Spring MVC

Spring MVC

- **Spring MVC is a popular framework which is implementing MVC Pattern**

- **DispatcherServlet is the heart of Spring MVC**
  - ✓ Implements the FrontController Pattern
  - ✓ It receives all the requests and delegates to other components

Sivaprasad.valluru@gmail.com

A SIMPLE MVC APP WITH OUT ANNOTATIONS

Sivaprasad.valluru@gmail.com

Web.xml configuration

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Simple Controller with out annotations

```
public class WelcomeController  implements Controller{

    @Override
    public ModelAndView handleRequest(HttpServletRequest request,
            HttpServletResponse response) throws Exception {

        ModelAndView modelAndView= new ModelAndView("/WEB-INF/jsp/welcome.jsp");

        modelAndView.addObject("message", "Welcome to  Spring MVC");

        return modelAndView;
    }
}
```

Spring-servlet.xml

```
<bean name="/welcome.htm" class="com.way2learnonline.WelcomeController"/>
```

- **/WEB-INF/jsp/welcome.jsp**

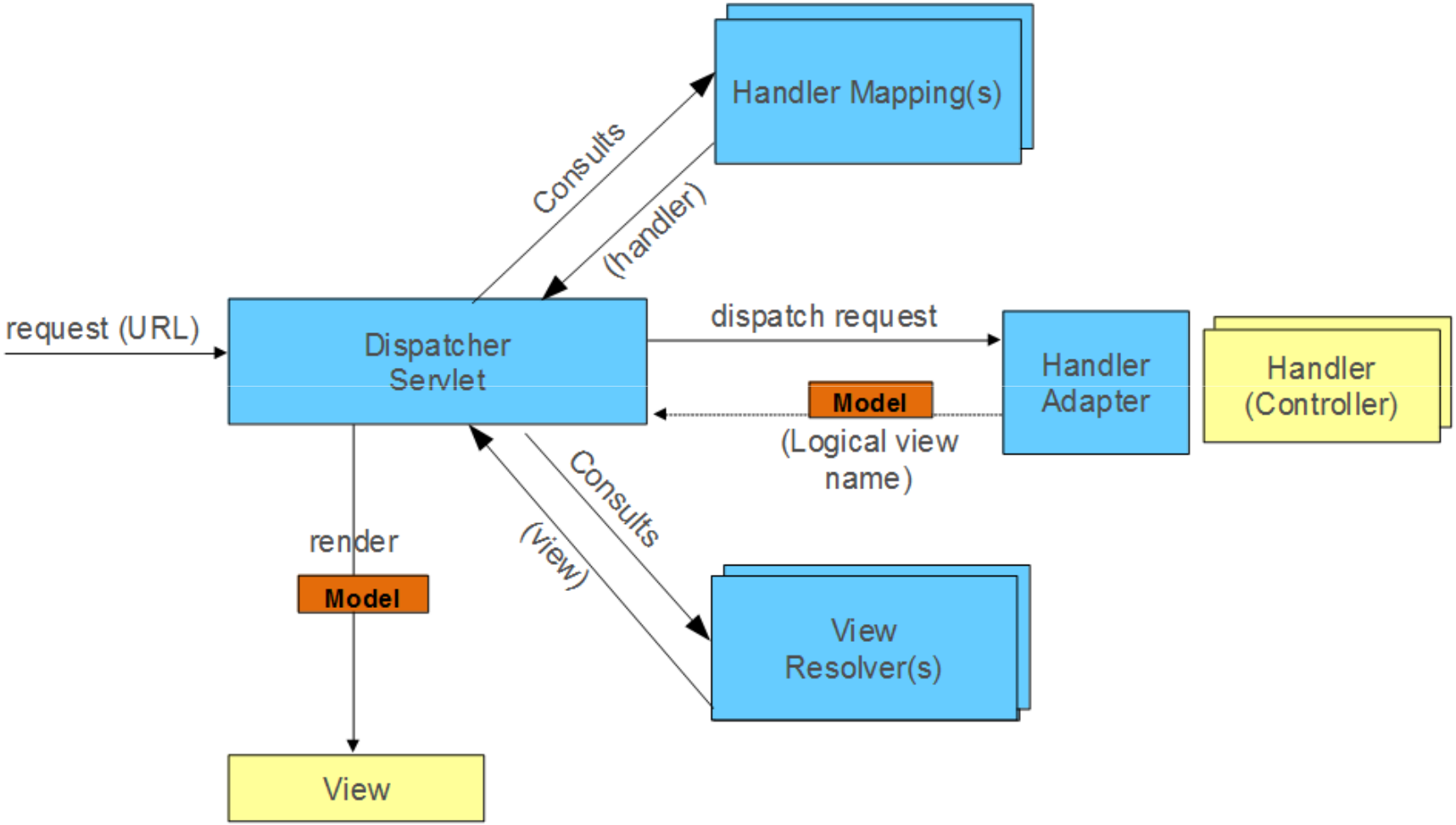
```
<html>
    <body>

        Hi !! ${message}

    </body>
</html>
```

Using View Resolver

- **Don't want to hard code the view name inside code.**
- **Want to give a logical name**
- **Solution :**
- **Configure a viewResolver as below :**

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

Flow of a request



Sivaprasad.valluru@gmail.com

LAB - MVC APP WITH OUT ANNOTATIONS

Sivaprasad.valluru@gmail.com

MVC USING ANNOTATIONS

Sivaprasad.valluru@gmail.com

Controller using Annotations

```
@Controller
public class WelcomeController {


    @RequestMapping(value="/welcome.htm")
    public String welcome(Model model){
        model.addAttribute("message", "Welcome to Spring MVC!!!");
        return "welcome";
    }

}
```

- **Component scanning has to be enabled**

Using ModelAndView as return type

- **Return type of method can be ModelAndView as shown below :**

```
@RequestMapping(value="/welcome.htm")
public ModelAndView welcome(){
    ModelAndView mav= new ModelAndView("welcome");
    mav.addObject("message", "Welcome to Spring @MVC");
    return mav;
}
```


Sivaprasad.valluru@gmail.com

- **Method can take HttpServletRequest, HttpServletResponse, HttpSession, Map, etc as show below :**

```
public String welcome(HttpServletRequest request,HttpServletResponse response,
                      HttpSession session, Map modelMap){
    modelMap.put("message", "Welcome to Spring MVC!!!");
    return "welcome";
}
```


Sivaprasad.valluru@gmail.com

Accessing Request Parameters

```
@RequestMapping(value="/welcome.htm")
public String welcome(@RequestParam String firstName, @RequestParam String lastName,
        Map<String,Object> modelMap){
    modelMap.put("message", "Welcome to Spring MVC!!!" +firstName +" "+lastName);
    return "welcome";
}
```

Optional Request Parameters

```
public String welcome(@RequestParam(value="fname",required=false) String firstName,
        @RequestParam String lastName,
        Map<String,Object> modelMap){
    modelMap.put("message", "Welcome to Spring MVC!!!" +firstName +" "+lastName);
    return "welcome";
}
```

Matrix Parameters

- **Matrix variables can appear in any path segment, each matrix variable separated with a ";"**
  - ✓ E.g: "/cars;color=red;year=2012"
- **Multiple values may be either ",“ separated "color=red,green,blue" or the variable name may be repeated "color=red;color=green;color=blue".**

```
// GET /pets/42;q=11;r=22

@RequestMapping(path = "/pets/{petId}", method = RequestMethod.GET)
public void findPet(@PathVariable String petId, @MatrixVariable int q) {

    // petId == 42
    // q == 11

}
```

```
// GET /owners/42;q=11/pets/21;q=22

@RequestMapping(path = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)
public void findPet(
        @MatrixVariable(name="q", pathVar="ownerId") int q1,
        @MatrixVariable(name="q", pathVar="petId") int q2) {

    // q1 == 11
    // q2 == 22

}
```

