**SMVC-310-Forms:**

**AFU**

1. Request lifecycle
2. Default Databindings (url expr -> obj.prop)
3. Dati form e response
4. @ModelAttribute: (tutorial)
   * 1. Ciclo di vita
     2. Read/write
     3. Dove definirlo
     4. Scope

**implicitModel:**

Nella invocazione da HandlerMethodInvoker.invokeHandlerMethod al generico metodo handler viene passata una Map chiamata implicitModel.

Se non e’ stata inizializzata da nessuna parte, questa Map e’ un contenitore vuoto passato al metodo handler per popolarlo di dati K, V.  
  
Prima di passare questo oggetto al handler l HandlerMethodInvoker nel metodo .invokeHandlerMethod verifica che se ci siano metodi annotati con @ModelAttribute.  
  
Se ci sono li esegue e li utilizza per popolare (la Map) dell implicitModel prima di passarlo all handler.

**BINDING DI UN DTO con form e handler**

E’ possibile creare un associazione tra un form ed un metodo oggetto in input del handler in maniera tale che SMVC si occupi di eseguire il binding dei dati in automatico.  
  
**<<prepare>>**form:form commandName=<KEY>

Utilizza se c’e’ un oggetto del MAV sotto chiave KEY. Se non esiste va in eccezioni di binding.  
  
Tale oggetto deve essere quindi creato nella fase di prepare del Controller.  
1) oggetto registrato nel MAV dall handler (**TC4**)  
2) Oggetto creato in un metodo con @ModelAttribute (default implicitModel) (TC3)

**<<handler>>  
POST dati da form (TC5)**

Per recuperare dati in request l handler puo’ prelevarli tramite args con: (invoke(handler, Object[] args)

1. Per tipo: User (T)
2. Model (e da li per chiave, facendo il match con command)
3. Marcando il metodo con @ModelAttribute(key) recuperandolo dal Model per quella chiave

Se c’e’ un metodo di init del command:

1. Viene fatto un merge tra il dto in binding e quello creato da init.  
   Se i campi sono null in binding, quelli dell init verranno utilizzati.
2. Ha piu senso questa collaborazione quando il @ModelAttribute ha uno scope Session (Vedi dopo)

**@ModelAttribute**Versione annotation delle operazioni di prima.  
  
 **<<prepare>>**  
Il @ModelAttribute a livello metodo, marca un metodo di init sul CommandObj che di fatto ottiene lo stesso che:  
MAV.add(key, Obj) precedentemente delegato al metodo <<prepare>>.

**<<handler>>**Il @ModelAttribute a livello arg del metodo handler permette di cercare nel Model l oggetto registrato sotto quella chiave.

**@ModelAttribute con key in @SessionAttribute**Il @ModelAttribute a livello metodo con una chiave definite nel @SessionAttribute (a livello classe) viene invocato un’ unica volta e inserisce il command Obj nello scope session.

**DEMO**

**UserController:**

**Display MAV: UserController.load  
  
Flow:  
demo01DisplayIndex.jsp > UserController /load>demo0DisplayResult.jsp**

**TC1: Caso standard**handler crea MAV  
dati MAV[key] accessibili injsp  
 **TC2: Risultato uguale a TC1 ma:**handler riceve in input un Model (implicitModel)  
modifica l implicitModel ricevuto (add K, V) aggiungendo dati.  
handler restituisce String  
-> MAV creato da String + implicitModel ()  
 **TC3: Obiettivo Risultato uguale a TC1 ma:l**oggetto creato ed aggiunto al Model in metodo con @ModelAttribute (prepare)

Verificare che:

1. Se l implicitModel viene intercettato sara’ accessibile (r/w) nell handler
2. Se l handler non lo intercetta (Object, Model, <T>) l oggetto rimarra’ coi valori di default impostati sul metodo <<prepare>>

**Demo Form processing**

**.save (save GET link)  
 (save POST form)  
  
Flow:  
demo01DisplayIndex.jsp > UserController /load>demo0DisplayResult.jsp**

1. Popolare oggetto nel MAV (new User), display in command in jsp.
2. Ricevere oggetto:
   1. Model (lettura su key)
   2. User (per tipo)
3. Interazione con @ModelAttribute:
   1. Prepare (scrive per key su Model)
   2. Handle (preleva per key dal Model)
   3. Verifica merge di prepare e handle da post form.
   4. Verifica che includendo key in @SessionAttribute:
      1. Il prepare viene eseguito solo una volta
      2. L oggetto viene condiviso tra le N chiamate
      3. La K, V e’ presente in sessione.

