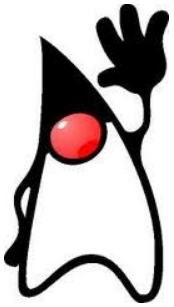
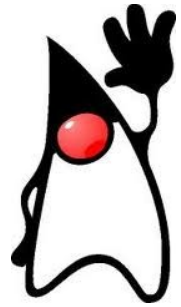


Spring MVC

e abbuyddA Beta

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Lo stack tecnologico di Abbuydda

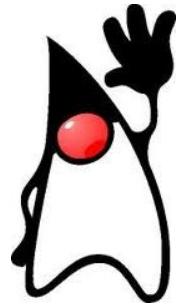


- Spring 2.5.6
  - Spring MVC
  - Spring Security 2.0.4
  - Spring JDBC
  - Spring Modules
- EhCache
- Paypal
- iText
- Servlet Container: Apache Tomcat 6
- DBMS: MySQL
- Direct Web Remoting
- JSP, Jquery

- IDE: Eclipse

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

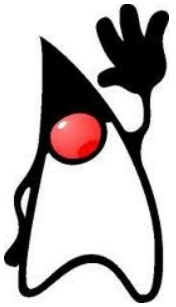
Cosa vediamo oggi?



- iniezione di dipendenza
- emme-vu-cì: ma che vuol dì?
- implementazione di un semplice controller
- gestire la logica di navigazione
- creare un vista
- impostare una form:
  - binding dei dati...
  - …e validazione (e mostrare errori, se ce ne sono!)

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Spring e Dependency Injection



La **dependency injection** (DI) nella programmazione orientata agli oggetti è un modello di progettazione (*pattern*) il cui principio fondamentale si basa sulla separazione tra le implementazioni concrete degli oggetti e la risoluzione delle dipendenze.
La dependency injection è una forma specifica di "**inversione di controllo**" in cui ciò che viene invertito è il processo per ottenere la necessaria dipendenza (espressa solo in termini di interfacce). Il termine è stato coniato da Martin Fowler.

In altre parole: *è una tecnica per disaccoppiare i componenti software tra loro dipendenti*.

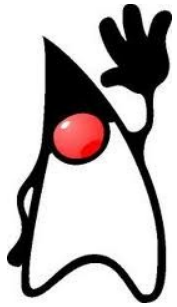
La dependency injection, invece di codificare le dipendenze direttamente nel programma, affida ad una terza parte il compito di istanziare i servizi necessari (in un contenitore che rappresenta il *contesto dell'applicazione*) e soddisfare le dipendenze.
Tutto ciò, in Spring, avviene attraverso uno o più *file di configurazione:*

```
<bean id="storeDao" class="com.prossimaisola.abbuydda.dao.jdbc.JdbcStoreDao">
        <property name="dataSource" ref="dataSource" />
</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
</bean>
```

Questa è una implementazione concreta ma chi usa il Dao lo fa attraverso l'interfaccia “StoreDao”

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Modello-Vista-Controllore



Il Paradigma MVC separa nettamente la logica di **business** di un'applicazione, da quella di **navigazione** e di **presentazione**.

Il Controller

1. gestisce la logica di navigazione ed interagisce con lo strato dei servizi per comandare la logica di business.

Il Modello

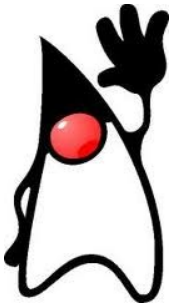
1. rappresenta il contatto tra il Controllore e la Vista;
2. contiene i dati da visualizzare nella Vista;
3. è popolato dal Controllore.

