Spring Framework　　　　Servlet-API

MVC Architecture : Manual

Architecture is implemented strictly, disciplined way

remove lot of Boiler-plate code

abstract the low level complexity

Focus more on business logic

Most popular frameworks to develop java application

J2EE : Java 2 Enterprise Edition : Framework to develop web app using java

Complex in nature
    # lots of deployment descriptor
    # lots of interface, abstract classes needs to be created to expose a single service
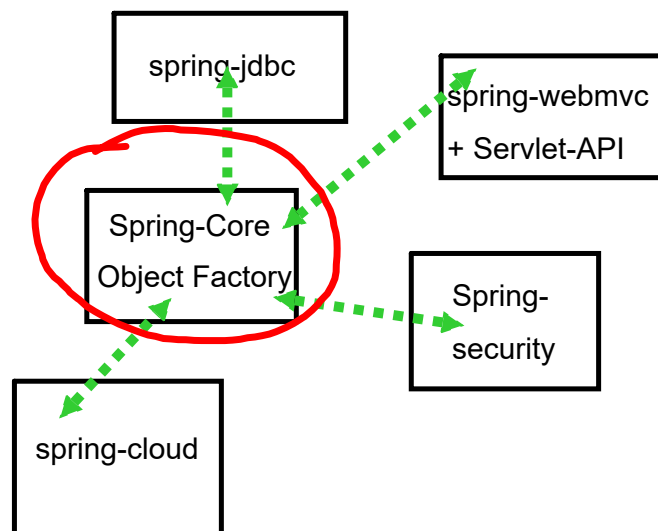   # productivity reduces, reduces efficiency