**TC4:**CommandObj creato nel handler   
disponibile in ProcessorIndex.jsp (Output)  
  
  
**TC5:**   
CommandObj creato nel metodo @ModelAttribute >   
disponibile (coi default in FormResult.jsp  
spedito tramite POST al metodo handler (processor)  
Ricevuto con @ModelAttribute(key)

**UserControllerWithValidators:**save con GET e POST.  
Validatore.

**TC6:**Validatore  
InitBinderMessageBundle (nel xml, bundle messages, bundle validation msgs)

**UserControllerWithSessions:**

**AFU:**interazione con Controller su piu steps.  
oggetto aggiunto al MAV tramite un metodo @ModelAttribute(KEY)

Verificare che

1. Se la (KEY) non e’ presente tra quelle elencate in @SessionAttribute (su classe) tra step2 e step3, lo step1 e’ perso. (**TC8**)
2. Se KEY si trova tra le vars in @SessionAttribute( su classe) (**TC9)**
   1. Il metdo @ModelAttribute viene chiamato solo una volta
   2. L Oggetto sopravvive alle N richieste verso questo (ed altri Controllers)
   3. E’ possibile rimuovere la chiave tramite metodo (altra demo)

**TC8:**

**IN:**metodo annotato con @ModelAttribute(key).  
NO @SessionAttribute (commentato).  
**OUT:**Nell ultima jsp I campo impostati sulla prima si sono persi e vengono sovrascritti da quelli nell initMethod

**TC9**:

**IN**:   
metodo annotato con @ModelAttribute(key).  
@SessionAttribute (key).

**OUT:**  
1)metodo annotato con @ModelAttribute(key) chiamato solo prima volta  
2) dati allo step1 non si perdono in step3  
3) chiave in nessun scope (UnexistingKey) va in eccezione

**ALTRE SPIEGAZIONI**

1. Before invoking the handler method, Spring invokes all the methods that have @ModelAttribute annotation. It adds the data returned by these methods to a temporary Map object. The data from this Map would be added to the final Model after the execution of the handler method.
2. Then it prepares to invoke the the handler method. To invoke this method, it has to resolve the arguments. If the method has a parameter with @ModelAttribute, then it would search in the temporary Map object with the value of @ModelAttribute.
3. If it finds, then the value from the Map is used for the handler method parameter. It it doesn't find it in the Map, then it checks if there is a SessionAttributes annotation applied on the controller with the given value. If the annotation is present, then the object is retrieved from the session and used for the handler method parameter. If the session doesn't contain the object despite of the @SessionAttributes, then an error is raised.
4. If the object is not resolved through Map or @SessionAttribute, then it creates an instance of the parameter-type and passes it as the handler method parameter. Therefore, for it to create the instance, the parameter type should be a concrete-class-type (interfaces or abstract class types would again raise an error).
5. Once the handler is executed, the parameters marked with @ModelAttributes are added to the Model.

**Regola:**Un Controller che annota una o piu chiavi con **SessionAttributes:  
  
PRE Handler: Cerchera’ in sessione un oggetto registrato su quella chiave:**- Se non lo trova provera’ a crearlo e registrarlo cercando un metodo annotatocon@ModelAttribute **POST Handler: Scrivera’ a fine metodo handler in sessione su quella chiave:**- Attraverso chiamate al Model api  
- Attraverso un metodo annotato con @ModelAttribute per quella chiave.  
  
Un metodo handler con un’annotation a livello args@ModelAttribute(k1) **Nella View:**Nella jsp come e’ possibile verificare la chiave del @ModelAttribute sara’ presente sempre sia su Request che Session Attribute.

I Custom tag di Spring o le EL daranno priorita all oggetto in Session.

**E’ possibile verificare che il metodo @ModelAttribute con la chiave che corrisponde all SessionAttribute viene eseguito solo la prima volta.**

**Let's explain @SessionAttributes lifecycle:**Quando su un Controller annotato con SessionAttribute({“k1”,”k2”}) viene invocato su un suo metodo handler annotato a livello args con @ModelAttribute(“k1”)

Inizializzazione:  
1) La DS verifica che nella session per la request corrente ci sia un oggetto per la chiave “k1”.  
  
2) Se non lo trova, cerca un metodo annotato con @ModelAttribute(“k1”) per quella chiave .  
Se trova il metodo lo esegue e popola k1 con scope session.

3) Se non trova la chiave ne in sessione ne trova un metodo annotato con @ModelAttribute va in eccezione.

Updates:

1. Se trova in sessione un oggetto per K1 lo utilizza per popolare l arg @ModelAttribute
2. Le modifiche su quell oggetto rimarranno in sessione.

Clear Session:

Un oggetto puo’ essere rimosso dalla session utilizzando le api di org.springframework.web.bind.support.SessionStatus (passato al metodo handler)  
 .setComplete (che ripulira la singola chiave)