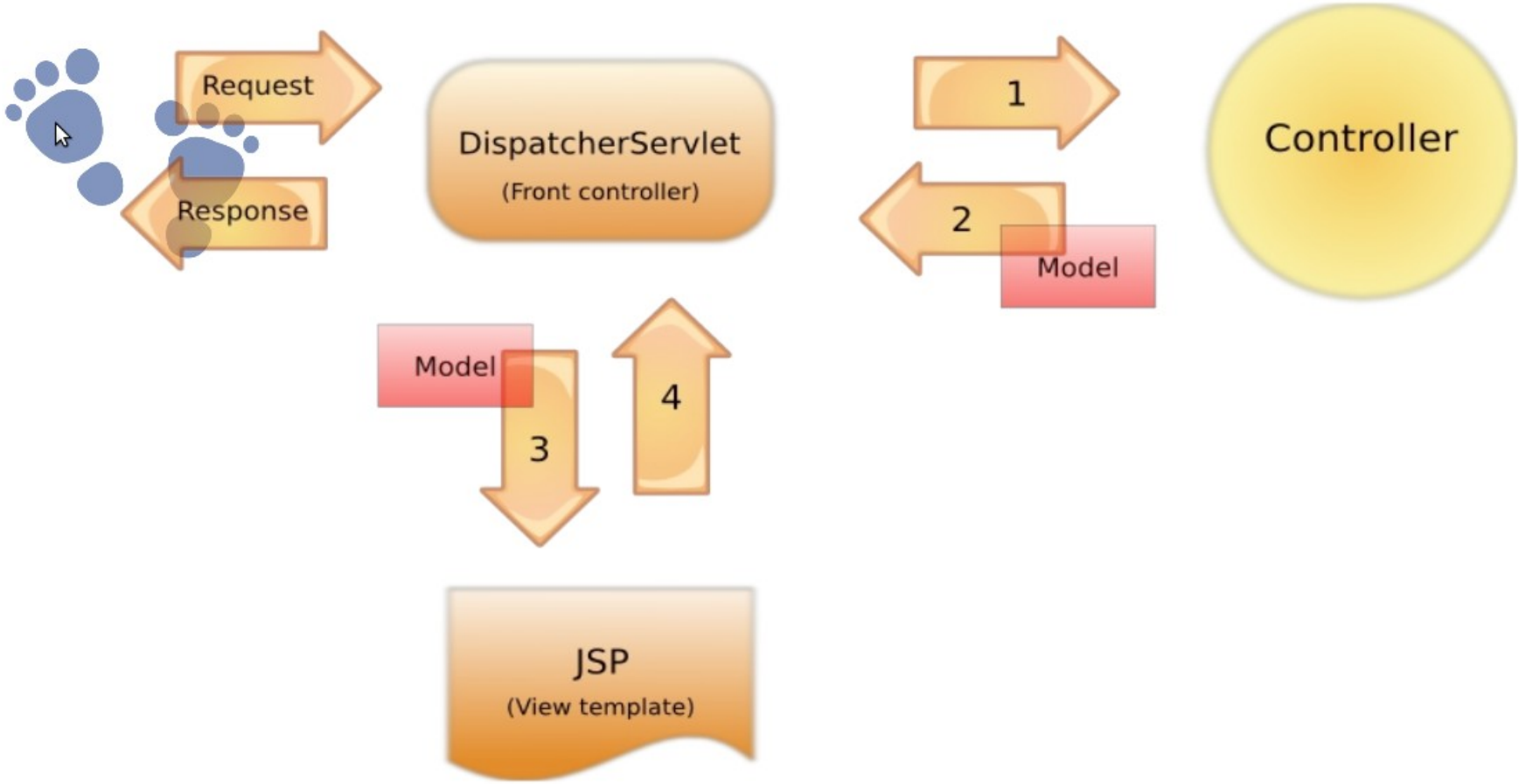
La Vista

4. estrae i dati dal modello;
5. è il contenuto da mostrare all'utente.

