Java Based Framework :

Web Based Application Dev using Java : J2EE

Spring : improvisation over J2EE

=>Multiple Deployment Descriptor (EJB)

=> Multiple Interface

=> Poor in performance in production

Rod Johnson : lightweight variant of J2EE (without EJB)) :

Spring

Spring Framework : goals

1. Any Java Based Application
2. Highly modular development
3. Lightweight : POJO (Plain Old Java Object)
4. Minimize the boiler plate code
5 Three pillar
    - IoC
    - DI
    - AOP

Modular

Servlet Spec / API

Multiple layer :

Core Container Module

Beans, Core APIs, Context....

Infrastructure Layer

Data Access Layer

Web Layers

Test Layer

Base Framework

Spring Projects :

Additional Spring Modules designed on top of Base Framework

Spring Cloud

Batch

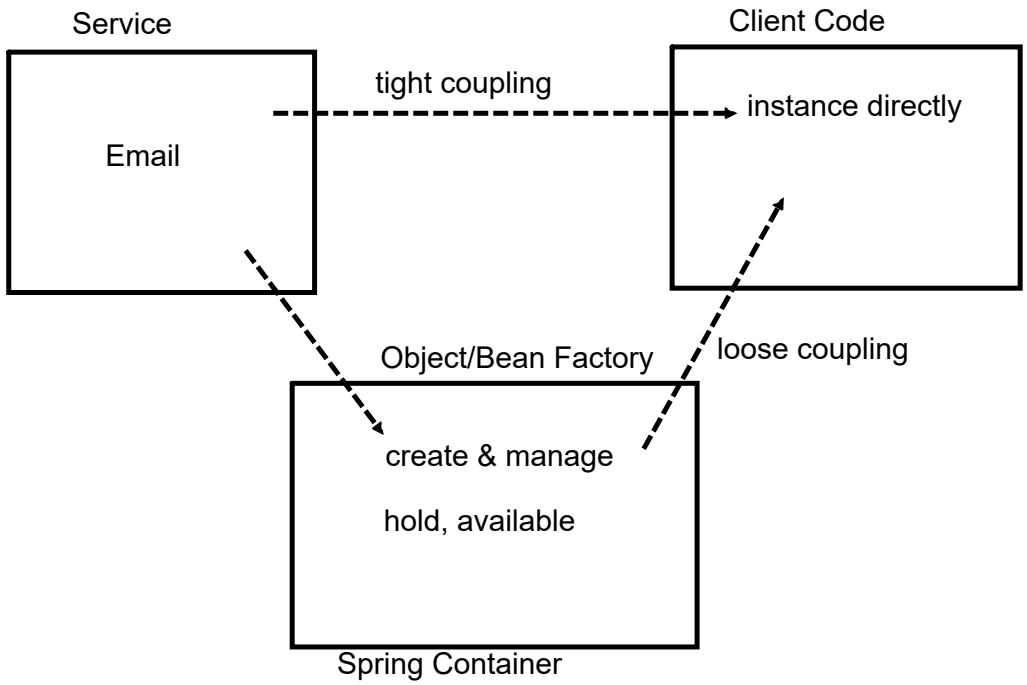
Data

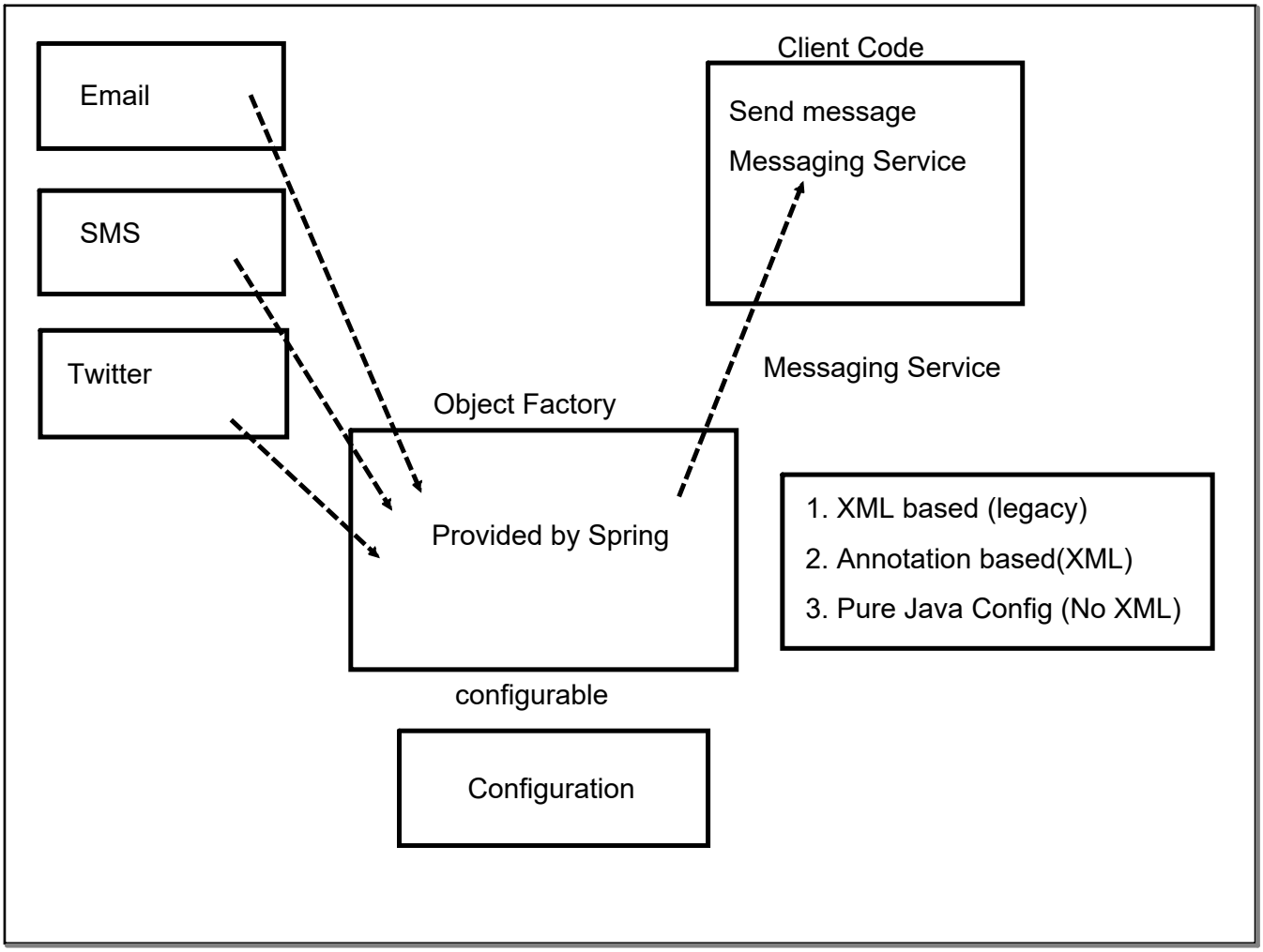
Security

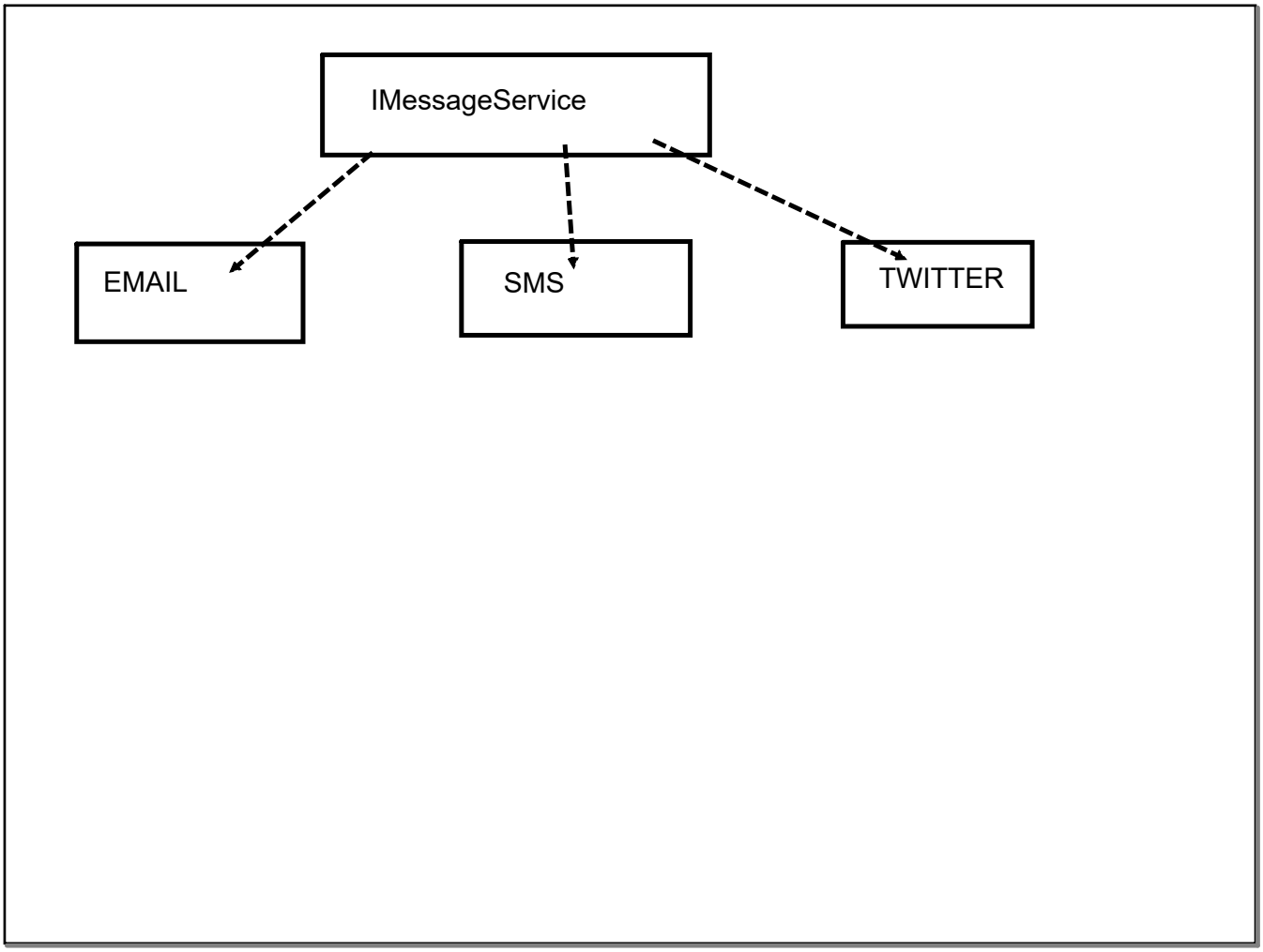
Base Framework : library jar

