Jakarta EE

Leonardo De Boni

December 7, 2022

CONTENTS

1	Introduzione 2		
	1.1 Ottenere un indirizzo	ip 2	
	1.2 Connettersi al server	2	
2	Servlet 2		
	2.1 Servlet annotations	2	
3	Struttura di un progetto w	eb 3	
4	Resources 3		
5	Filters 3		
	5.1 Filter annotations	3	
6	JSP (Jakarta Server Pages) 3		
	6.1 Expression Language	4	
	6.2 Direttive 4		
	6.3 Tag Library 4		
7	Sessione 5		
	7.1 Session e Filter	;	
8	Pattern MVC (Model View Controller)		
	8.1 Struttura MVC	•	
9	REST API 7		
10	JSON-B 7		
	10.1 Annotations 7		
11	JAX-RS 7		
	11.1 Mappatura dei metod	li 7	
	11.2 Mappatura dei paran	netri 8	
	11.3 Path come parametro	8	

INTRODUZIONE

Il package che ci interessa é: java.net. Le classi che ci interessano sono:

- InetAddress: rappresenta un indirizzo ip
- Socket: permette di creare un client
- ServerSocket: permette di creare un Server

Ottenere un indirizzo ip

Per ottenere l'indirizzo ip a partire da un nome possiamo fare cosí: InetAddress address = InetAddress.getByName(String hostName);

1.2 Connettersi al server

Per connettersi a un server dopo aver ottenuto l'indirizzo possiamo creare un client cosí:

```
Socket socket = new Socket(InetAddress adress, int port)
```

Dal socket possiamo ottenere un InputStream:

InputStream in = socket.getInputStream();

SERVLET

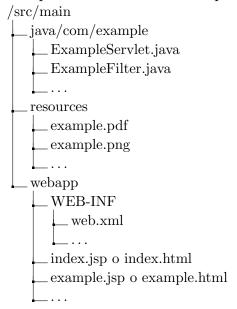
Classe che mi permette di elaborare una richiesta. Per crearne uno devo estendere la classe HttpServlet. I metodi che posso sovrascrivere che corrispondono alle varie richieste sono i seguenti:

- doGet (HttpServletRequest req, HttpServletResponse res)
- doPost (HttpServletRequest req, HttpServletResponse res)
- doHead (HttpServletRequest req, HttpServletResponse res)
- doOptions (HttpServletRequest req, HttpServletResponse res)
- doTrace (HttpServletRequest req, HttpServletResponse res)
- doPut (HttpServletRequest req, HttpServletResponse res)
- doDelete (HttpServletRequest req, HttpServletResponse res)

2.1 Servlet annotations

STRUTTURA DI UN PROGETTO WEB 3

Questa puó essere la struttura semplificata di un progetto JavaEE:



- **RESOURCES**
- **FILTERS**
- 5.1 Filter annotations
- JSP (JAKARTA SERVER PAGES)

JSP mi permette di scrivere codice java all'interno di HTML. Per fare ciò posso usare delimitatori dedicati:

- Scriptlets: <% ... istruzioni java ... %>
- Expressions: <% = ... espressioni java ... %> sará mostrato al client
- **Declarations**: <%! ... espressioni java ... %> posso dichiarare metodi

Nelle pagine JSP ho a disposizione vari oggetti impliciti:

- request HttpServletRequest
- response HttpServletResponse
- out OutputStream (response.getOutputStream())
- page this
- exception Throwable
- config ServletConfig

6.1 Expression Language

```
Permette di utilizzare expressions piú snelle:
   ${ expression }
Quindi per esempio
   <% = request.getParameter("username") %>
diventa:
   ${ param.username }
Inoltre ho a disposizione:
   TODO
```

6.2 Direttive

Le direttive sono indicazioni che influenzano sulla struttura dell'oggetto:

```
<% @page specifiche %>
<% @include risorsa da includere %>
« @taaglib libreria di tag speciali % >
```

La direttiva @page mi permette di impostare delle specifiche, estendere servlet o importare pacchetti:

```
<% @page
   extends="servlet"
   import="package"
   isErrorPage="true o false"
   errorPage="errorPage.jsp" %>
```

Con la direttiva @include posso includere una risorsa. L'utilizzo piú comune é per includere header e footer che rimangono uguali nella maggior parte delle pagine di un sito. Utilizzo:

```
<% @include file="/path/to/file" %>
```

