Spring MVC Framework...

POJOs---> beans (runtime)

#Framework for building web applications in JAVA

#Based on Model-View-Controller design pattern (seperation of concern)

#Leverages features of Core Spring (IoC,DI..)

Spring MVC implementation:

#Spring-way of building web-apps

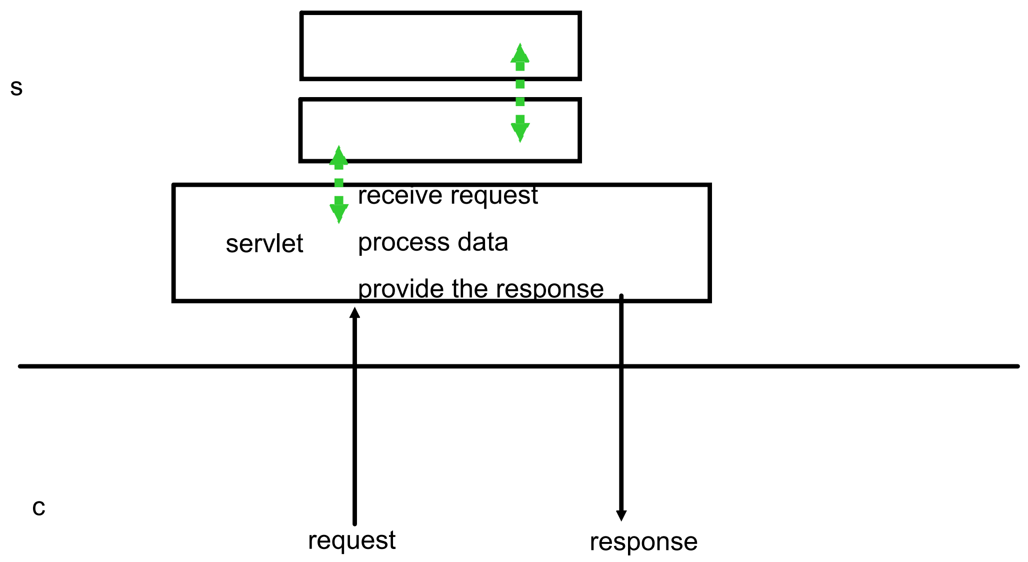
#Leverage a set of reusable UI components

#Spring API help manage application state for web requests (default : web request are stateless)

#Robust form handling : validation,conversions,mappings....