Rod Johnson

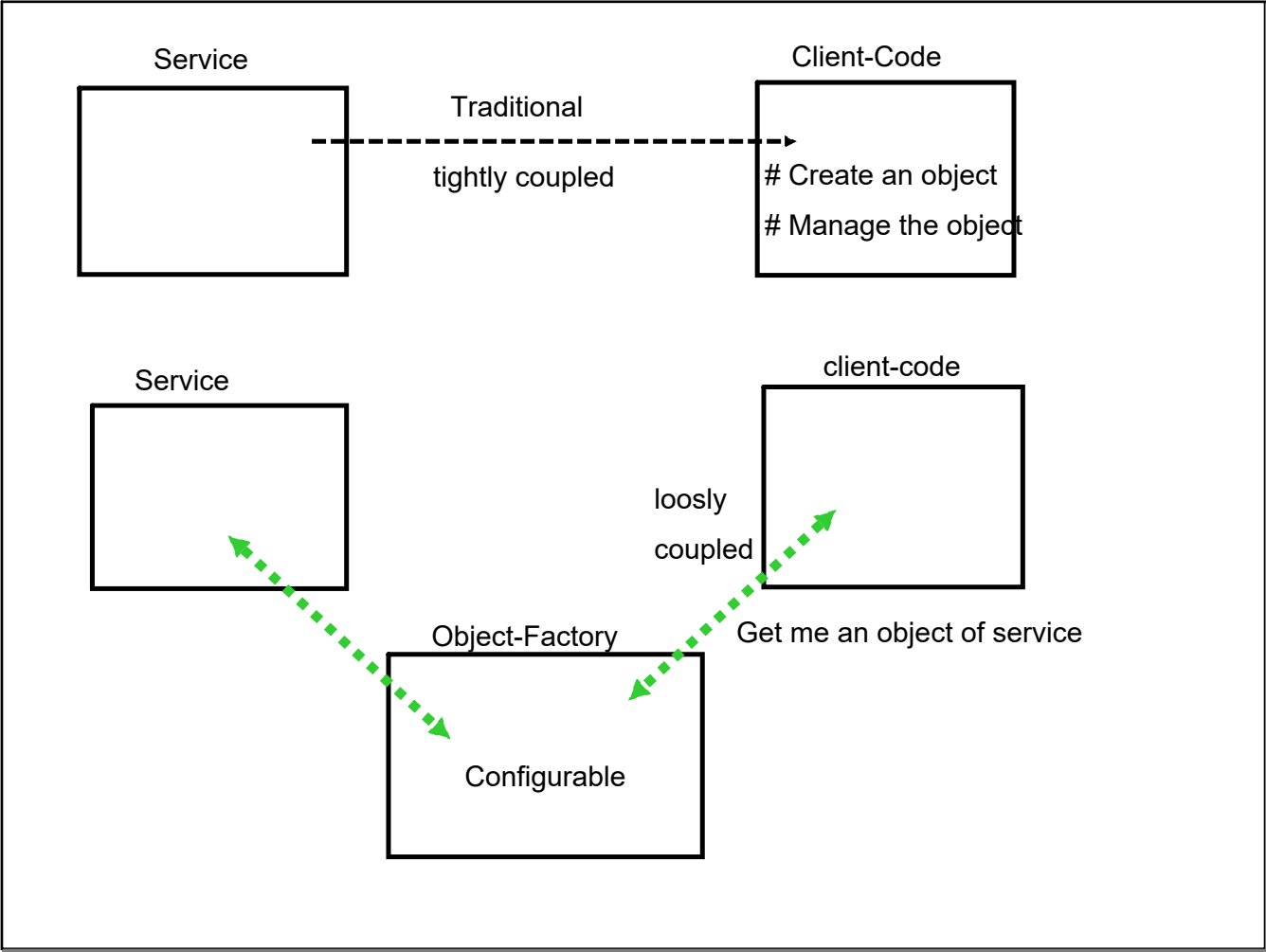=> Object Factory : Responsible for creating and managing object
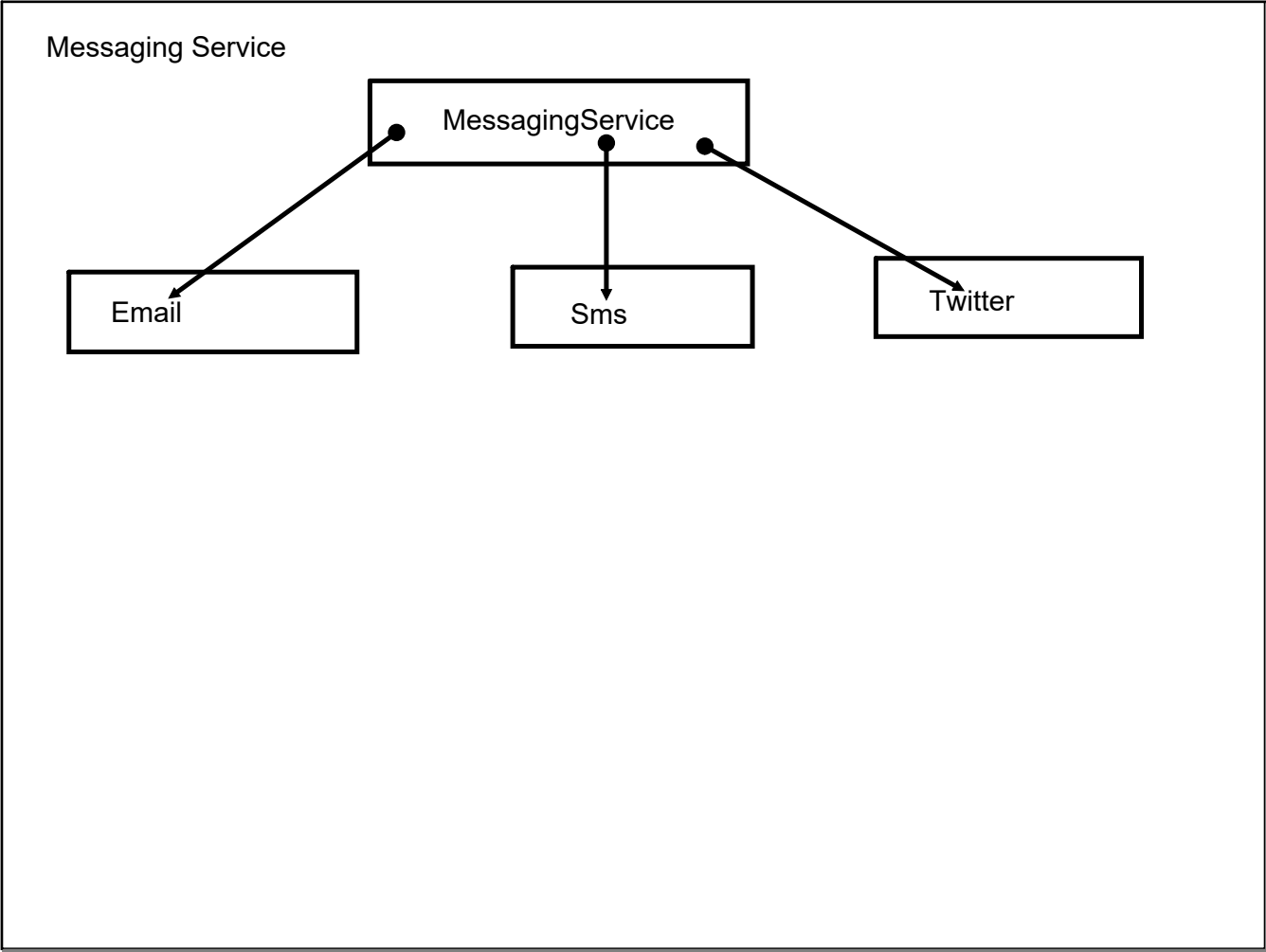
Increased the Productivity

