Spring MVC Framework

#Framework to develop web based app in Java based on MVC design pattern

#Spring Way of developing web app

-==>leveraging of core spring features

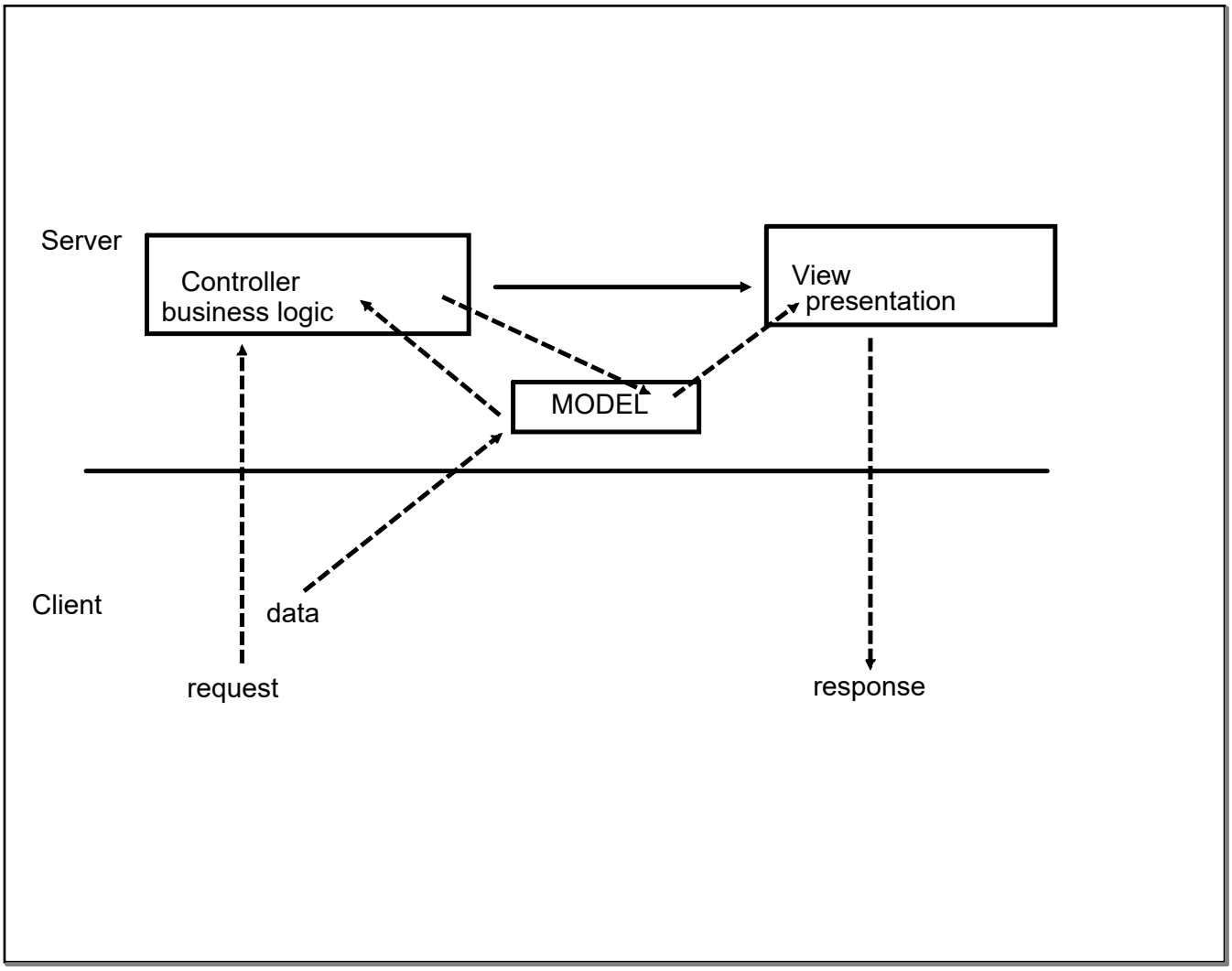
==>building UI

==>process form data

==>manage application state

==>Flexible config

MVC : seperation of concern


Server
Controller
business logic
View
presentation
MODEL
Client
data
request
response