IoC : Inversion Of Control :

Outsourcing the construction and management of Object : IoC




Email
SMS
Twitter
Client Code
Send message
Messaging Service
Messaging Service
Object Factory
Provided by Spring
1. XML based (legacy)
2. Annotation based(XML)
3. Pure Java Config (No XML)
configurable
Configuration

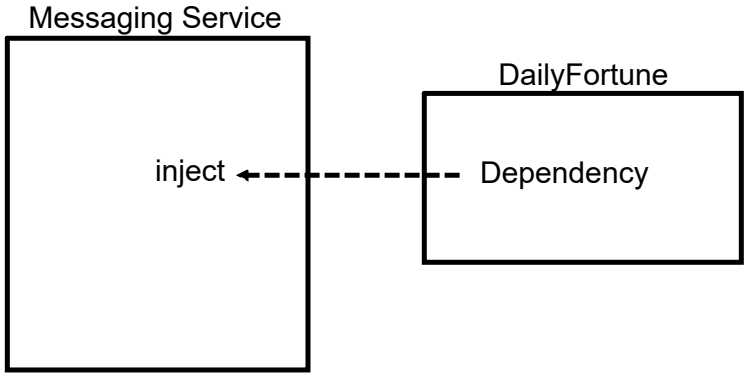

IMessageService
EMAIL
SMS
TWITTER

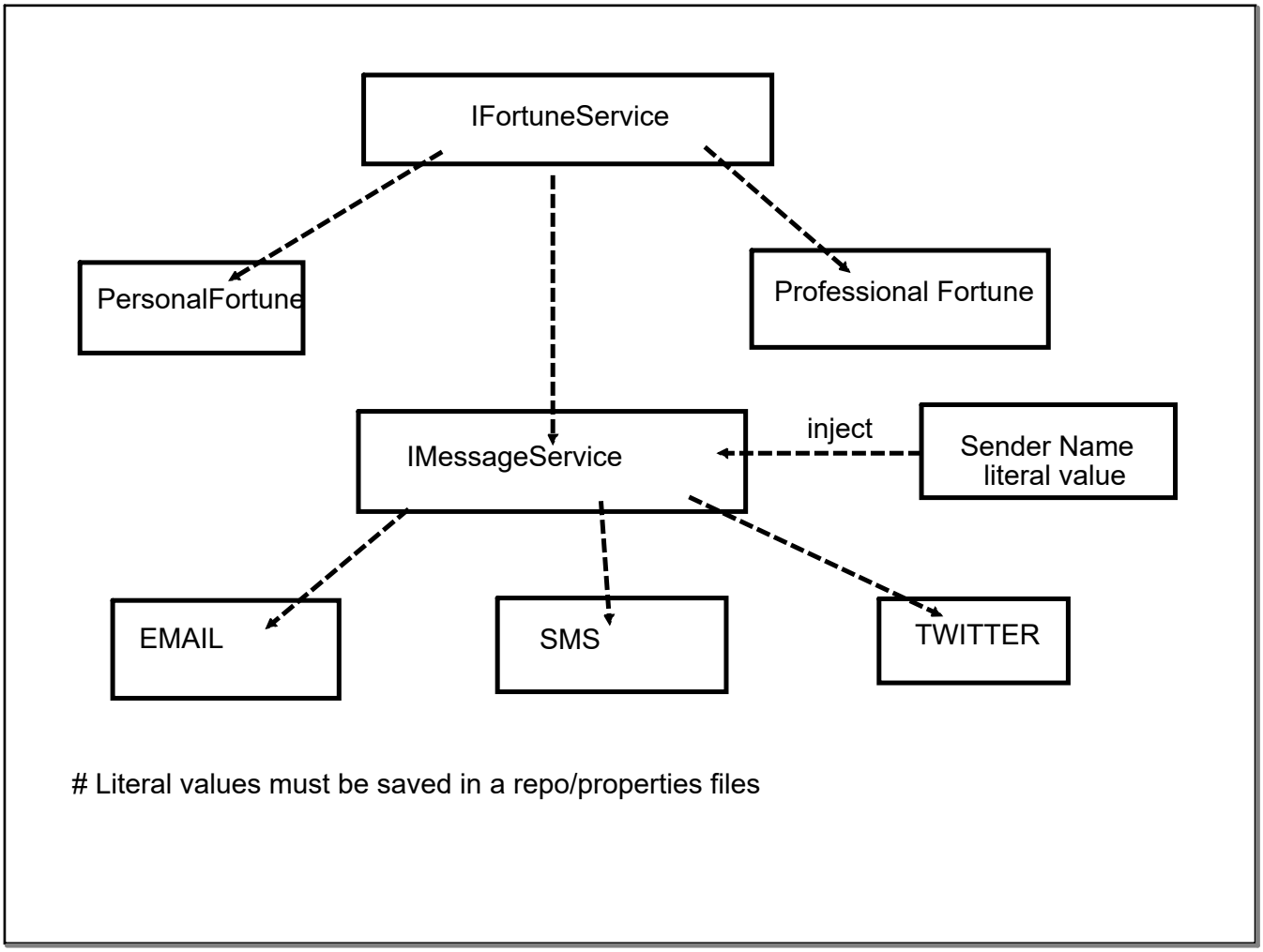
Dependency : Service may be dependent of multiple service