Increased the efficiency

1. Object Factory
2. Highly Modular
3. POJOs

Spring Framework

spring-jdbc

spring-webmvc
+ Servlet-API

Spring-Core
Object Factory

Spring-
security

spring-cloud

Messaging Service

```
                    ┌──────────────────────┐
                    │  ● MessagingService ● │
                    └──────────────────────┘
                      │         │         │
        ┌─────────────┘         │         └─────────────┐
        ▼                       ▼                       ▼
  ┌───────────┐         ┌───────────┐         ┌───────────────┐
  │   Email   │         │    Sms    │         │    Twitter    │
  └───────────┘         └───────────┘         └───────────────┘
```

Object Factory | Bean Factory | Application Context

Provided by Spring - Core Module

A Custom Configuration needs to be provided to define the behavior

of Object Factory

XML Based Configuration (Legacy)

Annotation Based Configuration (Modern)

Pure Java Based Configuration (Modern)

Std Spring Framework :

    bundle of few Modules

    => Core

    => Spring-web-mvc

    => Spring AOP (proxy)

Bean Factory works on two key principals

    1. IoC : Inversion of Control

    2. DI : Dependency Injection

IoC : Outsourcing the (control of) creation and management of Object
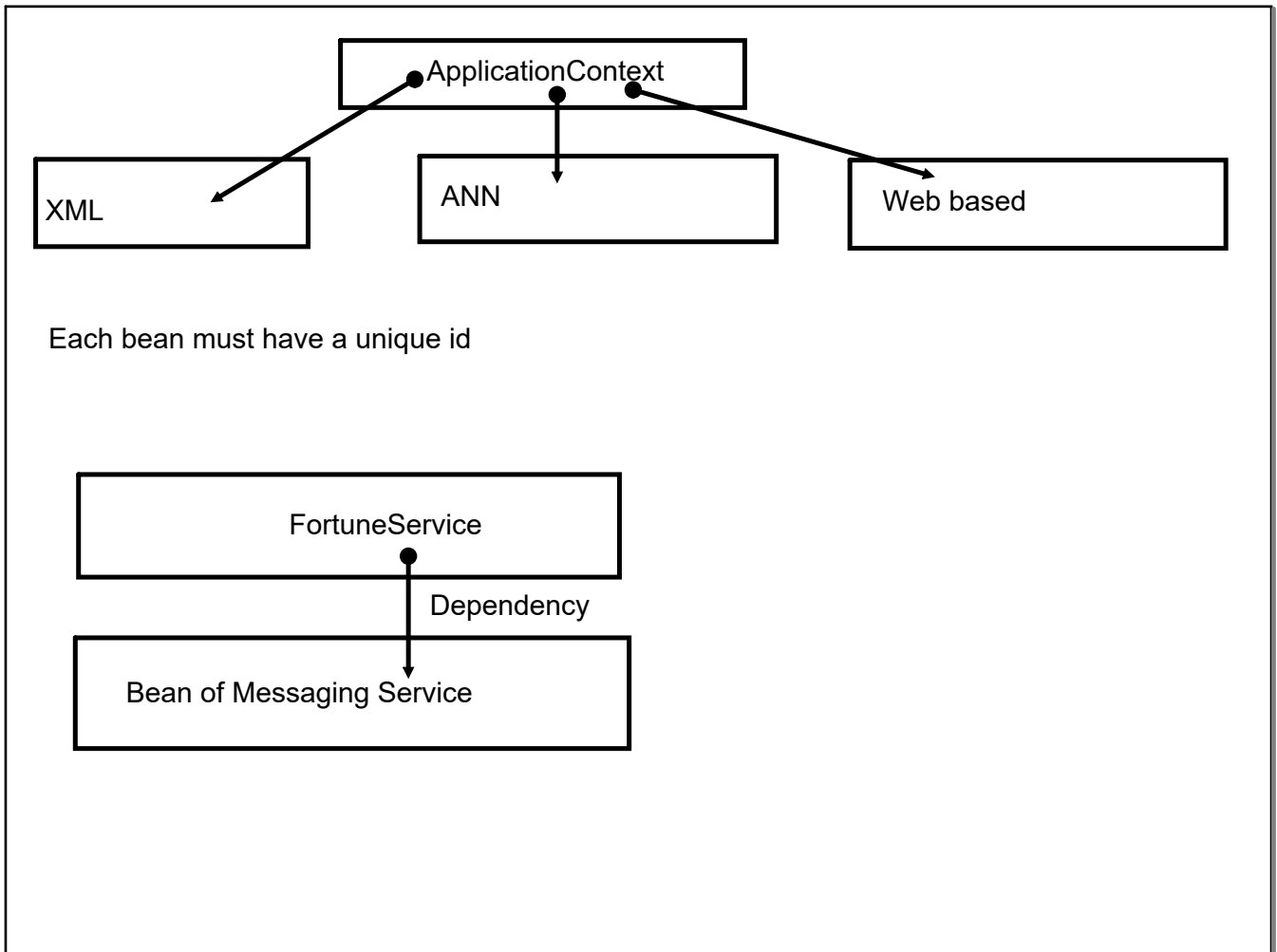
XML Based Config :