Optional matrix variable

```
// GET /pets/42

@RequestMapping(path = "/pets/{petId}", method = RequestMethod.GET)
public void findPet(@MatrixVariable(required=false, defaultValue="1") int q) {

    // q == 1

}
```


Sivaprasad.valluru@gmail.com

Obtaining all matrix variables in a MAP

```
// GET /owners/42;q=11;r=12/pets/21;q=22;s=23

@RequestMapping(path = "/owners/{ownerId}/pets/{petId}", method = RequestMethod.GET)
public void findPet(
        @MatrixVariable Map<String, String> matrixVars,
        @MatrixVariable(pathVar="petId"") Map<String, String> petMatrixVars) {

    // matrixVars: ["q" : [11,22], "r" : 12, "s" : 23]
    // petMatrixVars: ["q" : 11, "s" : 23]

}
```

Enabling Matrix Parameters

```
<mvc:annotation-driven enable-matrix-variables="true"/>
```

Specifying Request method in @RequestMapping

```
@RequestMapping(value="/welcome.htm",method=RequestMethod.GET)
```

```
@RequestMapping(value="/welcome.htm",method={RequestMethod.GET,RequestMethod.POST})
```


Sivaprasad.valluru@gmail.com

Mapping Request by presence of request params

```
@RequestMapping(value="/welcome.htm",params={"paramName"})
```

```
@RequestMapping(value="/welcome.htm",params={"firstName=Siva"})
```


Sivaprasad.valluru@gmail.com

Using @RequestMapping at class level

- **Can apply @RequestMapping at class or method level**
  - ✓ Class mapping is matched, then request

```
@Controller
@RequestMapping(value="/courses")
public class CourseController {


    @RequestMapping(value={"/viewIndividualCourse.htm","/s/viewIndividualCourse.htm"})
    public String viewIndividualCourse(@RequestParam("courseId")String courseId){

        return "viewIndividualCourse";
    }

    @RequestMapping(value = { "/browseCourses.htm","/s/browseCourses.htm","/s/myCourses.htm"})
    public String browseCourses(HttpSession session,HttpServletRequest request) {

        return "browseCoursesmain";
    }
```

Accessing Path Variables

```
@RequestMapping(value="/welcome/{fname}/{lname}")
public String welcome(@PathVariable("fname") String firstName,
        @PathVariable("lname") String lastName,
        Map<String,Object> modelMap){

    modelMap.put("message", "Welcome to Spring MVC!!!" +firstName +" "+lastName);
    return "welcome";
}
```

Using Regex with @PathVariable

```
@RequestMapping(value="/welcome/{id: [\\d]*}")
public String welcome(@PathVariable("id") String id,
        Map<String,Object> modelMap){

    modelMap.put("message", "Welcome to Spring MVC!!!" +id);
    return "welcome";
}
```


Sivaprasad.valluru@gmail.com

Formatting Request Parameters

```
@RequestMapping(value="/welcome/{dob}/{hourlyRate}")
public String welcome(@PathVariable("dob")@DateTimeFormat(iso=ISO.DATE)  Date dob,
        @PathVariable("hourlyRate")  @NumberFormat(style=Style.CURRENCY) Double hourlyRate,
        Map<String,Object> modelMap){

    modelMap.put("message", "Welcome to Spring MVC!!!" +dob + hourlyRate);
    return "welcome";
}
```

- **U need to add <mvc:annotation-driven />**
- **Otherwise the necessary formatters will not be registered**

The <mvc> Namespace

- **Mapping the *DispatcherServlet adds a servlet element to the URL***
  - ✓ /teller/accounts/3
- **Since Spring 3.0.4, *DispatcherServlet can map to /***

- **An additional element must be defined to pass requests to Application Server**
- **Used for static assets (images, javascript files, etc.)**
- **Only required when mapping to /**

  - ✓ <mvc:annotation-driven/>
  - ✓ <mvc:default-servlet-handler/>

Views Without a Controller

```
<mvc:annotation-driven/>

<mvc:view-controller path="/home" view-name="home"/>

<mvc:view-controller path="/welcome" view-name="welcome"/>
```

<mvc:annotation-driven>

- **Purpose:**
  - ✓ *Modernize Spring MVC*
  - ✓ Sets up default configuration beans
    - ✓ Handler mappings/adapters, exception resolvers etc
- ***Also enables new features since Spring 3.0***
  - ✓ Validation, formatting, and type conversion
  - ✓ *No longer 100% backwards compatible with Spring 2*

Sivaprasad.valluru@gmail.com

What beans do I get with <mvc:annotation-driven>?

- **See org.springframework.web.servlet.config.AnnotationDrivenBeanDefinitionParser javadoc**
  - ✓ This class implements <mvc:annotation-driven>

Sivaprasad.valluru@gmail.com

Performing Redirects

```
<mvc:redirect-view-controller redirect-url="http://www.google.com" path="/google"
context-relative="false" keep-query-params="false"/>
```

LAB- MVC USING ANNOTATIONS

Sivaprasad.valluru@gmail.com

Why ContextLoaderListener ?

- **Service Layer and DataAccess Layer should be loaded by a separate ApplicationContext .**

- **WHY?**

LAB- WAY2LEARN APP INTRODUCTION

Sivaprasad.valluru@gmail.com

Accessing Resources

- ***Use <mvc:resources> to fetch static resources***
  - ✓ provides a convenient way to serve static resources from locations other than the web application root, including locations on the classpath
  - ✓ Configures a **ResourceHttpRequestHandler**
  - ✓ The **cache-period** property may be used to set far future expiration headers

```
<mvc:resources mapping="/resources/**"
               location="/,/WEB-INF/resources/,classpath:resoruces"
               cache-period="31556926"/>
```


Sivaprasad.valluru@gmail.com

How to force the client to reload cached resources?

- **Step 1**
  - ✓ Create a properties file application.properties as below

```
application.version=1.0.0
```

- **Step 2**
  - ✓ Configure as follows

```
<util:properties id="applicationProps" location="/WEB-INF/application.properties"/>

<mvc:resources mapping="/resources-#{applicationProps['application.version']}/**"
 location="/WEB-INF/resources/"   cache-period="31556926"/>
```

- **Step 3**
  - ✓ In jsps, use the code similar to below :

```
<spring:eval expression="@applicationProps['application.version']" var="applicationVersion"/>

<spring:url value="resources-{applicationVersion}" var="resourceUrl">
    <spring:param name="applicationVersion" value="${applicationVersion}"/>
</spring:url>


<link rel="stylesheet" type="text/css" href='${resourceUrl}/css/common.css'>
```

LAB-ACCESSING STATIC RESOURCES

Sivaprasad.valluru@gmail.com

CONVENTION OVER CONFIGURATION

Sivaprasad.valluru@gmail.com

Conventions for Model Attribute Names

- **A model attribute name may be left unspecified**
  - ✓ A default name is selected
    - ✓ based on the concrete *type of the attribute*
    - ✓ arrays and collections: *type plus "List" suffix*

```
Account acc = accountManager.findAccount(accountNumber);
model.addAttribute(acc);          // Added as "account"

MonetaryAmount amount =
                 accountManager.annualInterest(accountNumber);
model.addAttribute(amount);       // Added as "monetaryAmount"

List<Account> accounts = accountManager.findAllAccounts();
model.addAttribute(accounts);     // Added as "accountList"
```