Car :

- parts of the car
- color
- engine power

DI : Dependency Injection





Literal values must be saved in a repo/properties files

Types of DI

Constructor based DI

Setter Based DI

```
<bean id="fService" class="com.dell.training.spring.core.service.PersonalFortune"></
bean>

      <bean id="msgService"
class="com.dell.training.spring.core.service.EmailService">
      <!-- Provision to recieve dependency : Cosntructor based -->
        <constructor-arg ref="fService"/>
     </bean>
```

Bean creation

Manage the bean (life cycle of bean)

1. How long bean will live???
2. How many instance are created ???
3. How beans are shared ???

Scope :

Default Scope is : Singleton

Create only single instance, cached in memory and shared the same instance

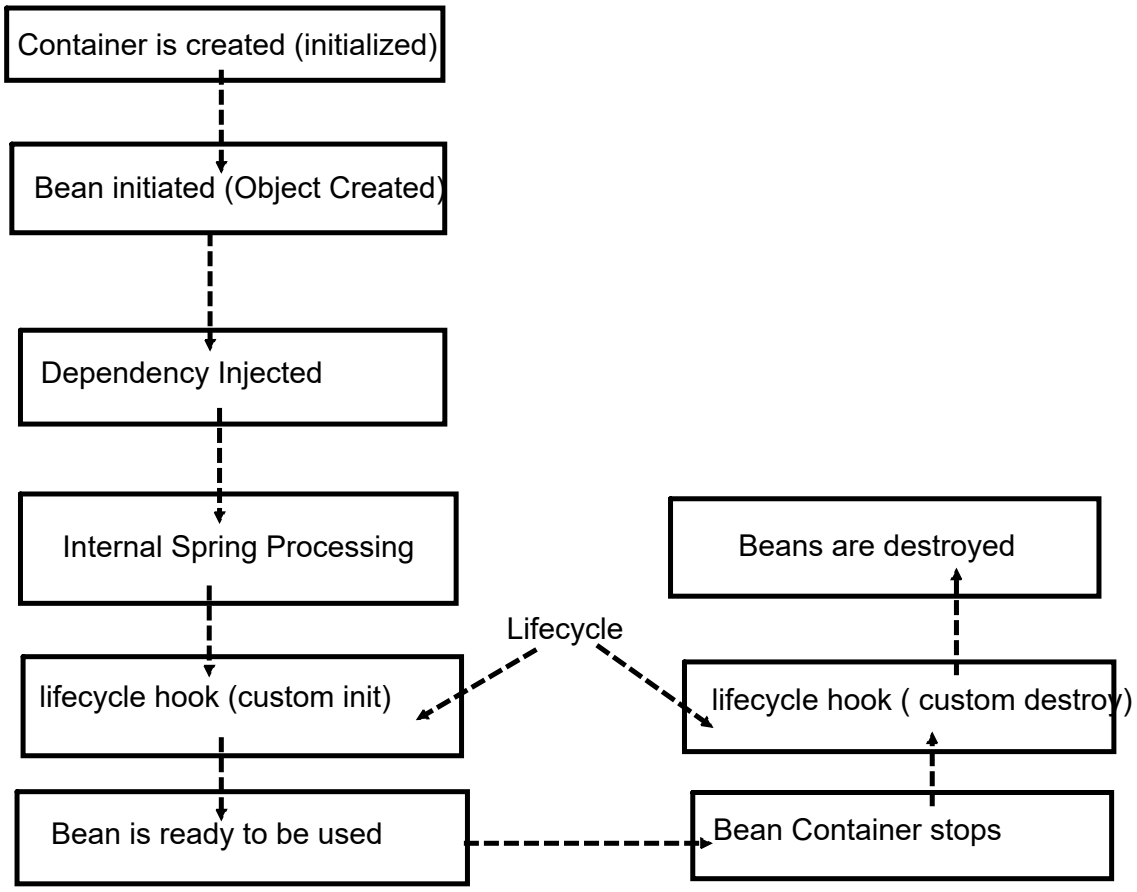
prototype :

new object on every demand

Web App

- request
- session
- global-session

Lifecycle stages:



Lifecycle hook

1. any name
2. any access modifier
3. must not be static
4. may have a return value, but can't be collected
5. Must not have any args

Prototype : Spring Bean Factory does not maintain the complete lifecycle of prototype beans !