#Flexible in config of view/presentation

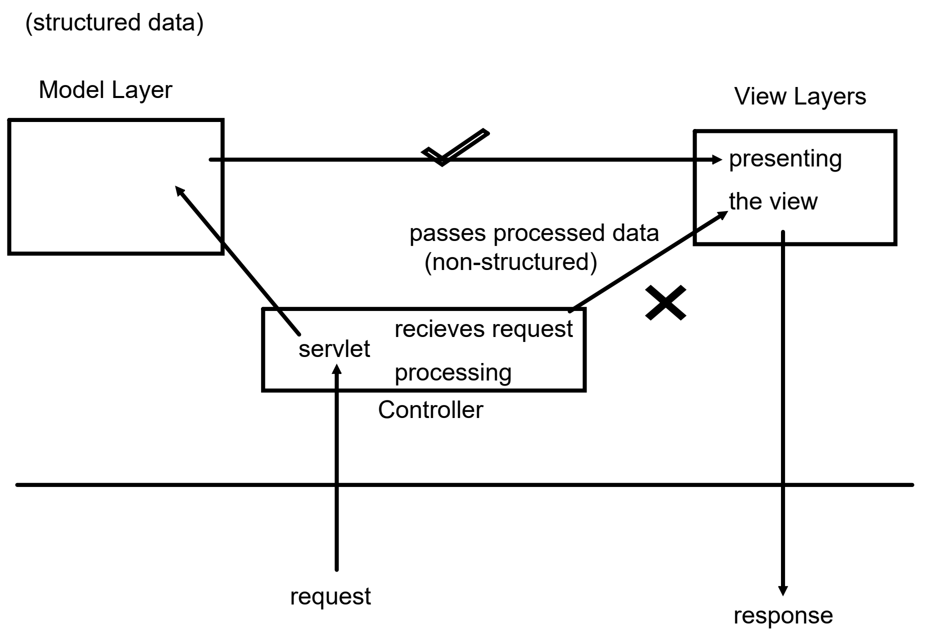
Servlet technology



MVC : seperation of concern

provides different container for different requirements of servlet spec

MVC design pattern



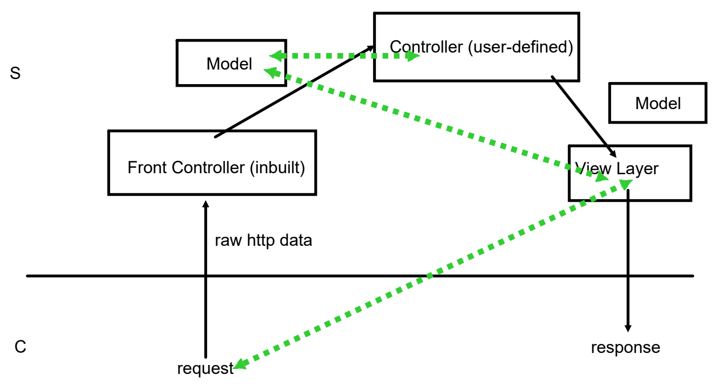
Model layer : responsible holding structured data and transacting data among diff elements/layers of app

Spring MVC :

Components:

1. View : set of web pages to layout UI components
2. Collection of Spring beans : (controllers,services,helpers,dao...)
3. Spring configuration : (XML,Annotation or Java)

Spring MVC behind the scene....



>Front Controller : DispatcherServlet
  #part of Spring framework...
 #already developed by Spring dev team

user-defined
 Controller classes (C)
 View templates (V)
 Model objects (M)

Controller :
    ==> Class created by developer
    ==> Contains the business logic
            #Handle the request
            #Store/retrieve (db, web service...)
            #Place the data in model
    ==> Send to appropriate view template...

Model: contains your data

all storage / retrieval would be modelled on Model structure

any java object / collections...

View

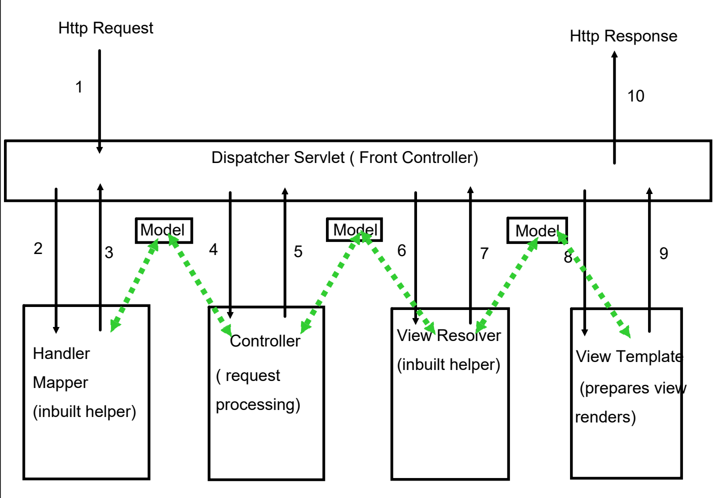
Flexible : supports many view templates

#most common : JSP + JSTL

other eg:

Thymeleaf,Groovy,Velocity, Freemarker...

Detailed Spring MVC internal working (DispatcherServlet)



1. request from client (along with raw http data)
2. DS sends request to Handler mapper to confirm/find the appropriate controller
3. recommendation sent back to DS
4. DS calls the specific controller based on recommendation
5. Controller returns back the view template name
6. DS sents view name to View Resolver to get description over view template
7. returns detailed description of view template (view location/extension...)
8. calls the actual view template
9. returns back the rendered result to DS
10. returns back that response to client...

web.xml (java servlet spec [1.4]))describes how to deploy web module on a servlet container(tomcat)

Implementation:

1. Dynamic Web Project (need to have web.xml deployment descriptor)
2. Add Spring framework api to lib folder(auto added to java build path)
3. get and add the lib support for jsp-jstl
4. Spring MVC configurations

Part 1: WEB-INF/web.xml

#Configure Spring MVC DispatcherServlet (register)

#Set up URL Mapping to Spring MVC DS

Part 2: another xml config file : WEB-INF/

#Each Servlet can have a personal config file to config its helper

#Default naming convention <servlet name>-servlet.xml (auto associated with servlet)

eg: dispatcher : dispatcher-servlet.xml

#any other custom name needs to be registered in web.xml

#spring config file~ applicationContext file

==>Add support for spring component scanning

==>Add support for conversion, formatting, validation...