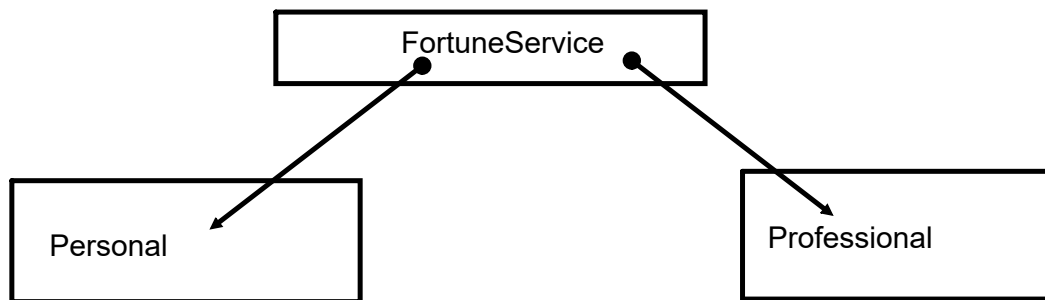   XML file + certain dependencies for support of additional spring tags

BEAN : Container(Object Factory) managed Object

Multiple classes provided for Bean Factory

\# way of config (XML or java)

\# env for which bean factory (simple java, web app)

ApplicationContext

XML

ANN

Web based

Each bean must have a unique id

FortuneService

Dependency

Bean of Messaging Service

```
                    ┌─────────────────────────────┐
                    │      FortuneService ●        │●
                    └─────────────────────────────┘
         ┌──────────────────────┐          ┌──────────────────────┐
         │                      │          │                      │
         │   Personal           │          │   Professional       │
         │                      │          │                      │
         └──────────────────────┘          └──────────────────────┘
```

Need the BEan factory to inject Fortune Service instance into EmailService

    # Constructor Based DI

    # Setter BAsed DI

```xml
<bean id="personal"
class="com.wf.training.spring.factory.service.PersonalFortune"></bean>
    <bean id="professional"
     <!-- Injecting Dependency -->

        <!-- Constructor Based -->
     <bean id="emailservice
class="com.wf.training.spring.factory.service.EmailService">
        <constructor-arg ref="personal"/>
     </bean>
```

Injecting the literal values :

 Delegate them to a text file (properties files)

literal values as key-value pair

need to specify property file in config

Bean Management :

      1. Life cycle

      2. Scope

=> Scope : Accessibility of bean

by default scope : Singleton : Single instance will be created

                 : Prototype : Diff each time

request

session   : Web based

application

Life Cycle of Bean

Bean Container (Factory) is created

Intantiates the bean

Injects the dependency

Follows internal processing

Life cycle Hook (method)--init

Platform to prepare the bean logically before it is made available

Bean is exposed to be consumed

Container is terminated...

Life cycle Hook (method)--destroy

Platform to perform clean-up operations

Bean is destroyed

Prototype : BEan container does not maintain life cycle..

Annotation based config
xml file : path reference

Creating the bean

@Component :
Any class decorated with @Component will be initiated by bean factory
By default the class name itself becomes the id , first character being small case...

DI using annotation
1. Constructor
2. Setter
3. Field

@Autowired : search for bean, if found, inject it

Scope : @Scope

Life cycle hook methods : Annotations

Pure Java Based Config :

xml file will be replaced by Java class

Pure Java Config :