Components of a Spring MVC Application

- # Collection of Spring bean (controllers,services,model....)
- # set of web pages (view) UI
- # Set of Spring config files

Java EE : servlet

Spring

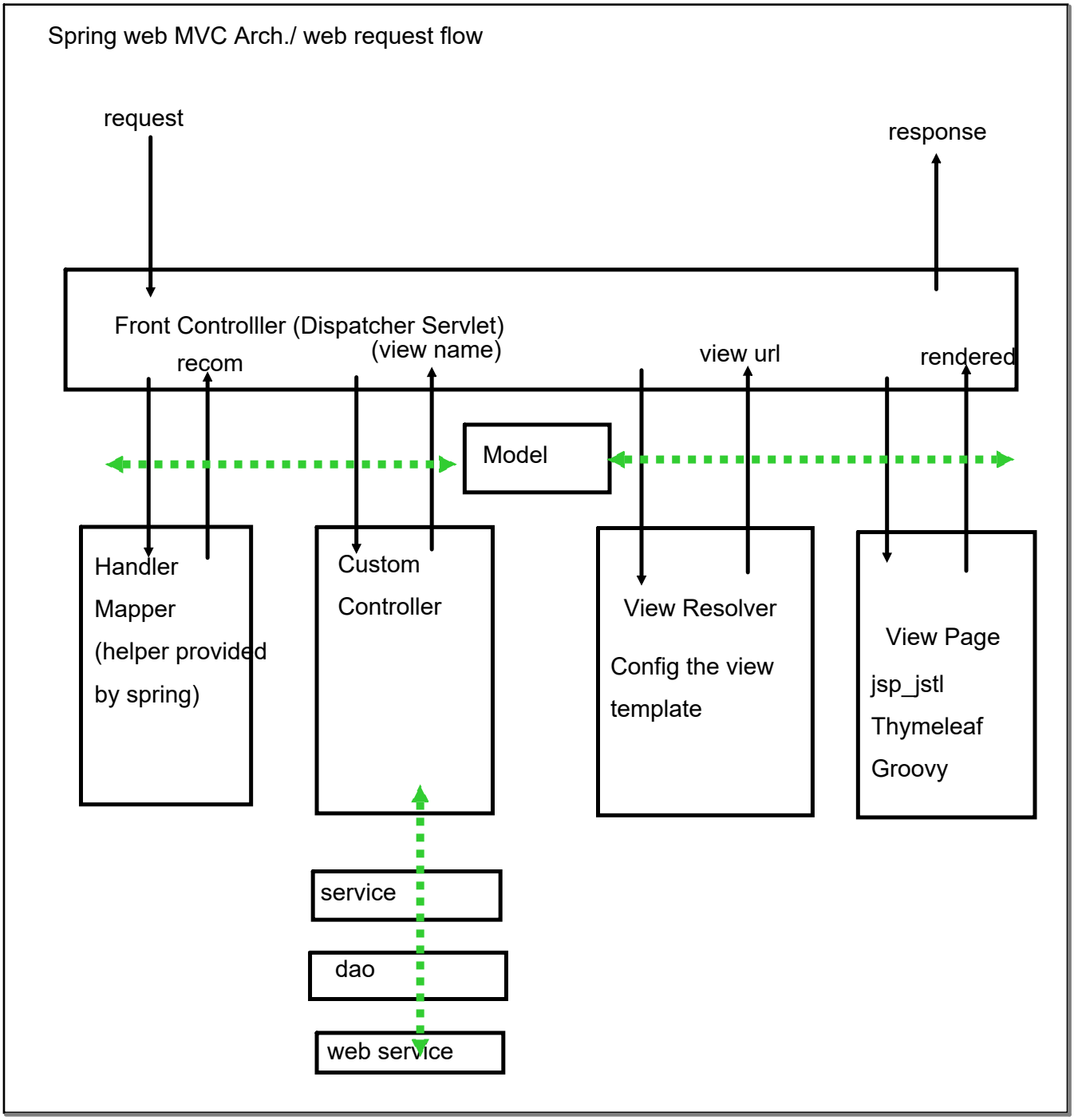
- # well defined MVC Framework
- # uses Servlet spec