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

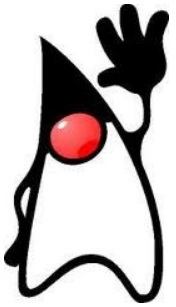
MVC in azione





Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

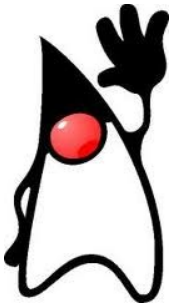
diciamolo al web.xml



```
<!-- this is part of the web.xml file -->
<servlet>
    <description>Spring MVC Dispatcher Servlet</description>
    <servlet-name>abbuydda</servlet-name>
    <servlet-class>org.springframework.web.
                servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>abbuydda</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```


Ivan Ricotti - eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Ecco a voi il signor Controller

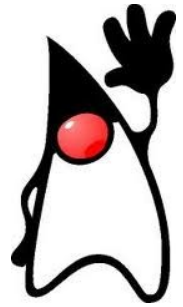


```
@Controller
@RequestMapping({"/login.html"})
public class LoginController {

   @RequestMapping
   public void index() {
      // do something useful
   }
}
```

I'm just a POJO!

Ivan Ricotti - eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

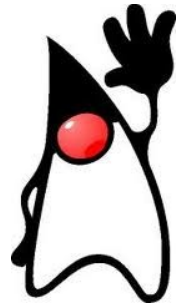
Configurazione del Contesto abbuydda-servlet.xml



```
<!-- Configures the @Controller programming model -->
<context:annotation-config />

<!-- Scan for controllers (and services) -->
<context:component-scan
     base-package="com.prossimaisola.abbuydda.web.controllers" />

<bean class="org.springframework....DefaultAnnotationHandlerMapping">
    <property name="order" value="1" />
</bean>

<!-- Views -->
<bean id="viewResolver"
       class="org.springframework....InternalResourceViewResolver">
    <property name="viewClass"
       value="org.springframework.web.servlet.view.JstlView">
    </property>
    <property name="prefix" value="/jsp/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Un giorno nei panni di un Controller...



```
@Controller
public class HomePageController {

    @Autowired
    private MessageService messageService;
    public void setMessageService(MessageService messageService) {
        this.messageService = messageService;
    }

    @RequestMapping("/index.html")
    public String homeHandler(ModelMap model) {
        User loggedUser = SiteUtils.getLoggedUser();
        model.addAttribute("mailbox",
          this.messageService.getUserMailBox(loggedUser);
        return "index";
    }

…
}
```

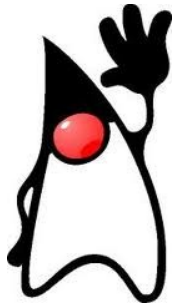
firma variabile!

Questo è il nome della vista da usare.
Anche il parametro di ritorno è “variabile”:

- Void (risoluzione di default)
- Stringa (path alla vista)
- Oggetto Vista (non necessariamente jsp)

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

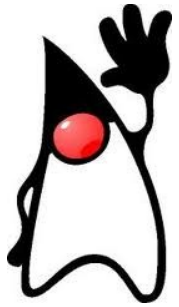
...e della Vista



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


    <div id="mailbox" >
            <c:url var="inboxUrl" value="./user/inbox.html"></c:url>
            <a href="${inboxUrl}">
                <c:if test="${mailbox.news gt 0}">
                    <img src="./images/messages/mail-new.png" >
                    <strong>[<c:out value="${mailbox.news}" />]</strong>
                </c:if>
                <c:if test="${mailbox.news == 0}">
                    <img src="./images/messages/mail-closed.png" >
                    [<c:out value="${mailbox.news}" escapeXml="false"/>]
                </c:if>
            </a>
    </div>
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

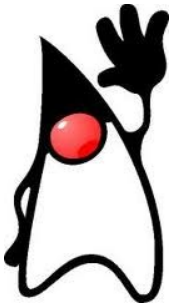
Uso del Modello (reprise)



```
@Controller
public class HomePageController {

    …

    @ModelAttribute("mailbox")
    public Mailbox getMailbox() {
        User loggedUser = SiteUtils.getLoggedUser();
        return this.messageService.getUserMailBox(loggedUser);
    }

}
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Recepire parametri... (1/2)



http://www.abbuydda.com/product/view.html?id=15

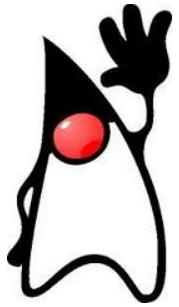
```
@Controller
public class ViewProductController {