Sivaprasad.valluru@gmail.com

Convention for View Names

- **Request-handling methods may leave view name unspecified:**
  - ✓ return null or void
- **A default *logical view name is selected***
  - ✓ Via *RequestToViewNameTranslator*
  - ✓ Leading slash + extension removed from URL

```
@RequestMapping("/accounts/list.html")
public void list(Model model) { ... }
```

accounts/list

```
<mvc:view-controller path="/welcome"/>
```

welcome

Shortcut For Adding a Single Model Attribute

- **A single model attribute can simply be returned**

```
@RequestMapping("/accounts/list")
public List<Account> list() {
    // Added to Model as accountList
    return accountManager.findAllAccounts();
}
```

Model Attribute naming convention used

```
@RequestMapping("/accounts/list")
public @ModelAttribute("accounts")
                                List<Account> list() {
    // Added to Model as accounts
    return accountManager.findAllAccounts();
}
```

Optionally annotate return type with model attribute name

Sivaprasad.valluru@gmail.com

Accessing Request Data

- **Access URL without using HttpServletRequest?**
  - ✓ Use @Value and SpEL instead
  - ✓ Can access any request properties this way
    - ✓ request.requestURI, request.requestURL,
    - ✓ request.queryString, request.method ...

```
@RequestMapping(value="/accounts")
public Account showAccount(Model model,
    @Value("#{request.method}") String httpMethod,
    @Value("#{request.requestURI}") String url) {
      Logger.info(httpMethod + ": " + url);
      ...
}
```


Sivaprasad.valluru@gmail.com

Accessing Request Data

- **More useful annotations**
  - ✓ @CookieValue
  - ✓ @RequestHeader

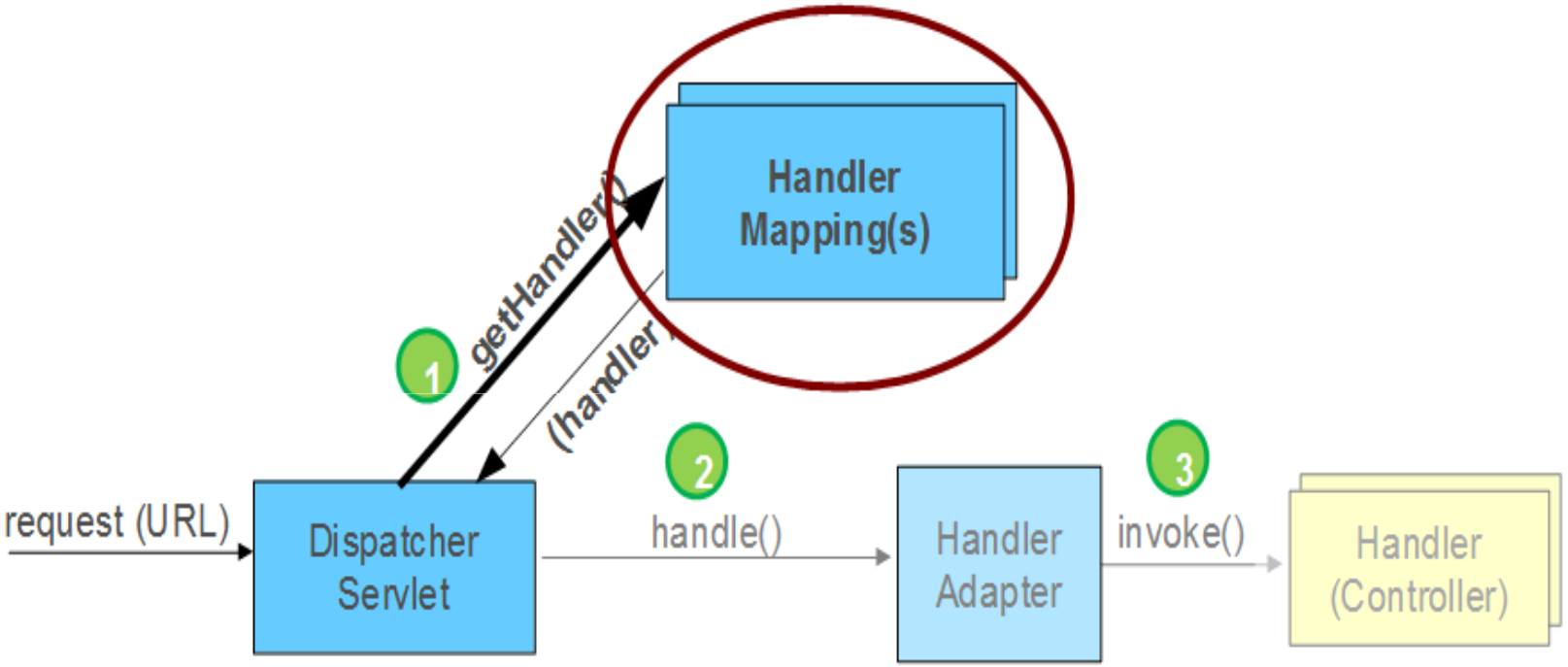
```
@RequestMapping(value="/orders/{id}",
method=RequestMethod.GET)
public String getOrder(Model model,
        @RequestHeader("user-agent") String browserType,
        @CookieValue("jsessionid") String sessionId)
{ … }
```


Sivaprasad.valluru@gmail.com

INFRASTRUCTURE BEANS

Sivaprasad.valluru@gmail.com

Locating A Controller



Sivaprasad.valluru@gmail.com

RequestMappingHandlerMapping

```
@Controller
@RequestMapping("/accounts")
public class AccountsController {

    @RequestMapping(method=RequestMethod.GET)
    public String list(Model model) { ... }
}
```

Class + method-relative
@RequestMapping

.../accounts

```
@Controller
public class AccountsController {

    @RequestMapping("/accounts")
    public String list(Model model) { ... }
}
```

Absolute method-level
@RequestMapping

.../accounts

Sivaprasad.valluru@gmail.com

ControllerClassNameHandlerMapping

- **Class name used to generate URL mappings**
  - ✓ HomeController → /home, /home/*
  - ✓ CoursesController → /courses, /courses/*

- **Method-level *@RequestMapping further narrows down a method within a controller***
  - ✓ 1st check HTTP request method (GET, POST, etc.)
  - ✓ 2nd check for presence of request parameter(s)
  - ✓ 3rd fall-back on the method name if necessary

Example Controller mapped by Convention

- **Mapped by using controller class name conventions plus method-level annotations**

Controller-level mappings:
/accounts, /accounts/*

```
@Controller
public class AccountsController {

    @RequestMapping(method=RequestMethod.GET)
    public void list(Model model) {
        ...
    }

    @RequestMapping(method=RequestMethod.GET)
    public void show(@RequestParam ("id") long accNum) {
        ...
    }
}
```

.../accounts/**list**

Method-relative @RequestMapping

.../accounts/**show**?id=123456789

Sivaprasad.valluru@gmail.com

Configuring more than one HandlerMapping

```
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">
    <property name="order" value="0" />
</bean>

<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping">
    <property name="order" value="1" />
</bean>
```

- **The first one to return a non-null value "wins"**

SimpleUrlHandlerMapping

```
<bean class="...SimpleUrlHandlerMapping">
    <property name="mappings">
        <value>
            /welcome=welcomeController
            /accounts/**=accountsController
        </value>
    </property>
</bean>
```