Spring MVC : design pattern - Front Controller

Front Controller (Spring)

#single component that recieves all request

#Servlet - DispatcherServlet (part of Spring framework)


Spring web MVC Arch./ web request flow
request
response
Front Controlller (Dispatcher Servlet)
recom
(view name)
view url
rendered
Model
Handler
Mapper
(helper provided
by spring)
Custom
Controller
View Resolver
Config the view
template
View Page
jsp_jstl
Thymeleaf
Groovy
service
dao
web service

#Handler mapper (consultant)

#log/record of controller/url mapping

#finds the appropriate controller for current request

#provides its recommendation

#View Resolver (consultant)

log of view page

config of view templates

creates and returns an appropriate url of view page

Environment

#spring framework

#Tomcat

#View : jsp + jstl (add additional jar)

Config:

web.xml : register DS (standard servlet spec config)

Spring config : (every servlet can have its own personal config file to config supported resources)

config consultant, bean container, application context (web application context)

naming conventions : <servlet-name>-servlet.xml

eg: dispatcher-servlet.xml

dispatcher-servlet.xml

#add dependency of mvc tag

bean container : add support of component scanning

config the view resolver

property

prefix : location

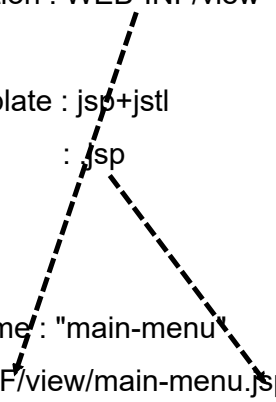
suffix : template

eg:

location : WEB-INF/view

template : jsp+jstl

: jsp



eg:

View name : "main-menu"

/WEB-INF/view/main-menu.jsp

MVC setup

#Create a Controller class

@Controller (inherited from @Component)

container : contain action/process method containing logic

#Create a controller method

add request mapping (url mapping) to method

return view name

create a view page...

Model implementation:

==>implicit model

==>explicit model

different mapping

different param handling

Form handling

Validation API

AOP

Every controller method is passed with servlet spec param

Special Spring annotation (wrapper around servlet objects)

Every controller method is passed with Model Obj (container), need to add data to that container (key-value), and it will be shared auto with view pages

View can access info using expression lang.

Mappings:

@RequestMapping can be applied at class level

#multiple url can be mapped with same method

#Request mapping can be classified based on http method

#can have a default method that maps with class level url

#can have a fallback method (gets called for bad url)

Info can be recieved as path variable

Controller method return types...

#ModelAndView

Spring MVC Forms Tags (Spring way):

can make use of data bindings

easier approach for validation

Auto access over java objects/beans

Security (CSRF)

Entity : Student

To connect/provide access of Student Object to form

#need to specify the spring namespace to access form tags

#need to specify the java jsp-jstl tag lib to access programmatic constructs

#need to add support for special config tags in config file (DS.xml): for properties files

Validation of client data:

Client side : client side scripts

Server side : Spring way

Spring way:

Validation API : Java Standard Bean Validation API (API for entity validation)

Available for server side app and client side (Swing/JavaFx)

Spring 4 and above have default support for validation API (Annotation)

API : Only a spec, not an implementation

popular impl : Hibernate validation (separate project) compliant JSR-303/309

Add jar files of impl

Process

==> Add Validation rules (annotations) to entity

==> Config the entry-form to display error message

==> Identify the validation status in controller method (server side)

use : Standard bean validation ann. (prevent vendor locking...)

#auto searched for available impl.....

Empty textbox is treated as empty String and not a null

Pre-processor method : interceptor to data submitted from client

==> remove all leading & trailing spaces

==>convert empty string to null

override the default mvc validation message

=> need to have properties file containing custom messages

=> config the DS.xml for prop file...

key : <errortype>.<object>.<fieldname>=<message>

eg: typeMismatch.student.freePasses=....