   Programmatically configure Bean Factory

 before 10 am or after 5 pm : personal fortune

 else : professional fortune

                    Expose the bean

@Component	@Bean

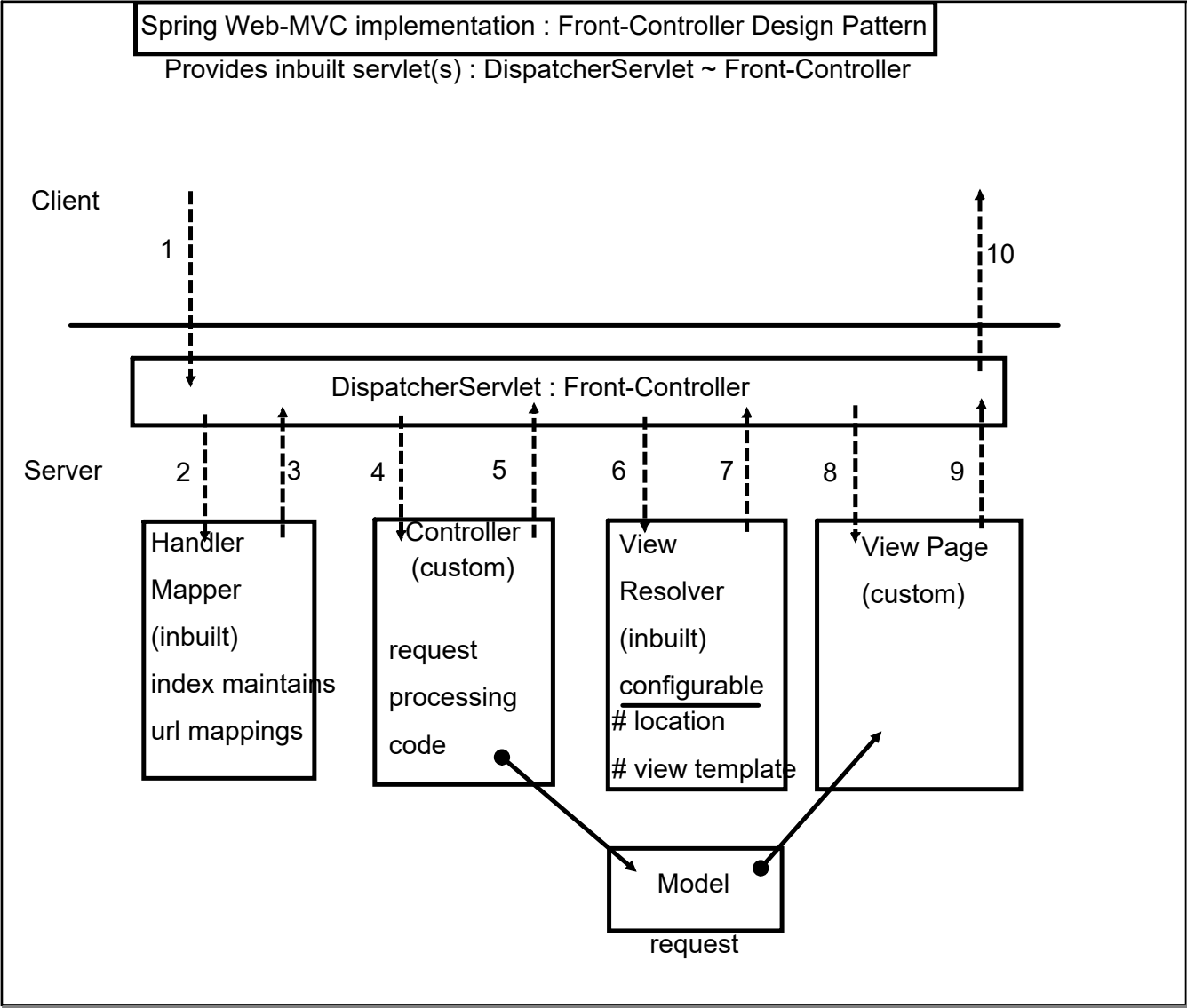 Class level   Method level

Spring web-mvc module : MVC architecture
uses Servlet-API : in an abstract
POJO

Controller : Servlet
View : JSP
Model : Data Structure

Controller : POJOs (Servlet capabilities)

View : Spring supports multiple view templates

     default : JSP + JSTL

     Thymeleaf

     Mustache

     FreeMarker

     Velocity

     Tiles

Model : Data Structure/Data Container

Spring Web-MVC implementation : Front-Controller Design Pattern

Provides inbuilt servlet(s) : DispatcherServlet ~ Front-Controller

Client

1   10

DispatcherServlet : Front-Controller

Server

2   3   4   5   6   7   8   9

Handler
Mapper
(inbuilt)
index maintains
url mappings

Controller
(custom)

request
processing
code

View
Resolver
(inbuilt)
configurable
location
view template

View Page
(custom)

Model

request

1. Need to register the DispatcherServlet

2. COnfig to target all requests to DS

web.xml (servlet-api)

Spring Based Config : (XML)

    + Bean Factory Config

    + web mvc config

View Resolver :

eg : "index" (string) name of view page (controller)

prefix : location

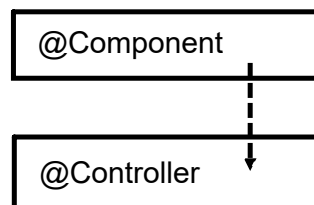suffix : View Template (extension)

