**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

**UserController:**

**Display MAV: UserController.load**

**TC1: Caso standard**handler crea MAV  
dati MAV[key] accessibili injsp

**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 @SessionAttribute o @ModelAttribute.  
  
Se ci sono li esegue e li utilizza per popolare l implicitModel prima di passarlo all handler.

**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>>

**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.  
  
**Necessario:**1) 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)

**POST dati da form**

2) Handler che prende in input un oggetto marcato con @ModelAttribute (coi campi che combaciano con campi form e chiave che combacia con commandName) (**TC5**)

In tal caso l’ oggetto spedito dalla form, a parita di chiave sovrascrive l oggetto new creato dal prepare.  
  
A questo punto il prepare si rivela una esecuzione inutile (@SessionAttribute)

**DEMO**

**.save (save GET link)  
 (save POST form)  
  
  
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) **TC7:**

**UserControllerWithSessions:**

**TC8:**Init con CommandObj a livello request.  
@SessionAttribute commentato.  
Command creato da codice.  
Metodi prendono in input oggetto creato inizialmente, ma che perde I campi.  
**Necessario aggiungere chiave nel MAV ogni volta**? **SI** se no non ha il cmb obj.

**TC9**: (Equivalente di TC8 con annotation).   
@ModelAttribute su initCommand method.  
@ModelAttribute sui metodi handlers.  
Passa sempre dal initCommand, per cui alla fine ha il val default

**TC10:**@SessionAttribute a livello classe che combacia (chiave) con @ModelAttribute.  
Verificare:  
@ModelAttribute(ss) chiamato una sola volta.  
Navigazione back e next mantiene I valori.  
Session Attribute viene rimosso con SessionStatus.setComplete()

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)