Custom Validation Constraints

#need to create a new annotation

add appropriate attributes

create a backup class containing the constraint logic

associate backup class with annotation

Pure Java Config

delete xml files

introduce the java class

replacement of web.xml

1. register/configure DS
2. url mapping

Java class : inherit inbuilt class (auto register the DS)

replacement of dispatcher-servlet.xml

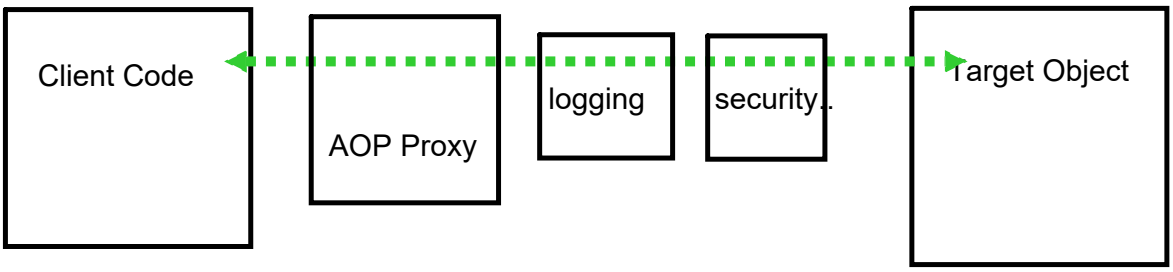
create a class

decorate with annotations

Aspect Oriented Programming (AOP)

==>cross-cutting concern

logging, security, transactions



separation of concern (mix the cross-cutting req with business code)

reusability

Configurable

AOP Support...

Two leading AOP framework for JAVA

1. Spring AOP
2. AspectJ

weaving : compile/load/run

Spring AOP :

simple AOP : run time only

can be applied at method level only

does not have complete support

still need a complete AOP Framework to be added : AspectJ

AspectJ

Original AOP framework (2001)