Controller-level mappings

```
@Controller
public class AccountsController {

    @RequestMapping(method=RequestMethod.GET)
    public String list(Model model) { ... }
}
```

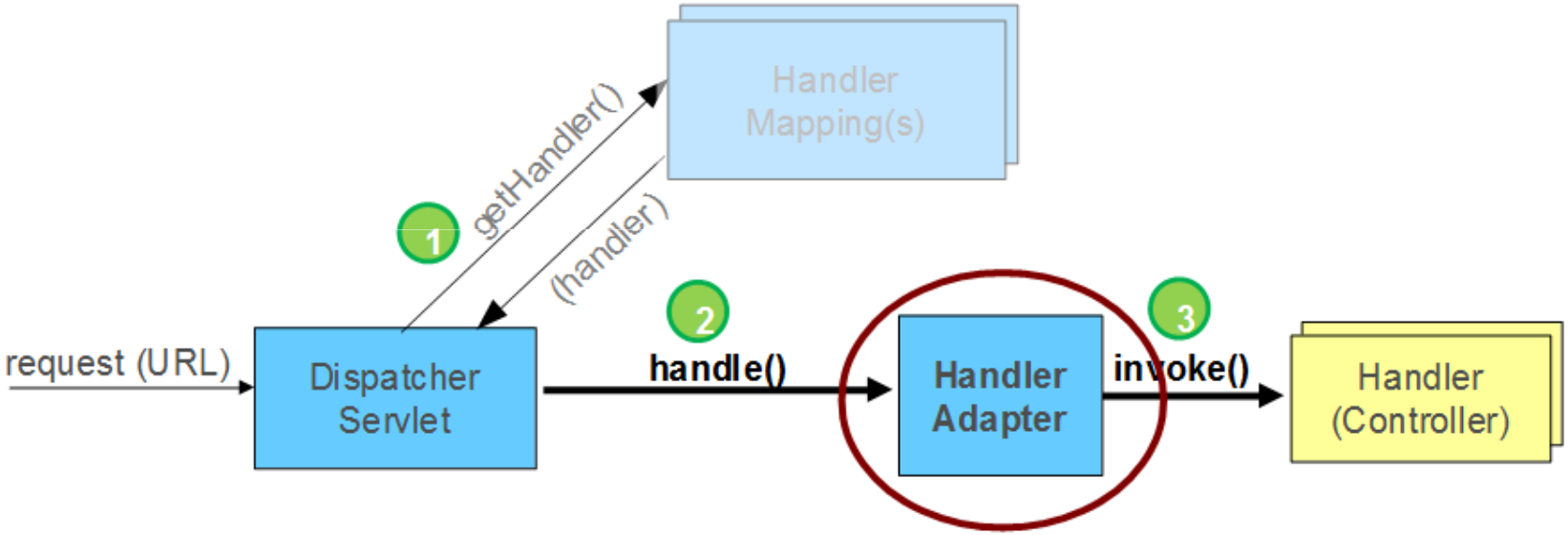
Method-relative @RequestMapping

.../accounts/list

HANDLER ADAPTERS

Sivaprasad.valluru@gmail.com

Controller Invocation through HandlerAdapter



Adapting Requests

- **DispatcherServlet invokes controllers (*handlers) using a HandlerAdapter***
- **Many types of handlers supported**
  - ✓ Such as @Controller classes
  - ✓ You may see others later
- **Separation of Concerns**
  - ✓ DispatcherServlet doesn't need to knowhow to invoke *any type of handler*
  - ✓ Allows new types of handler to be added
  - ✓ Default adapters provided out-of-the box

Sivaprasad.valluru@gmail.com

RequestMappingHandlerAdapter

- **Adapts calls to @RequestMapping methods**
- **Enables flexible method signatures**
  - ✓ Introspects input arguments
    - ✓ Supports HttpServletRequest, HttpServletResponse, HttpSession, Principal, Model *plus more*
- **Interprets output values**
  - ✓ Supports String (view-name), ModelAndView, ModelMap, Model, any other object as model-attribute
- **Configured by default**

URI Template Parameters

- **RequestMappingHandlerAdapter now provides extra support for PathVariables**
  - ✓ Automatically added to model
  - ✓ Can be used in a redirect request

```
@RequestMapping(value="/people/{firstName}/{lastName}")
public String peopleSearch(Model model,
        @PathVariable("firstName") String firstName,
        @PathVariable("lastName")  String lastName) {

   assert model.asMap().get("firstName").equals(firstName);
   assert model.asMap().get("lastName").equals(lastName);

   return "redirect:/person/{firstName}/{lastName}";
}
```


Sivaprasad.valluru@gmail.com

More Adapters

- **Other adapters exist to support other types of handlers**
  - ✓ WebflowHandlerAdapter
    - ✓ Passes web-flow requests to Web Flow Executor to run
- **HttpRequestHandlerAdapter**
  - ✓ Passes requests to HttpInvokerExporter (Spring Remoting)
- **Several adapters are created automatically when MVC initializes**