<u>\<servlet-name\>-servlet.xml</u>

eg: dispatcher-servlet.xml

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

/WEB-INF/views/index.jsp

Custom Resources

1. Controller : POJO, registered with Handler Mapper

```
┌──────────────────┐
│ @Component       │
└──────────────────┘
          ┊
          ┊
          ▼
┌──────────────────┐
│ @Controller      │
└──────────────────┘
```

identifying the HTTP Verb

Mapping will take place using getter/setter only

Maven Project

    1. archetype : web

    2. Add the Server Runtime Library

    3. convert java 1.5   to 1.8

    4. Adding dependencies

        1. spring framework

        2. servlet for DS

        3. jsp+jstl

Pure Java Config

~ add a maven plugin
web.xml (servlet-api)
~ Java Class

dispatcher-servlet.xml (spring) ~ Java class

Java Class for web.xml

Registered DS (auto - inherit inbuilt class)

Mapped the url

Java Class for Spring config

component scanning path

exposed a bean of ViewResolver

Form handling spring-way : Forms are critical

Custom Tag Library : JSP

Need to add the reference of custom tag library

Spring forms : map the forms (UI) with java classes

control the form behavior (UI) through java classes

Validation : Validator API : Hibernate-Validator (dependency)

Client - Side Validation : submission takes place when all constraint are satisfied

HTML5 attribute + JS

Server-Side Validation :

Absolute URL : fetch the context path : predeclared variable in JSP to access the context path

Annotation : interface(skeleton) + implementation class (logic)