Creates , shares , forgets...

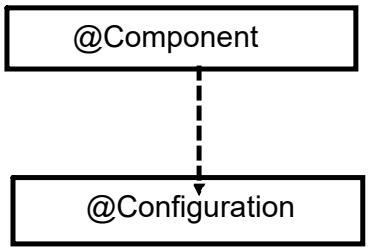
Annotation based config : XML file would be refer the resources (path)

Three approaches of DI

1. Constructor
2. Setter
3. Field based

Pure JAva Config

XML ~ JAva class



@Component : applicable on class

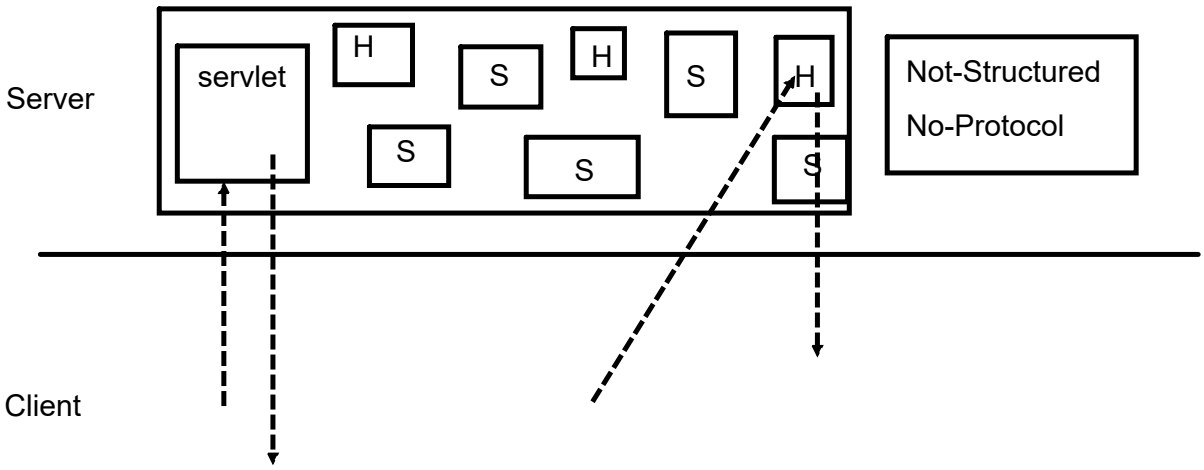
@Bean : applicable on method

Spring Web Module : APIs for developing Web Applications

Designed on top of Servlet API

static : HTML page

dynamic : Java class (Servlets)



