1. Intro
2. Spring Moduli
3. Spring MVC AFU
4. Architettura Componenti
5. Ciclo vita richiesta http
   1. Componenti SMVC
6. Boot DispatcherServlet:
   1. Web.xml
   2. Web fragment
   3. ServletContainerInitializer (in classpath con impl metodo start
7. Configurazione Dispatcher e componenti
   1. Dichiarativa (xml)
   2. Classpath scanning
   3. Contesti (ereditano da org.springframework.web.WebApplicationInitializer
8. Componenti SMVC e defaults (ciclo vita)
   1. UrlMapper
   2. Controllers
   3. ViewResolvers
   4. @EnableMvc (supporto annotations)
9. HandlerMapping
   1. Gerarchia classi
   2. Dichiarativa
   3. @Configuration
10. Controllers: (Demo) Mapping
    1. Controller interface e sue estensioni
    2. ViewControllers (redirect to view)
    3. Gestione dichiarativa
    4. Classpath Scanning
    5. @Configuration context class
11. Mapping metodi
    1. Annotations supportate
    2. HttpRequest to java
    3. Java to HttpResponse
12. Gestione Forms
    1. Default Databindings (url expr -> obj.prop)
    2. Dati form e response
    3. @ModelAttribute: (tutorial)
       1. Ciclo di vita
       2. Read/write
       3. Dove definirlo
       4. Scope
       5. Demo: lista prodotti caricati
    4. @SessionAttribute:
       1. Ciclo di vita
       2. Read/write
       3. Dove definirlo
       4. Scope
       5. Demo Wizard registrazione
    5. Bean con scope per webapp:
       1. Session:Realizzare uno Shopping cart
       2. Application: liste statiche (categorie)caricate al boot
    6. File Upload

1. Controllers Data Bindings:
   1. Custom DataBindings (Global o Controller level):
      1. Validator (campi)
      2. Converter (date)
      3. Vari flag di ignore
   2. Converter
      1. PropertyEditor
      2. Converters
      3. Formatters
   3. Validator
      1. Validator
      2. Messages
      3. Errors
      4. JSR-303 Validation
2. CustomTags
   1. Demo: Controllers con molti dati annidati (ServiceBackend)
   2. Custom tag per display dati
3. Spring REST Services
   1. JAX-RS annotations
4. Spring MVC ed Ajax:
   1. Single Page Applications
5. View resolvers:
   1. Jsp
   2. Tiles
   3. Velocity
   4. Pdf
   5. Excel
   6. Xml
   7. Json
6. Testing:

**NOTE**

Global Customization

To customize data binding globally, we need to

1. create a class that implements the org.springframework.web.bind.support.WebBindingInitializer interface.

Spring MVC provides a configurable implementation of this interface, the org.springframework.web.bind.support.ConfigurableWebBindingInitializer.   
  
2) An instance of the interface must be registered with the handler mapping implementation, so that it can be used.

1. After an instance of org.springframework.web.bind.WebDataBinder is created, the initBinder method of the org.springframework.web.bind.support.WebBindingInitializer is called.

The provided implementation allows us to set a couple of properties.   
When a property is not set, it uses the defaults as specified by the org.springframework.web.bind.WebDataBinder.   
If we were to want to specify more properties, it would be quite easy to extend the default implementation and add the desired behavior.

Per Controller Customization

For the per controller option, we must

1. implement a method in the controller and put the

org.springframework.web.bind.annotation.InitBinder annotation on that method.   
  
The method must have no return value (void) and at least an org.springframework.web.bind.WebDataBinder as a method argument.   
  
The method *can* have the same arguments as a request-handling method.   
However, it *cannot* have a method argument with the org.springframework.web.bind.annotation.ModelAttribute

annotation.   
This is because the model is available after binding; and in this method, we are going to

configure how we bind.

The org.springframework.web.bind.annotation.InitBinder annotation has a single attribute

named value that can take the model attribute names or request parameter names this init-binder method is going to apply to. The default is to apply to all model attributes and request parameters.

1. To customize binding, we need to configure our org.springframework.web.bind.WebDataBinder.

**@ModelAttribute:**

Usato per popolare un model bean (legato al Controller) di dati per le views.  
Es. Valori defaults form, referenceData per output.  
E’ possibile inserirla a livello metodo o livello metodo.arg.

A livello metodo. (WRITE)  
Il metodo restituisce un oggetto, che verra’ reso disponibile, (insieme eventualmente ai suoi parametri) nel metodo che dovra gestire la richiesta.  
I dati vengono popolati prima di raggiungere il metodo handler.

A livello Metodo.Arg: (READ)  
 (viene chiesto l’accesso ad un attribute del model per questo tipo)  
Se non e’ definito un metodo dal quale recuperare un model bean con lo stesso nome, ne viene creato uno vuoto.

TESTS: @ModelAttribute

1. SETUP:
   1. View-button-> Controller -> View Display object (ctag base)
   2. Metodo annotato
   3. Arg annotato
2. Debug ordine chiamata metodi (no input -> Defaults)
3. Input modifica ModelAttribute -> model updated
4. Metodo void (Map in input) e visualizzazione map
5. Modifica @ModelAttribute arg in modo che non combaci con metodo trovato -> bean nuovo e campi vuoti

**@SessionAttribute:**Permette a quella variabile, una volta inizializzata di rimanere in sessione tra le N richieste stesso utente (solo) per quel controller.  
Il meccanismo sottostante e’ la session http, tuttavia:  
- La chiave non sara’ accessibile attraverso session.getAttribute(key)  
- L’attributo marcato con l annotation, avra’ una visibilita limitata al Controller in cui e’ definita.  
E’ possibile rimuovere l’attributo dalla session con il metodo.

Possibile collocarla:

1. A livello classe
2. A livello metodo

DETAILS:

**configurazione DispatcherServlet:**The dispatcher servlet needs a WebApplicationContext that should contain all the beans that enable

the dispatcher servlet to configure itself.   
E’ possibile configurare tale contesto con:

1. File xml specific (produrra un XmlWebApplicationContext)
2. Classpath Scanning (specifico in quali classpath cercare classi con annotations)
3. Caricare contesti da classi collegate attraverso il ServletContextInitializer (ed estensioni di Spring)

**Componenti da configurare: (esteso)**

1. Componenti DispatcherServlet e loro defaults: (dispensa)
2. Override con @EnableMvc (sovrascrive I defaults con quello che e’ definito in WebMvcConfigurationSupport

10 CONTROLLERS:

Come viene mappata Httprequest to Controller.metodo?

1. Impl di Controller unico metodo di ingress
2. Controllers con RequestBody annotation:
   1. Se @EnableMvc e’ä attivo l UrlMapping di default e’

Se @EnableMvc e’ attivato