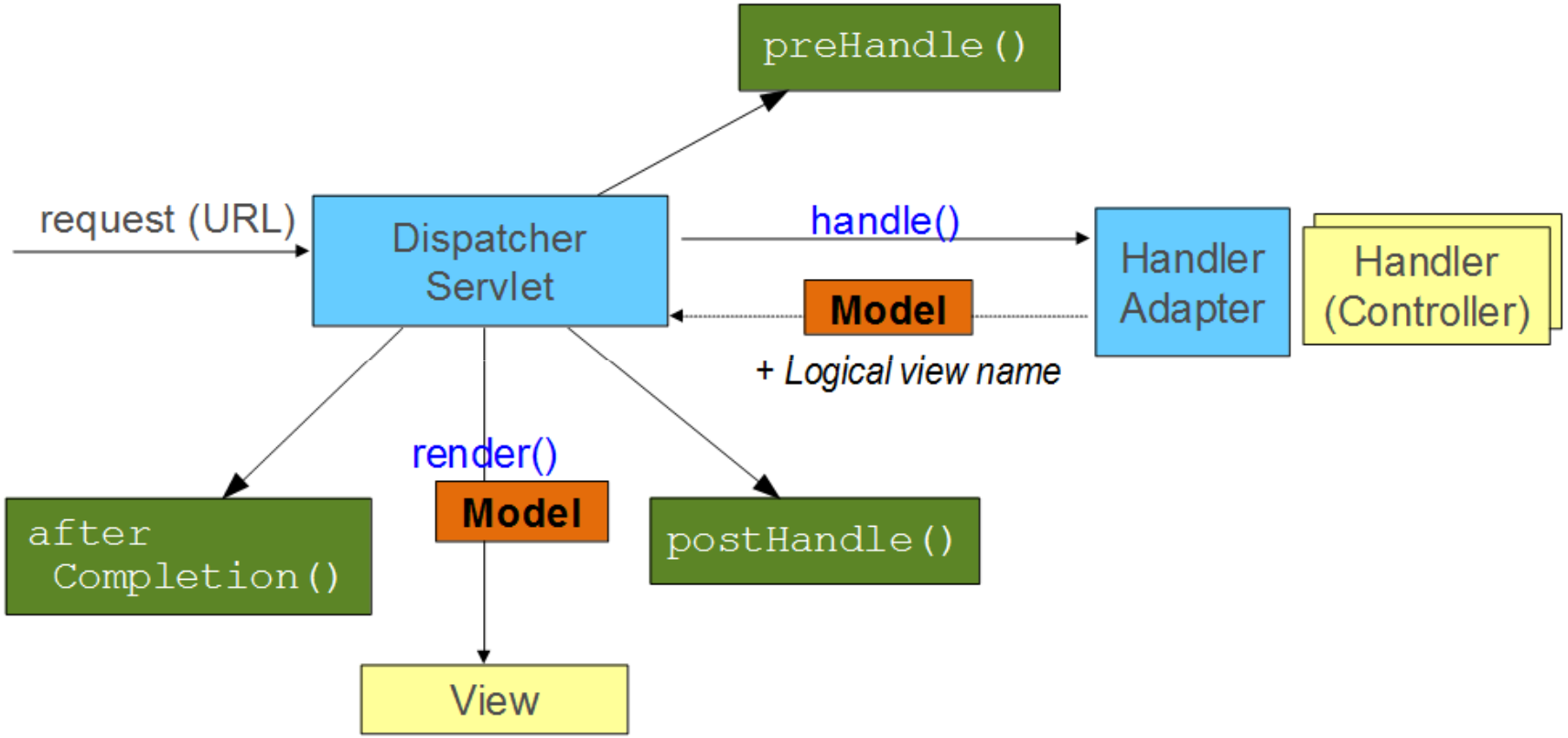
Sivaprasad.valluru@gmail.com

Handler Interceptors

Many useful cases

- Add common model attributes (menus, preferences)
- Set response headers
- Audit requests
- Measure performance (controller vs. rendering time)

Handler Interceptor Methods



Sivaprasad.valluru@gmail.com

Configuring Interceptors

- **Request interceptors are configured at the level of the HandlerMapping**

```
<bean id="measurementInterceptor" class="in.co.way2learn.MeasurementInterceptor" />

<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
    <property name="interceptors">
        <list>
            <ref bean="measurementInterceptor"/>
        </list>
    </property>
</bean>
```

Simplifying Interceptor Configuration

```
<mvc:interceptors>
    <bean class="in.co.way2learn.controllers.MeasurementInterceptor"/>
</mvc:interceptors>
```

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/secure/*"/>
        <mvc:exclude-mapping path="/secure/help"/>
        <bean class="in.co.way2learn.controllers.MeasurementInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
```

LAB – APPLYING INTERCEPTORS

Sivaprasad.valluru@gmail.com

MVC- JAVA CONFIGURATION

Sivaprasad.valluru@gmail.com

Simplifying configuration with @EnableWebMvc

- **Makes JavaConfig do <mvc:annotation-driven>**

```
@EnableWebMvc
@ComponentScan(basePackages={"com.way2learnonline.controllers"})
@Configuration
public class MvcConfig  {


    @Bean
    public ViewResolver viewResolver(){
        InternalResourceViewResolver viewResolver= new InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/view/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }


}
```


Sivaprasad.valluru@gmail.com

Customizing @EnableWebMvc

- **Implement WebMvcConfigurer**
  - ✓ Easier to extend WebMvcConfigurerAdapter
    - ✓ Override methods to customize
    - ✓ Can configure the default beans
    - ✓ Or specify your own beans as well/instead
- **Sample methods**
  - ✓ addInterceptors to specify your own interceptors
  - ✓ addResourceHandlers to serve static resources from the classpath
  - ✓ ConfigureDefaultServletHandling (next slide)
  - ✓ getValidator to define your own custom validator

customized WebMvcConfigurerAdapter

```
@EnableWebMvc
@ComponentScan(basePackages={"com.way2learnonline.controllers"})
@Configuration
public class MvcConfig extends WebMvcConfigurerAdapter {

    @Override
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**")
            .addResourceLocations("/WEB-INF/resources/");
    }

}
```

```java
@Override
public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/home").setViewName("homepage");
    registry.addRedirectViewController("/old", "/new");
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new MeasurementInterceptor())
            .addPathPatterns("/secure/*")
            .excludePathPatterns("/secure/help/*");
}
```

Configuring in web.xml -1

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>com.way2learnonline.RootConfig</param-value>
</context-param>

<context-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
</context-param>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

Configuring in web.xml -2

```
<servlet>
  <servlet-name>dispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>com.way2learnonline.MvcConfig</param-value>
  </init-param>
   <init-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

SUPPORT FOR SERVLET 3.X

Sivaprasad.valluru@gmail.com

SpringServletContainerInitializer

- **SpringServletContainerInitializer**
  - ✓ A bootstrap class that hooks into the Servlet 3.0 onStartup mechanism
  - ✓ Looks, in turn, for classes that implement Spring MVC's WebApplicationInitializer interface
  - ✓ Implement onStartup() to register Spring components
  - ✓ Best of all: no web.xml

- ***How it works:***
  - ✓ Servlet 3.0 uses Java SPI mechanism – expects the file /META-INF/services/javax.servlet.ServletContainerInitializer
  - ✓ Contains one line – FQCN of *SpringServletContainerInitializer*

Servlet 3 Configuration – no web.xml or Spring XML

```
public class MyWebAppInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext)
            throws ServletException {

        AnnotationConfigWebApplicationContext rootContext= new AnnotationConfigWebApplicationContext();
        rootContext.register(AppConfig.class);

        servletContext.addListener(new ContextLoaderListener(rootContext));


        AnnotationConfigWebApplicationContext dispatcherContext= new AnnotationConfigWebApplicationContext();
        dispatcherContext.register(CustomMvcConfig.class);

        ServletRegistration.Dynamic dispatcherServlet=
                servletContext.addServlet("dispatcherServlet", new DispatcherServlet(dispatcherContext));
        dispatcherServlet.setLoadOnStartup(1);
        dispatcherServlet.addMapping("/");

        servletContext.setInitParameter("resourcePath",
                "http://localhost:8080/02mvc-withoutwebxml/resources");
    }

}
```

- Instead of we writing all these logic to register dispatcher servlet and ContextLoaderListener, there is already a class AbstractAnnotationConfigDispatcherServletInitializer
- you can just extend this as shown below :

```
public class WebInitializer extends
        AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{RootConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return  new Class[]{MvcConfig.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }

}
```

WAY2LEARN

LAB- MVC USING JAVA CONFIGURATION WITH OUT WEB.XML

Sivaprasad.valluru@gmail.com

SPRING BOOT INTRODUCTION

Sivaprasad.valluru@gmail.com

What is Spring boot ?

- **Spring Applications typically require a lot of setup**

- **Consider working with JPA. You need:**
  - ✓ Datasource, TransactionManager, EntityManagerFactory, etc.
- **Consider a web MVC app. You need:**
  - ✓ WebApplicationInitializer / web.xml, ContextLoaderListener, DispatcherServlet, etc.
- **An MVC app using JPA would need all of this**

- **Much of this is predictable**

What is Spring Boot ?