==> config the Spring MVC View Resolver (varied view templates)

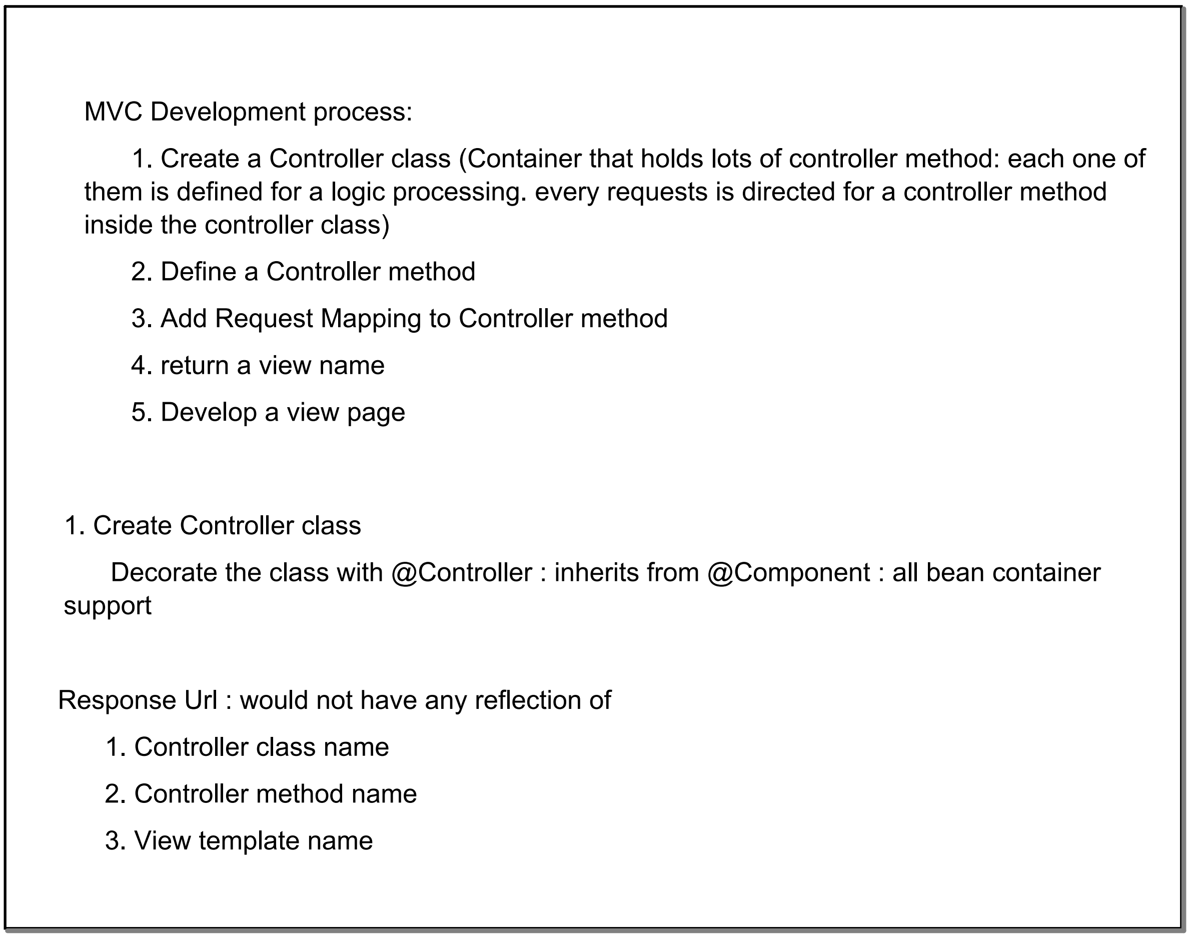
Controller returns view name:

1. prepend the prefix : location of view template
2. append the suffix : extension (type if view template)