    @Autowired
    private StoreService storeService;
    public void setStoreService(StoreService storeService) {
        this.storeService = storeService;
    }

    @RequestMapping(value="/product/view.html",
                    method=RequestMethod.GET)
    public String view(@RequestParam Integer id, ModelMap model) {
            this.storeService.getProduct(id);
            model.addAttribute("product", product);
            return "products/view";
    }

}
```

di default il parametro
si chiama come la variabile

Il framework mi passa ciò di cui ho bisogno!

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Recepire parametri... (2/2)

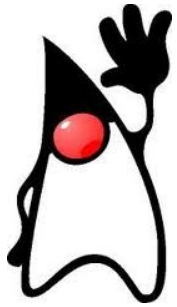


http://www.abbuydda.com/product/view/15

```
@Controller
public class ViewProductController {

    @Autowired
    private StoreService storeService;
    public void setStoreService(StoreService storeService) {
        this.storeService = storeService;
    }

    @RequestMapping(value="/product/view/{id}", method=RequestMethod.GET)
    public String get(@PathVariable Integer id, ModelMap model) {
        Product product = this.storeService.getProduct(id);
        model.addAttribute(product);
        return "products/view";
    }

    …

}
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Prepariamo una form

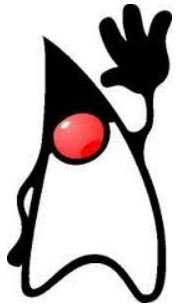


http://www.abbuydda.com/products/edit.html?productId=6

```
@Controller
public class ManageProductController {

    @RequestMapping(value="/products/edit.html",
                    method=RequestMethod.GET)
    public String get(@RequestParam(required=true) Integer productId,
                      ModelMap model) {
        Product product = this.storeService.getProduct(productUd);
        model.addAttribute(product);
        return "products/edit";

    …

}
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

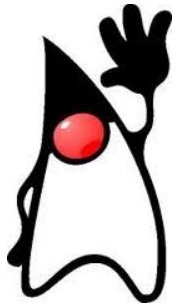
/products/edit.jsp



```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form modelAttribute="product" method="POST">

    <dl>
        <dt>Titolo</dt>
        <dd>
            <form:input path="title" />
            <form:errors path="title" cssClass="error" />
        </dd>
        <dt>Titolo</dt>
        <dd>
            <form:input path="description" />
            <form:errors path="description" cssClass="error" />
        </dd>
        ...
    </dl>

    <div class="actions">
        <input type="submit" name="add" value="Aggiungi" />
    </div>

</form:form>
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

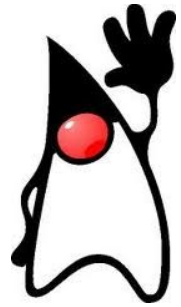
e ora processiamo la form!



```
@Controller
public class ManageProductController {

    …

    @RequestMapping(value="/products/edit.html",
                    method=RequestMethod.POST,
                    params="add")
    public String post(@Valid Product product, BindingResult result) {
        if (result.hasErrors()) {
            return "/products/edit.html";
        }
        this.storeService.saveProduct(product);
        return "redirect:/home";
    }
}
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Come solo @Valid?!?



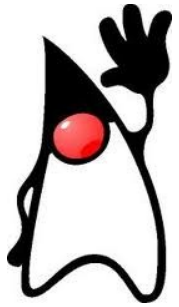
```
public class Product implements Serializable {

    @NotBlank
    @Size(max=80)
    String title;

    @NotBlank
    @Size(max=50)
    String description;

    @NotNull
    @Past
    Date publishDate;

    public Product() {
        this.publishDate = new Date();
    }

    ...
}
```

org.hibernate.validator.constraints

javax.validation.constraint

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

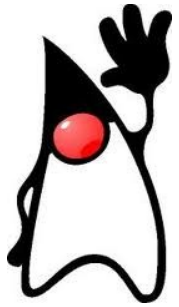
ma anche @NotNull non basta...