@EmployeeCode

Rule

1. Retention policy : compile/runtime

compile : @Override/@FunctionalInterface

runtime : validation
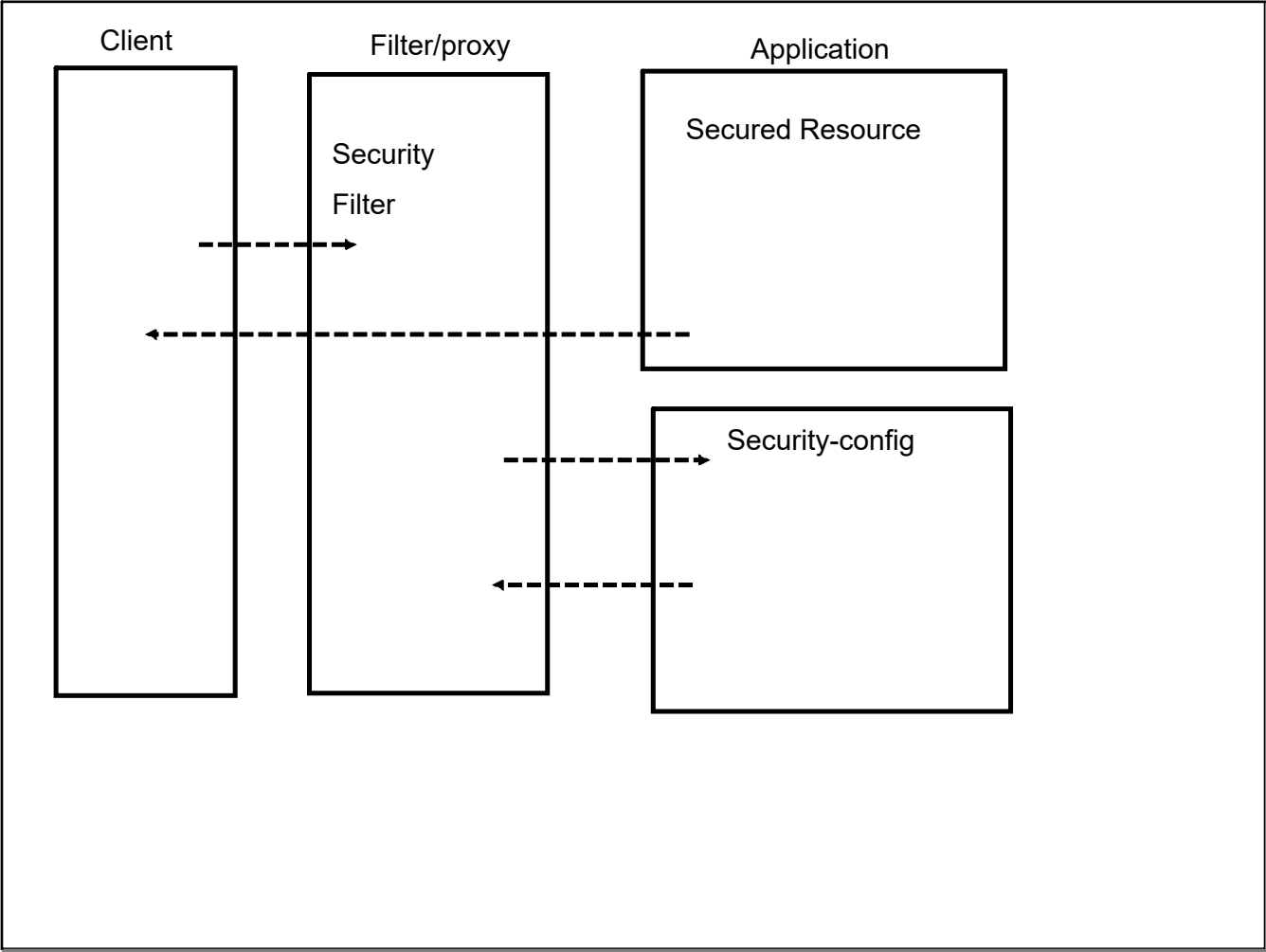
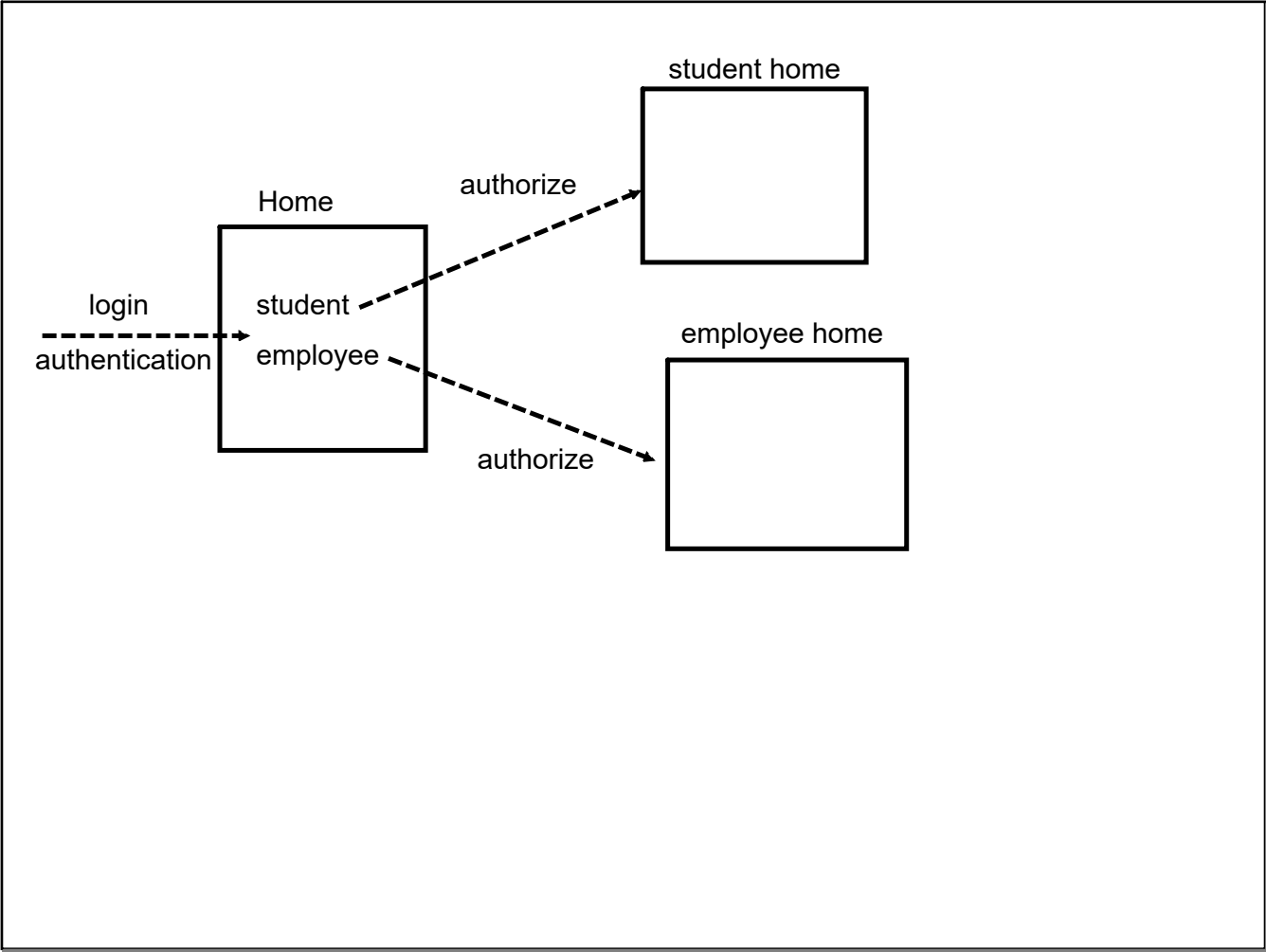2. Target : where that annotation can be used

method, class, field...

