Value** inserire i parametri da passare all’oggetto context
3. Cliccare su **ok**

## 6.5.1.2 Creare un file web.xml con NetBeans IDE

1 Scegliere New File dal menu del file.

2 Nello wizard del nuovo file selezionare la categoria **Web** e poi **Standard Deployment**

**Descriptor** sotto **File Types.**

3 Cliccare **next** e **finish.**

Il web.xml apparirà in *web/WEB-INF/.*

## 6.5.2 Dichiarare i welcome files

Il meccanismo dei welcome files permette di specificare una lista di files che il web container può agganciare a una richiesta per un URL (chiamata richiesta parziale valida) che non è mappata a un componente web.

Ad esempio supponiamo di aver definito come welcome file chiamato *welcome.HTML.*

Se un cliente richiede un URL tipo *host:port/webapp/directory* in cui *directory* non è mappata su una servlet o una pagina XHTML, il welcome file sarà ritornato al client che vedrà in risposta la pagina *welcome.HTML*.

Se un container web riceve una valid partial request il web container esamina la lista di welcome file, allega alla richiesta ogni welcome file nell’ordine specificato e controlla se una risorsa statica o una servlet nel WAR è mappata all’URL di tale richiesta, poi invia la prima risorsa coincidente contenuta nel WAR.

Se non è specificato nessun welcome file il server GlassFish userà il file nominato *index.html* come welcome file di default. Se non esiste neanche tale file ritornerà una directory listing.

Il welcome file va specificato nel web.xml, ad esempio per l’esempio hello1 la specifica è la seguente:

*<welcome-file-list>*

*<welcome-file>index.xhtml</welcome-file>*

*</welcome-file-list>*

Un welcome file specificato non deve avere uno slash né iniziale né finale.

L’esempio hello2 non specifica nessun welcome file perché la richiesta URL è mappata nel componente web *GreetingServlet* tramite l’URL pattern */greeting*.

## 6.5.3 Mapping Errors to Error Screens

Quando capita un errore durante l’esecuzione di un’applicazione web si può fare in modo che l’applicazione mostri una specifica error screen relativa alla tipologia di errore.

In particolare si può specificare un mapping tra lo status code ritornato dalla risposta a una richiesta HTTP o a un’eccezione Java ritornata da un qualunque componente web e un qualunque tipo di error screen.

Si possono avere più elementi *error page* nel deployment descriptor, ogni elemento identifica una differente tipologia di errore che può causare l’apertura di una pagina di errore.

Questa pagina di errore può essere la stessa per qualunque numero di elementi *error page*.

## 6.5.3.1 To Set Up Error Mapping Using NetBeans IDE

Questi passaggi sono usati in genere nelle applicazioni web anche se nn saranno applicati negli esempi di questo capitolo.

1. Aprire il progetto e aprire il web.xml (se non esistesse crearne uno)
2. Cliccare **Pages** nella finestra di editor, espandere il nodo **Error Pages**  e cliccare **Add**
3. Cliccare **Browse** per trovare la pagina di che si vuole rendere una Error Page
4. Specificare un codice di errore o un tipo di eccezione
   * Per specificare un codice di errore inserire lo status code HTTP che dovrebbe causare l’apertura della pagina di errore o lasciare il campo vuoto per includere tutti i codici di errore
   * Per specificare un tipo di eccezione inserire nel campo **Exception Type** l’eccezione che può causare il caricamento della pagina di errore. Per specificare gli errori throwable e le eccezioni usare *java.lang.Throwable*.
5. Cliccare **OK**

## 6.5.4 Dichiarare Resource References

Se un componente utilizza oggetti quali enterprise beans, data sources o servizi web, si utilizzano le annotazioni per iniettare tali risorse nell’applicazione.

Le annotazioni eliminano gran parte del codice boilerplatee degli elementi di configurazioni che erano necessari nelle versioni precedenti di Java EE.

Anche se l’iniezione delle risorse tramite le annotazioni possono essere convenienti per lo sviluppatore, ci sono alcune limitazioni nel loro utilizzo per le applicazioni web.

Per prima cosa le iniezioni sono possibili solo su oggetti gestiti dal container dato che lo stesso deve avere il controllo sulla creazione del componente per poter effettuarvi l’iniezione, quindi non è possibile utilizzare le iniezioni su oggetti tipo i componenti JavaBeans.

Tuttavia i managed beans sono gestiti dal container quindi accettano l’iniezione delle risorse.

I componenti che possono accettare l’iniezione delle risorse sono i seguenti:



Questo capitolo spiega come usare alcune annotazioni supportate dal web container per iniettare le risorse.

## 6.5.4.1 Dichiarare il riferimento a una risorsa

L’annotazione *@Resource* è utilizzata per dichiarare il riferimento a una risorsa, come una fonte dati, un enterprise bean o le variabili d’ambiente.

Questa annotazione è specificata su una classe, un metodo o un campo.

Il container gestisce l’iniezione dei riferimenti alle risorse dichiarate dall’annotazione *@Resource* e le mappa nelle risorse JNDI appropriate.

Nell’esempio seguente l’annotazione *@Resource* è utilizzata per iniettare una fonte dati in un componente che necessita di avere la connessione alla fonte dati (dataSource), che è fornita utilizzando la tecnologia JDBC per accedere a un database relazionale:

*@Resource javax.sql.DataSource catalogDS;*

*public getProductsByCategory() {*

*// get a connection and execute the query*

*Connection conn = catalogDS.getConnection();*

*...*

*}*

Il container inietta questa fonte dati prima sui componenti resi disponibili all’applicazione. Il mapping JNDi della fonte dati p dedotto dal nome del campo *catalogDS* e dal tipo *javax.sql.DataSource*.

Se si hanno più risorse che necessitano di essere iniettate in un componente, bisogna usare l’annotazione *@Resources* per contenerle come mostrato sotto:

*@Resources ({*

*@Resource(name="myDB" type=javax.sql.DataSource.class),*

*@Resource(name="myMQ" type=javax.jms.ConnectionFactory.class)*

*})*

L’esempio di applicazione web di questo tutorial utilizza la Java Persistence API per accedere a database relazionali. Questa API non necessita di creare esplicitamente una connessione alla fonte dati, perciò gli esempi non utilizzano l’annotazione *@Resource* per iniettarla.

In ogni caso questa API supporta le annotazione *@PersistenceUnit* e *@PersistenceContext* per iniettare istanze *EntityManagerFactory* e *EntityManager* .

## 6.5.4.2 Dichiarare il riferimento a un servizio web

L’annotazione *@WebServiceRef* offre un riferimento a un servizio web.

Il seguente esempio mostra l’utilizzo di tale annotazione per dichiarare un riferimento a un servizio web.

*WebServiceRef* utilizza l’elemento *wsdlLocation* per specificare l’URI del file di servizio WSDL esportato.

*...*

*import javax.xml.ws.WebServiceRef;*

*...*

*public class ResponseServlet extends HTTPServlet {*

*@WebServiceRef(wsdlLocation="http://localhost:8080/helloservice/hello?wsdl")*

*static HelloService service;*