apply it : method-level, constructor, field level

weaving : all type

aspectjweaver

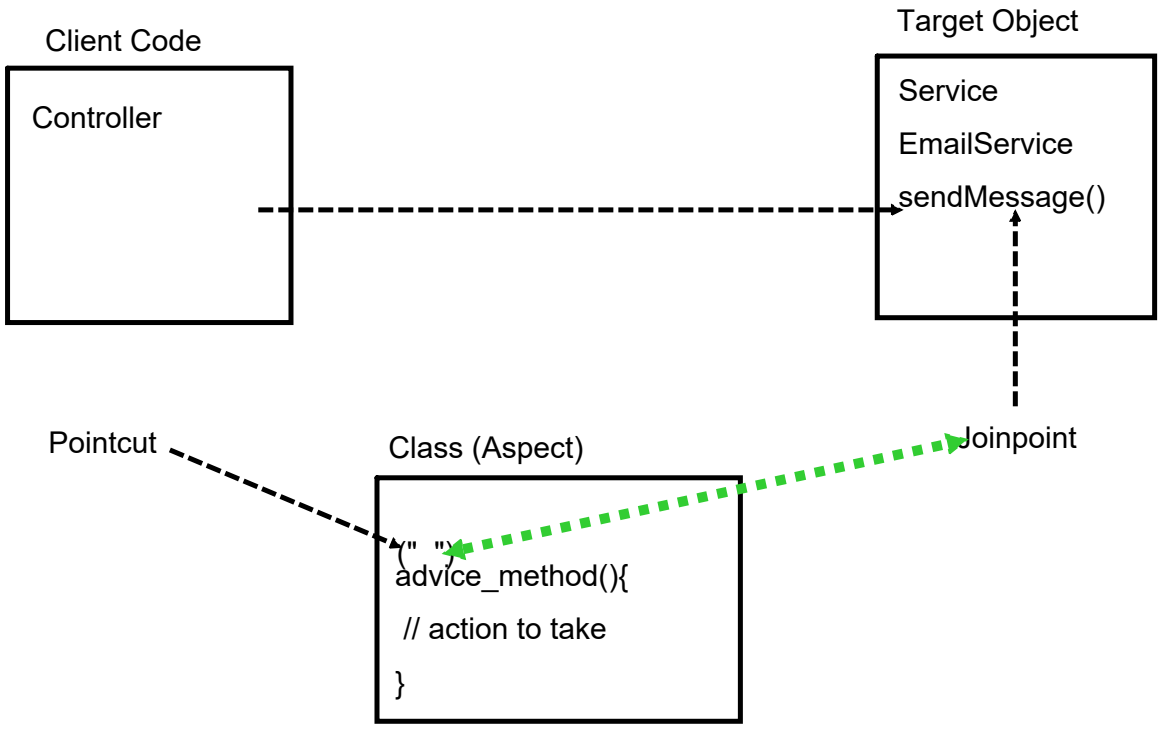
AOP Terminology

Aspect : module/class that holds diff cross-cutting

Advice : Action to be taken (method inside aspect class : when it should be applied as well)

JoinPoint : target object/method when the advice is to be applied

Pointcut : predicate expression/ pattern matching connects Advice and Joinpoint



Advice Type:

- Before : run before the joinpoint method
- After returning : run after method (success)
- After throwing : run after method (exception thrown)
- After finally : run after method (every case)
- Around Advice : run both before and after..

process:

- Target Object : eg : Messaging Service
- Create a special java class for config the AOP
- Create client code (controller)
- Create a special class (Aspect)

Pointcut Expression Language

execution(modifiers-pattern?
return-type-pattern
declaring-type-pattern?
method-name-pattern (param-pattern) throws-pattern?)

pattern with ? (optional)
can use wildcard (*)

exact match
@Before("execution(public String com.training.mvc.services.EmailService.sendMessage(String,String))")

match any sendMessage method in any class
"execution(public String sendMessage(String,String))"

match any method starting with send... in any class
"execution(public String send*(String,String))"

match any sendMessage method in any class with any access-modifier and any return type
"execution(* * sendMessage(String,String))" // "execution(* sendMessage(String,String))"

param-pattern

1. exact match eg : (String, String)
2. () : match method with no arg
3.(*) : method with one arg of any type
4.(..) : 0 or more arg of any type

Reusing Pointcut expression

have a pointcut declaration

multiple advices can be ordered...

Combining pointcut expression

apply multiple pointcut exp on a single advice

execute advice conditionally

Logical operators

&&

||

!

eg:

```
@Before("exp1() && exp2()")
@Before("exp1() || exp2()")
@Before("exp1() && !exp2()")
```

Use Case:

execute an advice on calling of any method of any class under services package

, leaving alone getter/setter methods

Advice method does have access over Joinpoint method

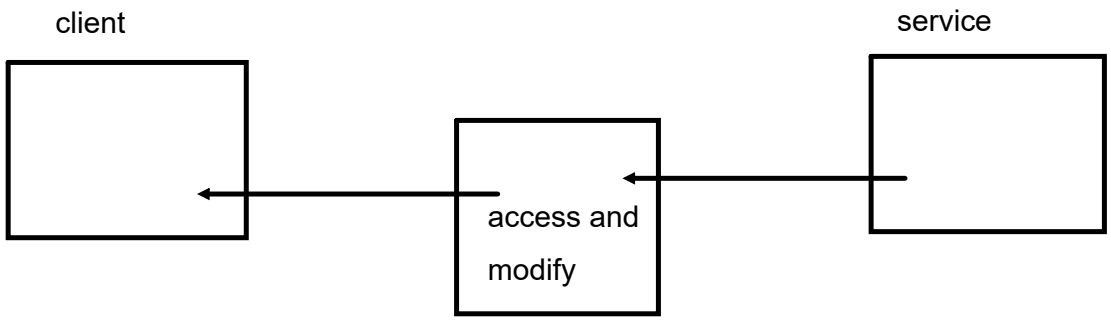
Access over method signature

Access over arg passed to joinpoint method

@AfterReturning :

Access over the returned value

can modify the returned result



@AfterThrowing Advice :

if an exception is thrown from joinpoint

access over exception type

@After finally :

cannot have access over returned value or exception

@Around

more fine grained control over joinpoint method

need to explicitly invoke the joinpoint method

1. Around encapsulate complete AOP of joinpoint

access and control over returned value / exception

Exception:

Consume, handle , rethrow the exception