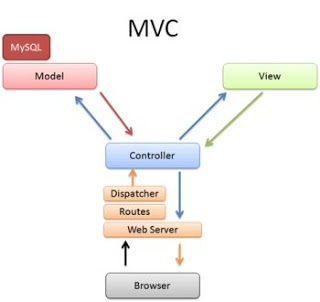
**POJO:** Plain Old Java Object. It’s a java class which has its properties set private. Use getters/setters to access member variables.

**Model:** Represent data of application.

**View:** User interface

**Controller:** Get data from view, process it, then display it back on view.



**IoC Container:** Inversion of control, cover a board range of techniques that allow an object to become a passive participant in the system. Instead of create a new object of other class, it create an private variable, then let someone else (in this case is Spring) inject the object you need to that private variable

**Dependency Injection:** a design pattern that removes the dependency of the programs. Provide the information from the external source such as XML file. Make code coupled.

**Bean:** contains the information called Configuration metadata which is needed for the container to know:

- How to create a bean

- Bean’s lifecycle details

- Bean’s dependencies

Metadata translate into a set of following properties that make up each bean:

Class, name, scope, constructor-arg, properties, autowiring mode, lazy-initialization mode, initialization method, destruction method.

**Bean Scopes:**

Singleton: (default) In Spring container, a singleton scoped bean will be instantiated only once and the same will be used for its lifetime. As an static object

Prototype: allow bean to instantiated whenever it is requested. As a normal class, when you create instance it create a new one.

Request: instantiate the bean for a single HTTP request. This instantiated object lives through the HTTP request.

Session: bean instantiated lives through the HTTP session.

Global\_session: equal as session scope on portlet-based web app.

**ApplicationContext**: an interface providing configuration information to an application. Provide:

Bean factory method for accessing app components.

Ability to load file resource in a generic fashion.

Ability to publish events to registered listeners.

Ability to resolve messages, supporting internationalization.

Inheritance from a parent context.  Single parent context can be used by an entire web application, while each servlet has its own child context that is independent of that of any other servlet.

**BeanFactory:**  a subset of ApplicationContext and provide lesser functionalities.

**@RequestMapping**: Controller have methods, in order to know which method gets the order from which URL, we put annotation @RequestMapping with the URL corresponding go as an argument above that method.

Argument can be changed depend on which data we want to get from the order and send to server (HttpRequest, HttpResponse, Object…).

Applied both class and method.

**@RequestMapping (produce={“application/json”, “application/xml”}, consumes=“text/html”)**

🡪 method can consume message only with Content-Type as text/html and is able to produce messages of type application/json and application/xml

**@RequestMapping with @PathVariable : can create dynamic variable at the url .**

@RequestMapping("/profile/{username}")

public String showProfile(Model model, @PathVariable("username") String username){}

{username} on @RequestMapping is PathVariable, it means that this method will serve any request of the format “/profile/username”. We capture the actual username in the next line using @PathVariable annotation, and store in the String username

**@RequestMapping fallback method:**

@RequestMapping(“\*”)

public String fallbackMethod(){ return “fallback method”;}

Create a fallback method for the controller class to make sure we are catching all the client requests even though there are no matching handler methods.

**@RequestBody:** Spring will convert the content of the incoming request body to parameter object. Contain the body of the HTTP request.

**@ResponseBody;** Spring will write its return value to http response automatically.

**@ModelAttribute:** annotated method parameter can be mapped to an attribute in a model using @ModelAttribute in controller. Provide reference data for model.

The primary objective of this annotation to bind the request parameters or form fields to an model object.

**- On method parameter:** the argument should be retrieved from the model. If not present in the model, argument should be instantiated first, then added to model.

**- On method:** indicate the purpose of that method is to add one or more model attributes. Map request parameter or form fields to an model object.

Allow access to the object in View.

**@CookieValue:** used to bind a method parameter to a HTTP cookie.

**@RequestHeader:** used to bind a header value to a method parameter.

**@RequestParam:** bind request parameters to a method parameters in controller. Hold the value of request parameter.

**@Component, @Service, @Repository, @Controller**: used for automatic bean detection using class-path.

**@Component:** generic and can be used across application.

Use this to enable auto scanning all the beans in the specified package and define them in bean config file.

Put “context:component” in bean config file 🡪 enable auto scanning feature in Spring.

**@Service:** annotate classes at service layer level.

**@Controller:** annotate classes at presentation layers level.

**@Repository:** annotate classes at persistence layer, which will act as database repository

**@RequestBody and @RequestParam using together:**

**Ex:** RequestBody get “name = abc”. Meanwhile RequestParam get: “abc”

If RequestBody was retrieved first then RequestParam. So there will be no data for RequestParam to get 🡪 Error 400

RequestBody get “name= abc” -> RequestParam get ??? -> ERROR

Otherwise, RequestBody will retrieve data missing the value because value was retrieve by RequestParam

RequestParam get “abc” -> RequestBody empty

**ModelAndView**: Every method in Controller class return an instance of ModelAndView class. This class represent for each view, where the app will be address to after controller processing finish.

For example: index.jsp is a view.

An instance of Student class will be pointed as model of view index.jsp in ModelAndView.

**Autowired:** Automatically connect beans. (caution because this’s one on one relationship 1 interface can only wire with only one bean)

**Spring MVC Architecture**

**Spring’s Dispatcher Servlet:** acts as a front controller between Spring app and its client. The Dispatcher Servlet intercepts requests coming to the app and consults the Handler Mapping for which controller to be invoked to handle the

request.

**Handler Mapping:** find appropriate controllers that handle specific requests.

**Controller:** process the requests by calling other business/service classes. Output can be attached to model objects which will be sent to view. To know which view will be rendered, the controller consults the View Resolver.

**View Resolver:** finds the physical view files from the logical names.

**View:** physical view files (jsp, html...)

**Restful Web Service Spring**

Building Restful web service in Spring HTTP requests are handled a controller. Identified by the @RestController annotation

**Stateful**: After receiving data from client side, server side will process it and send result back to client side. Then, server side does not keep any data from client side, so there’s no connection between server side and client side

**Stateless:** Opposite of stateful.

**Hibernate:** (Hibernate in short) is an object-relational mapping library for the Java language

**Hibernate Query Language (HQL):** hibernate provides an SQL language similar to SQL language, allow SQL-like queries to be written against Hibernate data object

All data type in sql table is redefined as an object in java, for easy data binding, hibernate also provides useful function to collect object data as list.

**Restful:**

**Representational State Transfer**

GET: **List** the URIs and perhaps other details of the collection's members.

PUT: **Replace** the entire collection with another collection.

POST: **Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.

DELETE: **Delete** the entire collection.

Lib used in this case is jboss can be easily found here :

http://docs.jboss.org/resteasy/docs/3.0.6.Final/userguide/html\_single/index.html#Installation\_Configuration