JSP : Another way of writing servlet ---> compiled into a servlet

More presentation : JSP --> Servlet

More Logic : Servlet

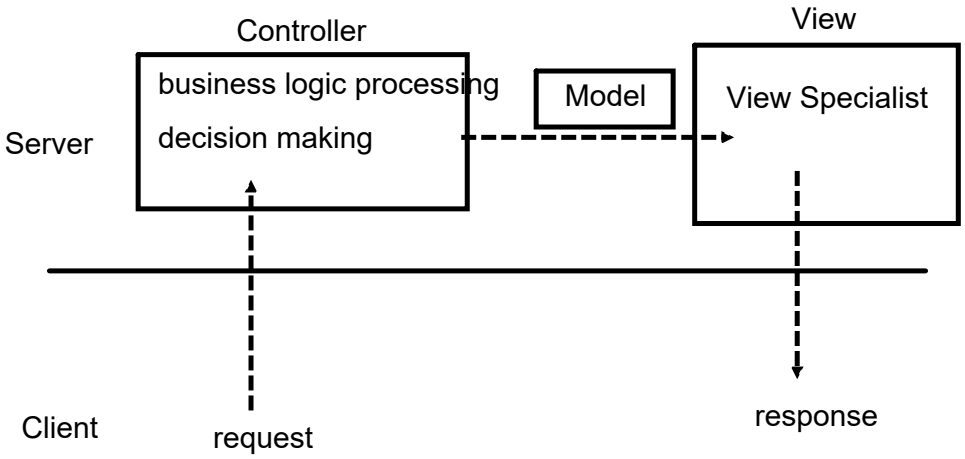
Architecture/Framework : standardize

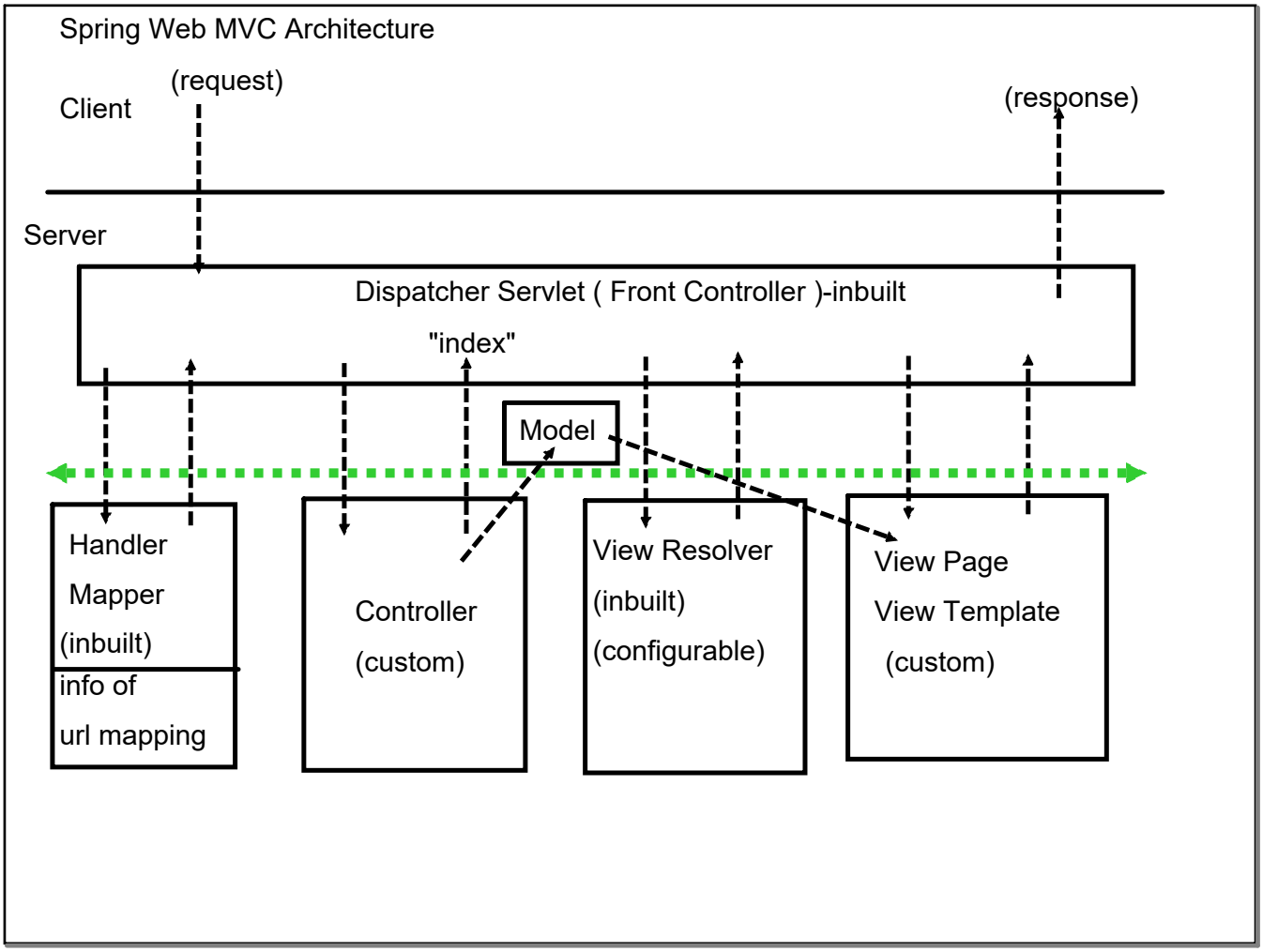
MVC architecture

View Template (Special API)

jsp + jstl

Thymeleaf, Groovy, Freemarker,Mustache




Spring Web MVC Architecture
Client
(request)
(response)
Server
Dispatcher Servlet (Front Controller)-inbuilt
"index"
Model
Handler
Mapper
(inbuilt)
info of
url mapping
Controller
(custom)
View Resolver
(inbuilt)
(configurable)
View Page
View Template
(custom)

Resources:

1. Spring framework
2. external api - Servlet API/JSP-JSTL
3. Development Server (Java based Dynamic Web App) : Tomcat, Glassfish, JBoss

Tomcat : integrate with Eclipse IDE (launch & deployment is automatic)

Eclipse IDE provides a project template for Servlet based web application

web.xml : de-facto std/file used for configuration in Servlet based application

1. We need to register a servlet :

   Servlet Based Application : All/Any Servlet created must be registered in web.xml file

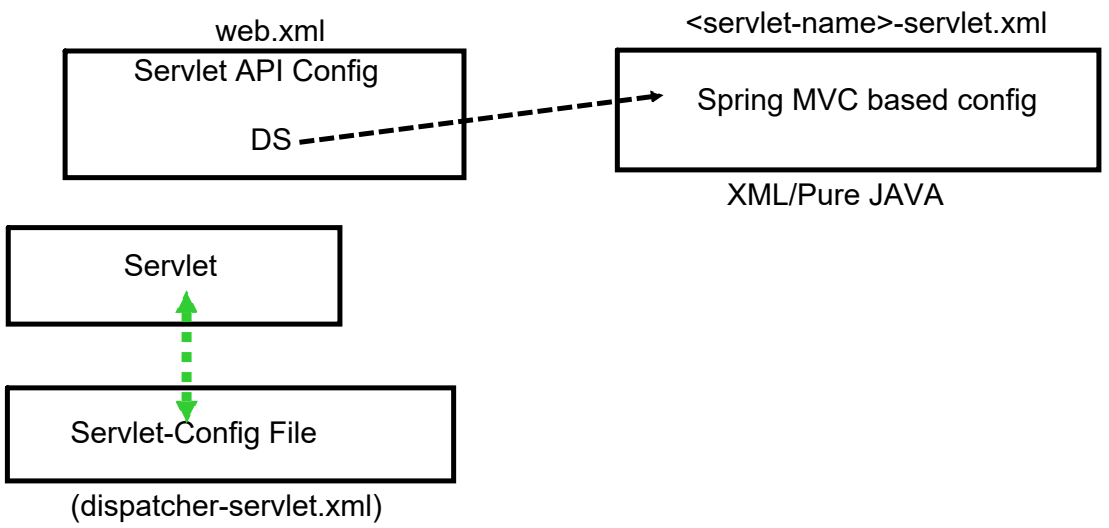
   Map the url : for which URL Servlet would be invoked