La direttiva @taaglib mi permette di includere librerie di tag come per esempio jstl. Utilizzo di esempio:

```
<% @taaglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

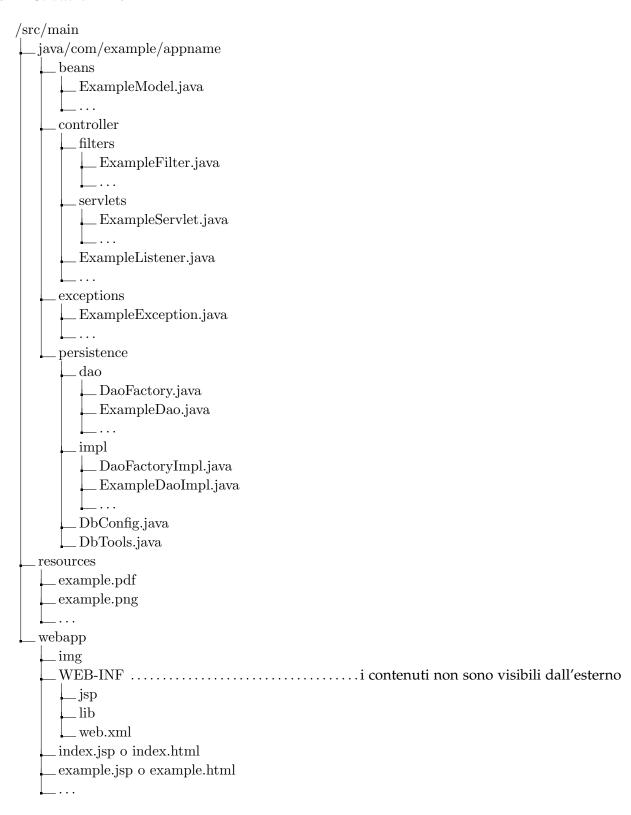
6.3 Tag Library

```
L'equivalente dell'if é:
<c:if test="${expression}" var="nome">
   ${nome}
</c:if>
Se volessi creare una catena if/else if/else posso usare:
TODO
Per creare un ciclo for posso fare:
<c:forEach var="i" begin="1" end="5>
   Punto numero ${i}
</c:forEach>
Oppure posso iterare su una collection:
<c:forEach items="${collection}" var="item">
```

```
<li>${item.property} </li></c:forEach>
```

- 7 SESSIONE
- 7.1 Session e Filter
- 8 PATTERN MVC (MODEL VIEW CONTROLLER)

8.1 Struttura MVC



Q REST API

10 JS0N-B

Questa libreria mi permette di effettuare conversioni immediate da JSON a un Oggetto con proprietà. Per prima cosa devo ottenere il builder:

```
Jsonb builder = JsonbBuilder.create();
```

Successivamente se devo fare una serializzazione (Object→JSON):

String json = builder.toJson(Object object);

Invece se devo fare una deserializzazione (JSON→Object):

Object object = builder.fromJson(String json, Object.class);

10.1 Annotations

All'interno dei POJO posso usare annotations per cambiare il comportamento del builder durante le conversioni. A disposizione ho:

- @JsonbProperty("property-name") Applicato a un getter posso cambiare il nome di una propietà nel JSON
- @JsonbTransient Ignora la proprietà

11 IAX-RS

Per impostare il path della mia Application devo prima creare una classe che estende

jakarta.ws.rs.core.Application

e poi utilizzare l'annotation:

@ApplicationPath("path")

Quindi tutti gli endpoint della mia API saranno mappati sotto il path che ho appena impostato. Per impostare il path di un endpoint posso usare l'annotation:

@Path("path")

11.1 Mappatura dei metodi

Se volessi che un metodo di un Servlet risponda a un certo metodo HTTP posso anteporre l'annotation corrispondente:

endpoint	metodi	altro
@POST	CREATE	@PATCH
@GET	READ	@HEAD
@PUT	UPDATE	@OPTIONS
@DELETE	DELETE	

Inoltre posso mappare grazie alle annotation i parametri passati tramite url al parametro di un metodo Java. Esempio:

```
url: http://example.com/appname/api/endpoint?parameter=example
Posso mappare il parametro a un attributo del mio metodo che risponde a GET:
   public Object method(@QueryParam("parameter") String string)
Se ripondo a un POST (Form) l'utilizzo é uguale ma il nome dell'annotation é:
    @FormParam
```

11.3 Path come parametro

```
Per utilizzare il path come parametro si possono usare le seguenti annotation:

@Path("{parameter}")

@Path("{parameter}/endpoint")

public Object method(@PathParam("parameter") String string)

Possiamo inoltre assegnare a un parametro un valore di default usando l'annotation:

@DefaultValue
```