- **Spring Boot uses sensible defaults, mostly based on the classpath contents.**

- **E.g.:**
  - ✓ Sets up a JPA Entity Manager Factory if a JPA implementation is on the classpath.
  - ✓ Creates a default Spring MVC setup, if Spring MVC is on the classpath.
- **Everything can be overridden easily**

Maven Dependencies

- **Spring Boot defines parent POM and "starter" POMs**

```
 <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>1.2.5.RELEASE</version>
   </parent>

<dependencies>
     <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter</artifactId>
   </dependency>

   <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-test</artifactId>
   </dependency>

   <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-jdbc</artifactId>
   </dependency>

   <dependency>
       <groupId>mysql</groupId>
       <artifactId>mysql-connector-java</artifactId>
   </dependency>
</dependencies>
```

Spring Boot Parent POM

- **Parent POM defines key versions of dependencies and Maven plugin**
  - ✓ Ultimately optional – you can have your own parent POM

Spring Boot works with Maven, Gradle, Ant/Ivy

Sivaprasad.valluru@gmail.com

Spring Starter POMs

- **Allow an easy way to bring in multiple coordinated dependencies**
  - ✓ "Transitive" Dependencies

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>
</dependencies>
```

Resolves ~ 16 JARs!
- spring-boot-*.jar
- spring-core-*.jar
- spring-context-*.jar
- spring-aop-*.jar
- spring-beans-*.jar
- aopalliance-*.jar
- etc.

Version not needed!
Defined by parent.

Spring Boot application starters

- ✓ spring-boot-starter
- ✓ spring-boot-starter-aop
- ✓ spring-boot-starter-batch
- ✓ spring-boot-starter-data-jpa
- ✓ spring-boot-starter-jdbc
- ✓ spring-boot-starter-jersey
- ✓ spring-boot-starter-test
- ✓ etc

Sivaprasad.valluru@gmail.com

@EnableAutoConfiguration

- **@EnableAutoConfiguration causes Spring Boot to automatically create beans it thinks are needed**
  - ✓ Usually based on classpath contents
  - ✓ But you can easily override

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class BankApplication {

    public static void main(String[] args) throws Exception {
        ApplicationContext context=SpringApplication.run(BankApplication.class);
        BankService bankService=context.getBean(BankService.class);
        bankService.transfer(new Long(1), new Long(2),1000);

    }

}
```

@SpringBootApplication

- **Very common to use @EnableAutoConfiguration, @Configuration, and @ComponentScan together.**

- **Boot 1.2 combines these with @SpringBootApplication**

```
@SpringBootApplication
public class BankApplication {

    public static void main(String[] args) throws Exception {
        ApplicationContext context=SpringApplication.run(BankApplication.class);
        BankService bankService=context.getBean(BankService.class);
        bankService.transfer(new Long(1), new Long(2),1000);

    }

}
```

Externalized Configuration

- **Spring Boot allows you to externalize your configuration so you can work with the same application code in different environments.**

- **To externalize configuration ,You can use**
  - ✓ properties files,
  - ✓ YAML files,
  - ✓ environment variables
  - ✓ command-line arguments.

Property values can be injected directly into your beans using the @Value annotation, accessed via Spring's Environment

Sivaprasad.valluru@gmail.com

PropertySource order

- ✓ Command line arguments.
- ✓ JNDI attributes from java:comp/env.
- ✓ Java System properties (System.getProperties()).
- ✓ OS environment variables.
- ✓ Profile-specific application properties outside of your packaged jar (application-{profile}.properties and YAML variants)
- ✓ Profile-specific application properties packaged inside your jar (application-{profile}.properties and YAML variants)
- ✓ Application properties outside of your packaged jar (application.properties and YAML variants).
- ✓ Application properties packaged inside your jar (application.properties and YAML variants).
- ✓ @PropertySource annotations on your @Configuration classes.
- ✓ Default properties (specified using SpringApplication.setDefaultProperties).

Accessing command line properties

- By default SpringApplication will convert any command line option arguments (starting with '--', e.g. --server.port=9000) to a property and add it to the Spring Environment.

- command line properties always take precedence over other property sources.

Sivaprasad.valluru@gmail.com

Application Properties files

- **SpringApplication will load properties from application.properties files in the following locations and add them to the Spring Environment:**

  - ✓ A /config subdir of the JVM's working directory.
  - ✓ The JVM's working directory
  - ✓ A classpath /config package
  - ✓ The classpath root

- **If you don't like application.properties as the configuration file name you can switch to another by specifying a spring.config.name environment property.**

  - ✓ java -jar myproject.jar --**spring.config.name**=myproject

- **You can also refer to an explicit location using the spring.config.location environment property**

  - ✓ java -jar myproject.jar --**spring.config.location**=classpath:/default.properties,classpath:/override.properties

Configuring DataSource Bean using Environment

```
@SpringBootApplication
@PropertySource("classpath:db.properties")
public class BankApplication {

    @Autowired
    private Environment env;

    @Bean
    public DataSource dataSource(){
        BasicDataSource dataSource= new BasicDataSource();
        dataSource.setDriverClassName(env.getProperty("db.driverclassname"));
        dataSource.setUrl(env.getProperty("db.url"));
        dataSource.setUsername(env.getProperty("db.user"));
        dataSource.setPassword(env.getProperty("db.password"));

        return dataSource;
    }

    public static void main(String[] args) throws Exception {
        ApplicationContext context=SpringApplication.run(BankApplication.class,args);
        BankService bankService=context.getBean(BankService.class);
        bankService.transfer(new Long(1), new Long(2),1000);

    }

}
```

Without using Environment

```
@SpringBootApplication
@PropertySource("classpath:db.properties")
public class BankApplication {

    @Bean
    public DataSource dataSource(@Value("${db.driverclassname}") String driverClassName,
            @Value("${db.url}") String url,
            @Value("${db.user}") String userName,
            @Value("${db.password}") String password    ){
        BasicDataSource dataSource= new BasicDataSource();
        dataSource.setDriverClassName(driverClassName);
        dataSource.setUrl(url);
        dataSource.setUsername(userName);
        dataSource.setPassword(password);

        return dataSource;
    }

    public static void main(String[] args) throws Exception {
        ApplicationContext context=SpringApplication.run(BankApplication.class,args);
        BankService bankService=context.getBean(BankService.class);
        bankService.transfer(new Long(1), new Long(2),1000);

    }

}
```

@ConfigurationProperties

```
@Bean
@ConfigurationProperties(prefix="db")
public DataSource dataSource(){
    BasicDataSource dataSource= new BasicDataSource();

    return dataSource;
}
```


Sivaprasad.valluru@gmail.com

With out any datasource bean configuration

application.properties

```
spring.datasource.url=jdbc:mysql://localhost/springdb
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
```

- **Default datasource prefix is spring.datasource**

- **See DataSourceAutoConfiguration**

Preference

- **What if I pass properties through command line ?**

- **Will Command Line args take precedence over PropertySource?**

Embedded Database Support

- Spring Boot can auto-configure embedded H2, HSQL and Derby databases.

- You don't need to provide any connection URLs, simply include a build dependency to the embedded database that you want to use.

```
<dependency>
    <groupId>org.hsqldb</groupId>
    <artifactId>hsqldb</artifactId>
    <scope>runtime</scope>
</dependency>
```

Typesafe Configuration Properties

```java
@Component
@ConfigurationProperties(prefix="connection")
public class ConnectionSettings {

    private String username;

    private InetAddress remoteAddress;

    // ... getters and setters

}
```

```yaml
# application.yml

connection:
    username: admin
    remoteAddress: 192.168.1.1
```

```java
@Service
public class MyService {

    @Autowired
    private ConnectionSettings connection;
```

```java
@Configuration
@EnableConfigurationProperties(ConnectionSettings.class)
public class MyConfiguration {
}
```

LAB : USING SPRING BOOT

Sivaprasad.valluru@gmail.com

MVC- USING SPRING BOOT

Sivaprasad.valluru@gmail.com

BootStraping DispatherServlet with out Boot

- **With out boot, using JavaConfig, we have bootstrapped DispatherServlet as below :**

```
public class WebInitializer extends
        AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{RootConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return  new Class[]{MvcConfig.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }

}
```