23

Spring - Security Module

authentication(valid) + authorization (role based)

Dependency:

> spring-security-web
>
> spring-security-config   Spring Security Module
>
> spring-security-taglibs

student home

Home

authorize

login

authentication

student

employee

employee home

authorize

1. Add Dependency

2. a class to activate security filter

3. a class to add config

default security (all resources are secured) :

provide an inbuilt login form

3 authentication ways

    1. httpBasic : not recommended

    2. formLogin (inbuilt)

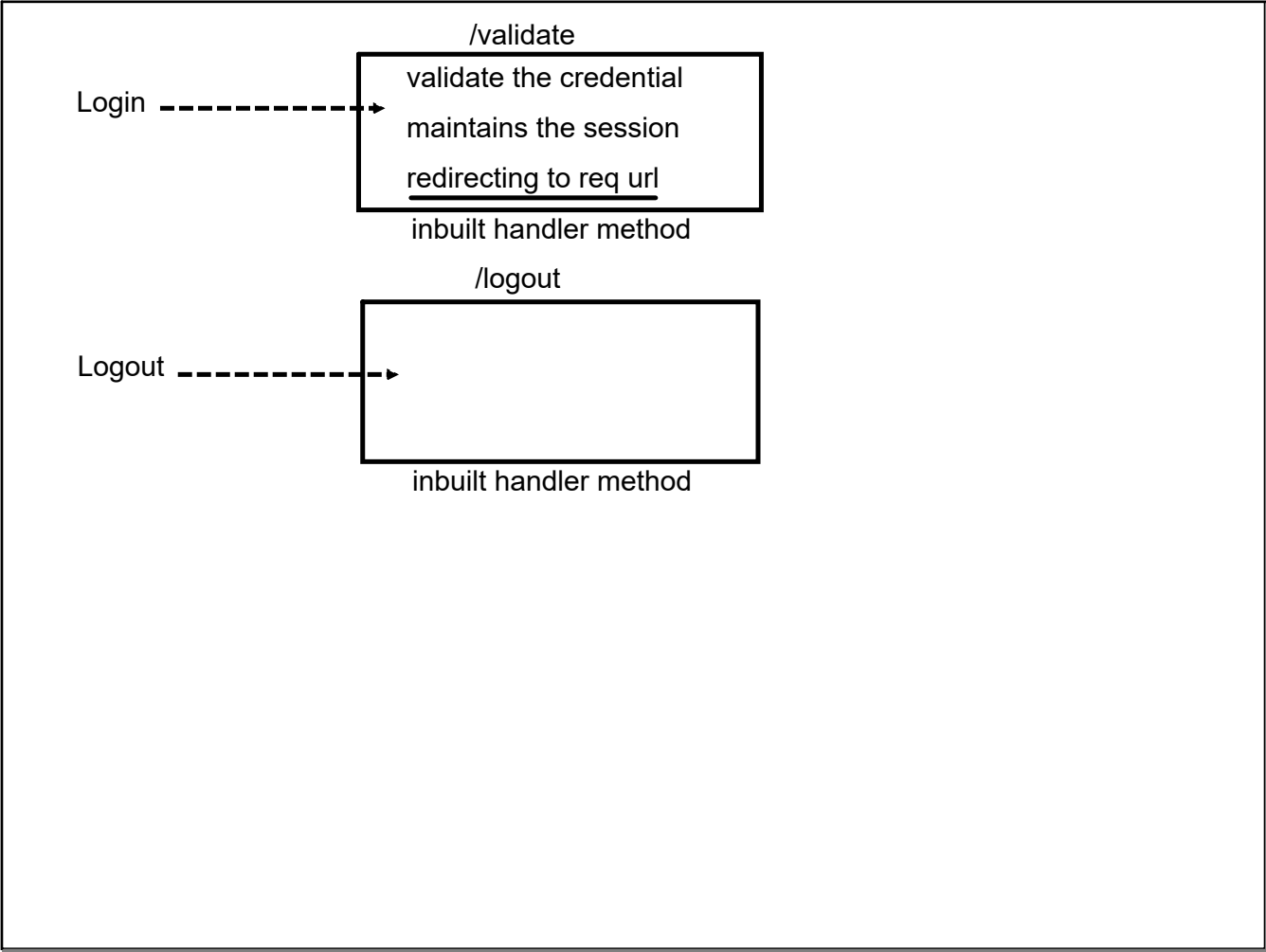   3. formLogin (custom)

form : spring-form

security over CSRF attack

spring security for every form , will pass a security token, while submission same token needed to sent back, checked by security filter

Every form must be spring-form

Custom login (spring-form)

    # login : username

    # password : password

    # method : POST

/validate

Login ------------→ | validate the credential

maintains the session

redirecting to req url

inbuilt handler method

/logout

Logout ------------→

inbuilt handler method

taglib of security

    1. get some info about logged in user