```
<!-- Spring 2.5.x + Spring Modules (va iniettato) -->
<bean id="configurationLoader"
      class="org.springmodules.validation.
             ...AnnotationBeanValidationConfigurationLoader"/>

<bean id="beanValidator"
     class="org.springmodules.validation.bean.BeanValidator">
    <qualifier value="beanValidator"
            type="org.springmodules.validation.bean.BeanValidator" />
    <property name="configurationLoader" ref="configurationLoader" />
</bean>


<!-- da Spring 3.0.x (e non va nemmeno iniettato!) -->
<mvc:annotation-driven validator="validator" />

<bean id="validator"
       class="org.springframework.validation.
              .beanvalidation.LocalValidatorFactoryBean" />
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

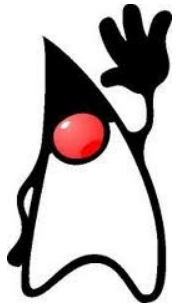
E per gli attributi complessi?



```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<form:form modelAttribute="product" method="POST">

    <dl>
        ...
        <dt>Data di Pubblicazione</dt>
        <dd>
            <form:input path="publishDate"  />
            <form:errors path="publishDate" cssClass="error" />
        </dd>
    </dl>

</form:form>
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

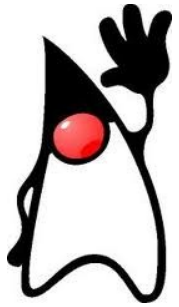
A loro ci pensano i PropertyEditor...



```
@Controller
public class ManageProductController {

    …

        @InitBinder
        public void initBinder(WebDataBinder binder) {
             binder.registerCustomEditor(Producer.class, "merchant",
                      new ProducerEditor(this.actorsService));
             binder.registerCustomEditor(Date.class, "publishDate"
                      new CustomDateEditor(DATE_FORMAT, false));
        }

    …
}
```


Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

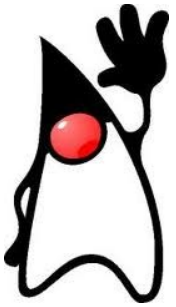
...oppure i nuovi ConversionService



```
<!-- Configures the @Controller programming model -->
<mvc:annotation-driven conversion-service="conversionService" />

<bean id="conversionService"
    class="org.springframework.format.support
                .FormattingConversionServiceFactoryBean">
    <property name="converters">
        <list>
            <bean class="com.prossimaisola.abbuydda.web.
                            converters.ProducerConverter" />
        </list>
    </property>
</bean>
```


Ivan Ricotti - eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

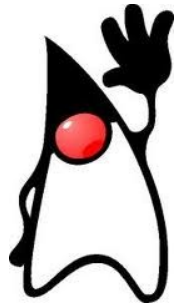
...a cui abbiamo dato un aiutino



```
public class ProducerConverter implements Converter<Integer, Producer> {

    @Autowired
    private ActorsService actorsService;
    public void setActorsService(ActorsService actorsService) {
        this.actorsService = actorsService;
    }

    @Override
    public Producer convert(Integer producerId) {
        return this.actorsService.getProducer(producerId);
    }

}
```

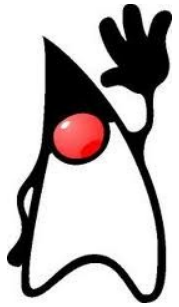

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Domande?



Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz

Riferimenti



- Spring documentation:
  - http://www.springsource.org/documentation
- Spring reference:
  - http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html
- Spring in Action:
  - http://www.manning.com/walls4/
- Eclipse J2EE e SpringIDE:
  - Eclipse J2EE - http://www.eclipse.org/downloads/moreinfo/jee.php
  - SprindIDE - http://dist.springframework.org/release/IDE
  - STS - http://www.springsource.com/developer/sts

Ivan Ricotti – eLabor sc
i.ricotti@elabor.biz
http://www.elabor.biz