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
        <!-- need to inject 2 dependency -->
        <property name="prefix" value="/WEB-INF/view/"/>
        <property name="suffix" value=".jsp"></property>
    </bean>
```

eg: controller return a view name "main-menu"

create a path: /WEB-INF/view/main-menu.jsp



MVC Development process:

1. Create a Controller class (Container that holds lots of controller method: each one of them is defined for a logic processing. every requests is directed for a controller method inside the controller class)
2. Define a Controller method
3. Add Request Mapping to Controller method
4. return a view name
5. Develop a view page

1. Create Controller class

Decorate the class with @Controller : inherits from @Component : all bean container support

Response Url : would not have any reflection of

1. Controller class name
2. Controller method name
3. View template name

SpEL: Spring Expression Language : ${}

: direct access over Model data

Data Process:

1. extend the method to process data:
2. Read the form data in controller method
3. Convert name into upper case
4. Add uppercase version to model to be shared with view

#Controller method have the access over HttpServletRequest object

#Controller method have the access over Model object

#Reading client data: spring-way

Special Annotation to refer client data @RequestParam

#Spring will read param data from request object and bind it to the variable uname

#different possibilities of @RequestMapping

#different possibilities of Reading client data

#different possibilities of adding data to model

@RequestMapping : used to map request url with a specific controller method

==>can be used with the controller class directly:

#<url mapped with class>/<url mapped with method>

eg:

HomeController (/base) : /home : method1() : /base/home

StudentController (/student): /home : method1() : /student/home

#Fallback method for controller mapping

#We can configure Controller method to respond to a specific http method

#by default method responds to all http method

Same url can be mapped to multiple methods if http method they respond to are different

Forms in spring mvc:

Traditionally : HTML forms (input) : robust and not compatible with framework req.

==>Spring MVC Form Tag : building block for web pages

#configurable

#reusable

#can make use of data bindings

#can make use of validations

#have access over java objects/beans

(auto set/get data from java objects)

MVC Form tags at the time renderring--> HTML tags

<form:form : form

<form:input : text field

<for:textarea : multi-line text field

Fully compatible and can be easily integrated with regular HTML

TO get support of Spring MVC form tag:

#need to add spring namespace in jsp file :

#we work on student entity

Showing Form:

Spring Controller activity

must add Model attribute : student entity

student entity object(bean) will hold form data for data binding

#instead of hardcoded country list: outsource the list to student class

properties file to load country options

#add countries.properties file : /WEB-INF/countries.properties

#add refrence of prperties file in dispatcher-servlet.xml

#take help of jstl core tags for looping contructs: need to add namespace inf jsp file

Performing input from user : validation

Form Validation

req:

- required field
- valid numbers in range
- valid format
- ....
- custom business validation rule

Java Standard bean Validation API

#Defines metadata (annotation) for entity validations

#Available for server app and client app

Spring version 4 and higher supports bean validation API

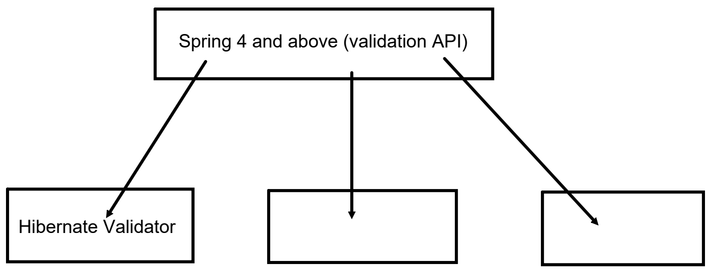
Annotation supported for validation:

- @NotNull
- @Min
- @Max
- @Size
- @Pattern.....

Java Standard bean validation api (JSR-303/309)

#Only a specification....vendor independent ....portable

#Still need an implementation



have fully compliant JSR-303/309

not tied to ORM

Hibernate : annotation (implementation)

javax : annotation (java bean validation) : suggested to standard bean validation api annotation: prevent vendor locking

Spring MVC Validation:

1. Apply validation rules directly inside the entity class(upon entity fields)
2. specify the error messages in your entity class
3. Annotation based validation
4. Check the constraint satisfaction inside controller method..

Required filed validator

#Apply the appropriate/req annotation on entity fields

#Provide required message to be displayed if constraints are not satisfied

#add error form tag in view page to display error message

#in controller method we need to check if constraints are satisfied...

Empty HTML field submissions are empty String and not the null values

1. Empty HTML field
2. Blank spaces

Requirement:

#Empty HTML field : empty string ==> NULL

#Remove all leading and trailing spaces

Add special method: pre-process each request to our controller

root cause for int based problem is : primitive int

Need to convert into Wrapper type

#Need to override the default Spring MVC validation message

=>inside src directory : new sub-directory

=>add properties file

=>message to override the MVC Validation message

=>key: <type of validation> + "." + <object name> + "." + <field name>

eg: typeMismatch.student.freePasses=<message>

=>Add reference of properties file in dispatcher-servlet.xml file...

Adding Custom Validation:

1. Need to create a annotation
2. Add the validation rule (need to create a separate class)