BootStrapping WebApp with Spring Boot

- ✓ Do we need to BootStrap DispatcherServlet manually when using Spring Boot ?
  - ✓ See DispatcherServletAutoConfiguration
- ✓ Just using @EnableAutoConfiguration will bootstrap DispatcherServlet.

```
public class Way2LearnApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(MvcConfig.class,RootConfig.class);
    }

}
```


Sivaprasad.valluru@gmail.com

AutoConfiguration of InternalResourceViewResolver

- **Just have to define following in application.properties**
  - ✓ spring.view.prefix:/WEB-INF/view/
  - ✓ spring.view.suffix:.jsp
-
- **See WebMvcAutoConfiguration and Observe how InternalResourceViewResolver is defined.**

Adding ResourceHandlers

- **Open WebMvcAutoConfiguration and observe addResourceHandler() method**

- **/** is mapped to below locations :**
  - ✓ "classpath:/META-INF/resources/", "classpath:/resources/", "classpath:/static/", "classpath:/public/"
- **So, Static resources have to be kept under above locations to be auto detected.**

LAB – SPRING MVC USING BOOT

Sivaprasad.valluru@gmail.com

SPRING -TILES

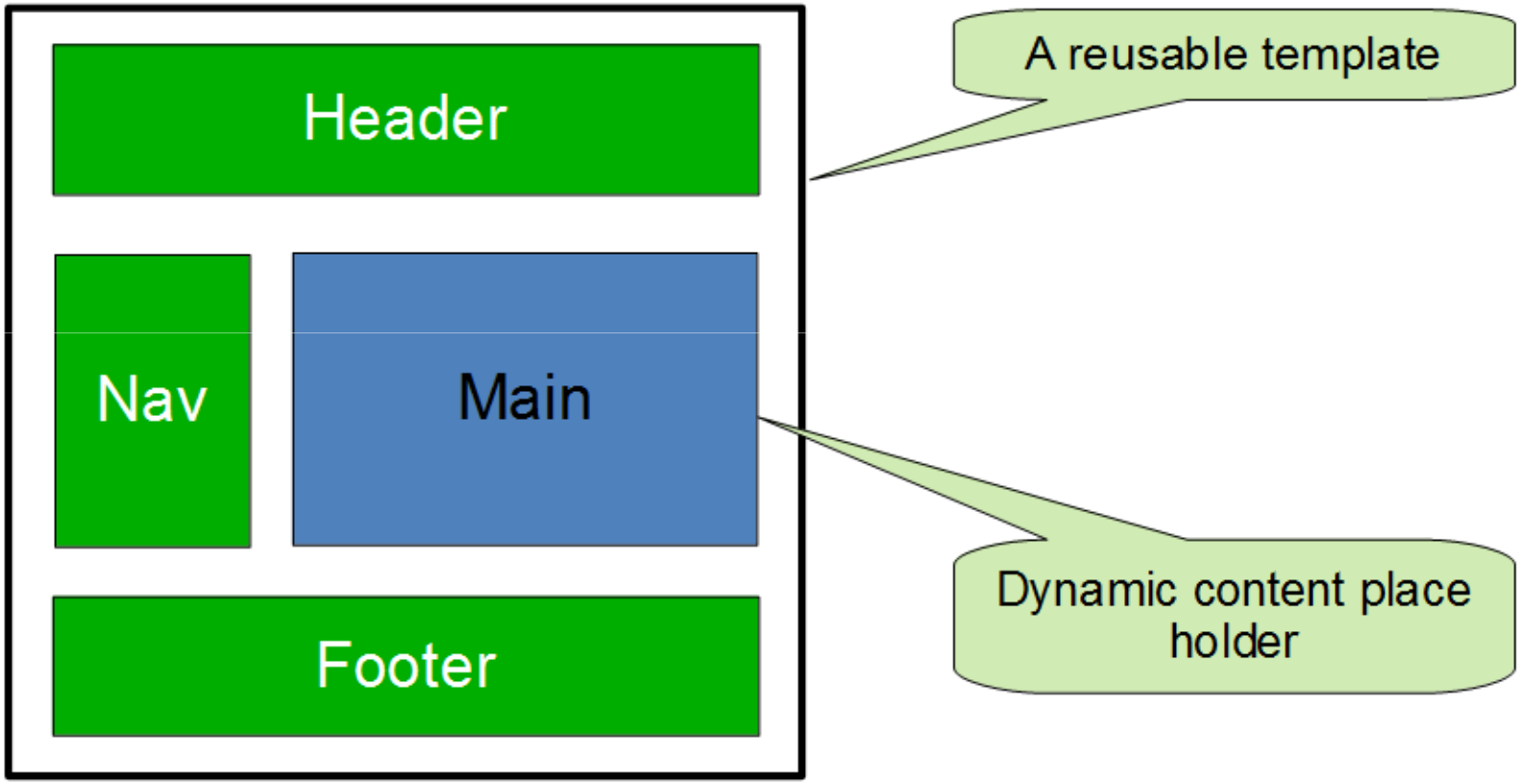
Sivaprasad.valluru@gmail.com

The Composite View Pattern

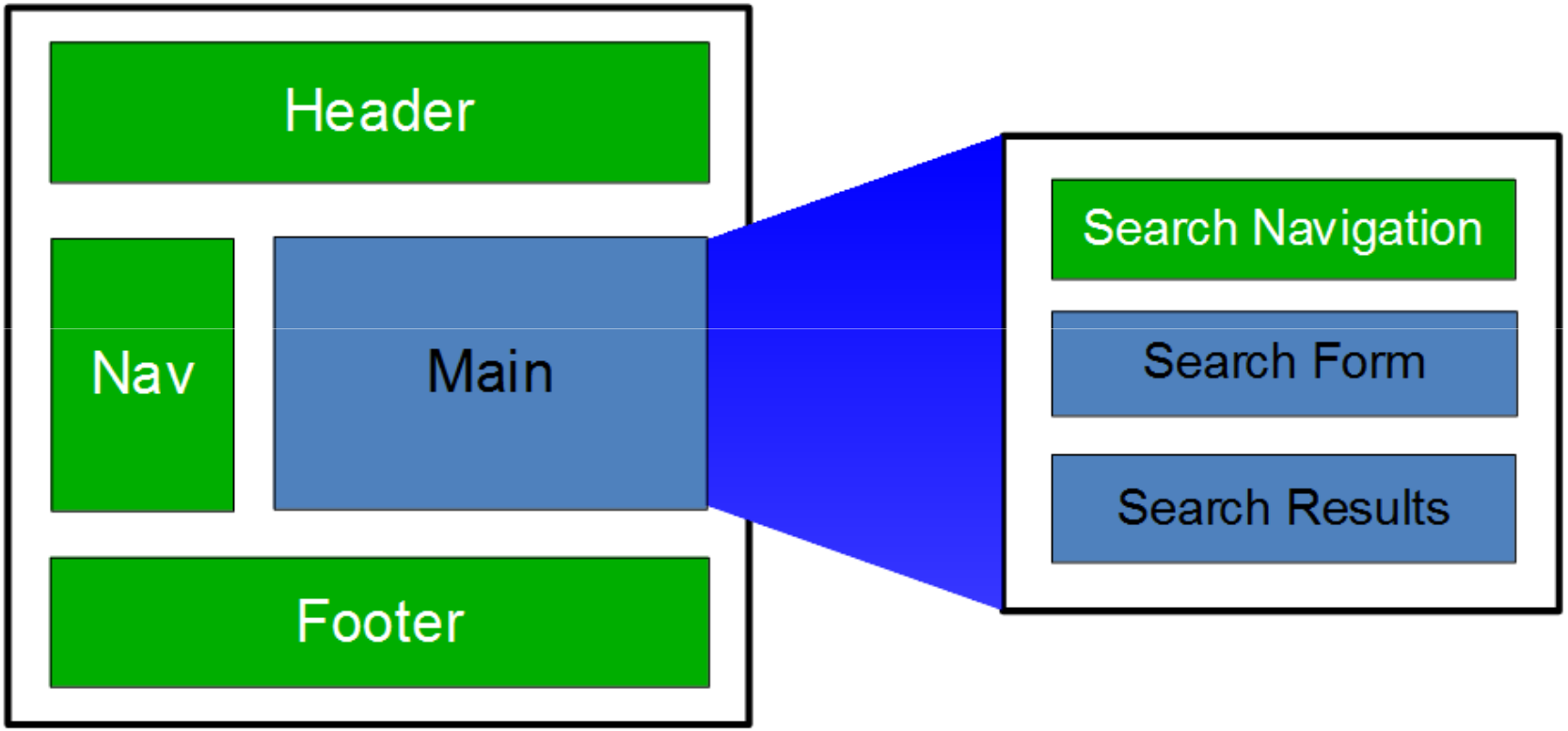
- **Create a template that holds the common elements of a page**
- **Leave placeholders where dynamic content is needed**
- **Re-use the template and supply what it requires**
  - ✓ sub-views
  - ✓ simple string values