   Spring :

      Spring provide inbuilt servlet : DS---> register

      Map all/any url to DS

Configuration:



prefix : location of view pages

suffix : View Template

```
<property name="prefix" value="/WEB-INF/views/"/>
<property name="suffix" value=".jsp"/>
```

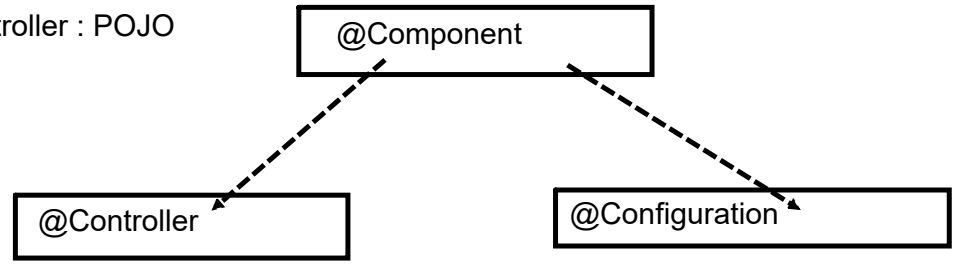
Controller

-return name of view page

eg : "index"

View Resolver : <prefix><viewname><suffix>

: /WEB-INF/views/index.jsp

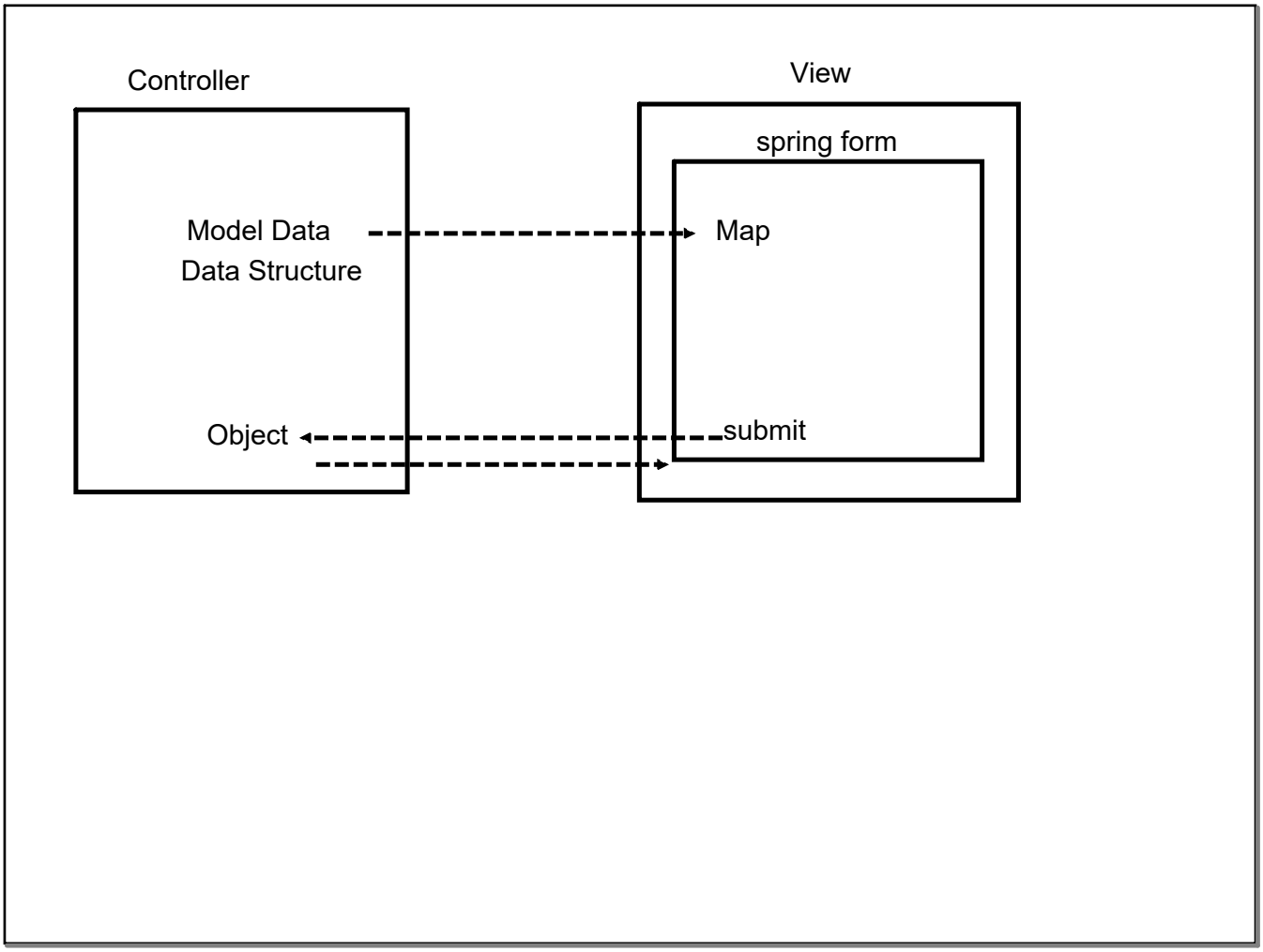


Handling Forms in Spring:

Spring Form Tag : taglibs

Advantages:

1. Map Model Object/Data with entry fields
2. Handle Validation in better way (server side validation/messages)
3. Forms will Secure (CSRF Attack)

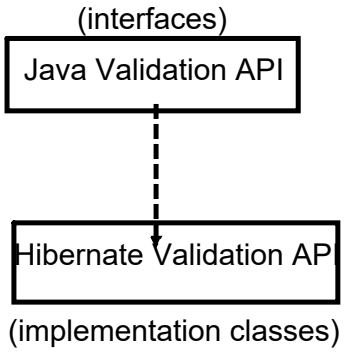

Controller
View
spring form
Model Data
Data Structure
Map
Object
submit

Validation :

client-side : Javascript

server-side :

Validation API : Hibernate Validation API



Java Validation API :

TO validate any java object

Only a spec : guidelines, rules (no implementation)

#Use the JAva Validation API annotation

Auto fetch implementation if lib is there

Prevent Vendor Locking

Pure Java Config (Spring MVC)

web.xml ~ Java classes

dispatcher-servlet.xml ~ Java classes

1. Registering the DS : Need to inherit an inbuilt class to register DS
2. URL Mapping

Project Management Tools

Maven

Gradle

Ant

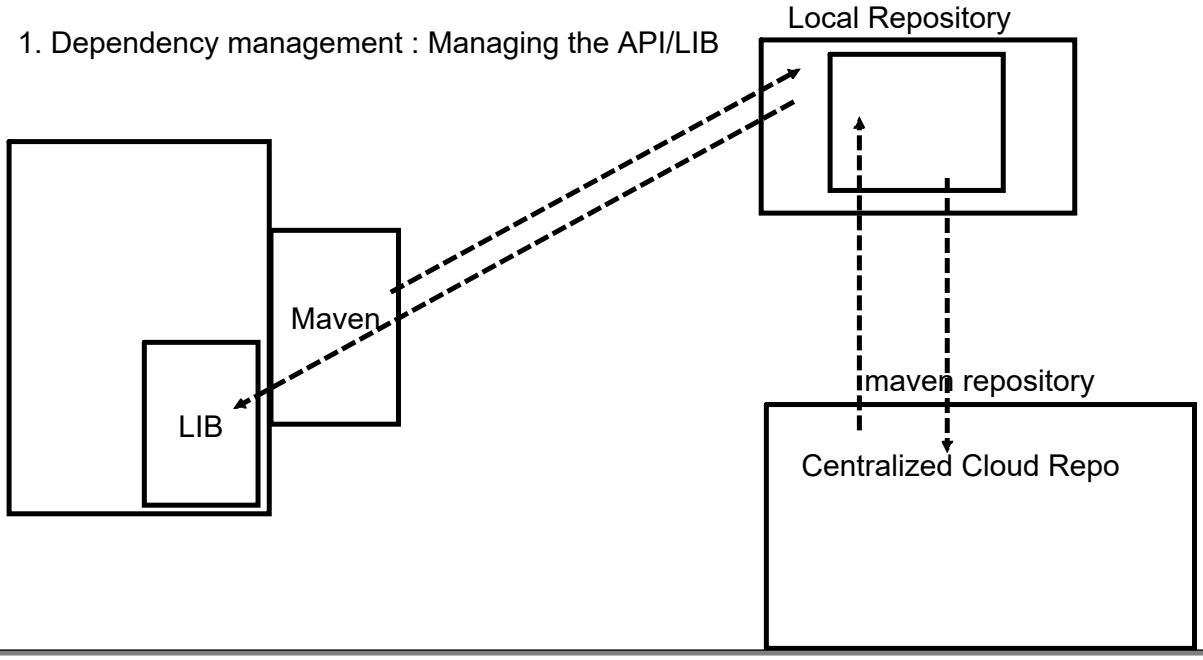
install Maven :

creates local repository : .m2 Folder

Eclipse IDE already have maven plugin

Maven :

1. Dependency management : Managing the API/LIB



Structure your project:

- std application architecture

GAV Coordinates:

- GroupID
- ArtifactId
- Version

pom.xml : Project Object Model