Sivaprasad.valluru@gmail.com

Composite View Example



Nested View Composition



A Tiles Template Example

```
<%@ taglib prefix="tiles"
            uri="http://tiles.apache.org/tags-tiles" %>
```

```
<head>
    <title>
        <tiles:insertAttribute name="title" />
    </title>
</head>
<body>
    <div id="header"> ... </div>
    <div id="body">
        <tiles:insertAttribute name="main" />
    </div>
    <div id="footer"> ... </div>
</body>
```

Placeholder for dynamic content

/WEB-INF/layouts/standard.jsp

Accessing Tiles Template From JSP Page

```
<%@ taglib prefix="tiles"
                uri="http://tiles.apache.org/tags-tiles" %>
```

```
<tiles:insertTemplate template
                        ="/WEB-INF/layouts/standard.jsp" >
    <tiles:putAttribute
          name="title" value="Welcome To My
Application" />
    <tiles:putAttribute name="main">
        <h1>
            Welcome to My Application
        </h1>
    </tiles:putAttribute>
</tiles:insertTemplate>
```

Specify the dynamic content via attributes

Note: Although this works, it is *not* the usual way to use tiles. Tiles definition files are more typical – next section.

Creating Tiles Definitions

- **Tiles allows creating external XML configuration to define the structure of a page**

- **A Tiles configuration file contains one or more Tiles definitions**

- **Tiles definitions are reusable fragments consisting of a template and attributes**

Sivaprasad.valluru@gmail.com

Tile Definition Example

```
<tiles-definitions>

    <definition name="baseLayout" template="/WEB-INF/jsp/commons/baseLayout.jsp">
        <put-attribute name="title" value="Way2Learn for Free" />
        <put-attribute name="offerBanner" value="/WEB-INF/jsp/commons/offerBanner.jsp" />
        <put-attribute name="topMenu" value="/WEB-INF/jsp/commons/topMenu.jsp" />
        <put-attribute name="body" value="homePageBody" />
        <put-attribute name="footer" value="/WEB-INF/jsp/commons/footer.jsp" />
    </definition>

    <definition name="homePageBody" template="/WEB-INF/jsp/commons/body.jsp">
        <put-attribute name="bannerBox" value="/WEB-INF/jsp/commons/bannerBox.jsp" />
        <put-attribute name="homePageMainBody" value="homePageMainBody"/>
        <!-- <put-attribute name="clients" value="/WEB-INF/jsp/commons/clients.jsp" /> -->
        <put-attribute name="aboveFooter" value="/WEB-INF/jsp/commons/aboveFooter.jsp" />
    </definition>

    <definition name="homePageMainBody" template="/WEB-INF/jsp/commons/homePageMainBody.jsp">
        <put-attribute name="leftMenu" value="/WEB-INF/jsp/commons/leftMenu.jsp" />
        <!-- <put-attribute name="body" value="Just body here" /> -->
         <put-attribute name="body" value="/WEB-INF/jsp/commons/youTubeVideoPage.jsp" />
        <put-attribute name="rightMenu" value="/WEB-INF/jsp/commons/news.jsp" />
    </definition>
```

Using importAttribute in JSP

```
<tiles:importAttribute name="navigationTab" />
<c:if test="${navigationTab eq 'home'}">
    …
</c:if>
```

```
<definition name="accounts/list" extends="standardLayout">
  <put-attribute name="title" value="accounts.list.title" />
  <put-attribute name="main" value="/WEB-INF/accounts/list.jsp" />
  <put-attribute name="navigationTab" value="accounts" />
<definition>
```

/WEB-INF/tiles.xml

Integrating Tiles in Spring MVC

- **Spring MVC provides support for configuring and using Tiles**
  - ✓ **TilesConfigurer** helps to bootstrap Tiles with a set of Tiles configuration files
  - ✓ **TilesView** interprets logical view names as Tiles definition names

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.tiles3.TilesViewResolver"/>

<bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles3.TilesConfigurer" autowire="default">
    <property name="definitions">
        <list>
            <value>/WEB-INF/tiles.xml</value>
        </list>
    </property>
</bean>
```

Tiles Configurer using MVC Namespace

```
<mvc:tiles-configurer  validate-definitions="true">
    <mvc:definitions location="/WEB-INF/tiles.xml"/>
</mvc:tiles-configurer>
```


Sivaprasad.valluru@gmail.com

LAB- WORKING WITH TILES

Sivaprasad.valluru@gmail.com

LAB : TILES WITH SPRING BOOT

Sivaprasad.valluru@gmail.com

HANDLING EXCEPTIONS

Sivaprasad.valluru@gmail.com

- Spring provides HandlerExceptionResolver for resolving Exceptions to either error views ,status codes,etc

```
public interface HandlerExceptionResolver {

        ModelAndView resolveException(HttpServletRequest request,
                HttpServletResponse response,
                Object handler, Exception ex);
}
```


Sivaprasad.valluru@gmail.com

Exception Resolvers

- **SimpleMappingExceptionResolver**
  - ✓ Exception resolver implementation that maps exception class names to view names
- **DefaultHandlerExceptionResolver**
  - ✓ Exception resolver implementation that translates standard Spring exceptions to HTTP status codes
- **ResponseStatusExceptionResolver**
  - ✓ Custom exceptions in Spring applications can be annotated with @ResponseStatus, which takes a HTTP status code as its value. This exception resolver translates the exceptions to its mapped HTTP status codes
- **ExceptionHandlerExceptionResolver**
  - ✓ Exception resolver implementation that resolves exceptions using annotated @ExceptionHandler methods.

SimpleMappingExceptionResolver

```
<bean id="simpleMappingExceptionResolver"
    class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <map>
            <entry key="com.way2learnonline.exceptions.CourseNotFoundException" value="coursenotfound" />
            <entry key="com.way2learnonline.exceptions.InvalidRequestException" value="invalidrequest" />
        </map>
    </property>
    <property name="exceptionAttribute" value="ex" />
    <property name="defaultErrorView" value="defaultErrorPage" />
</bean>
```

Adding ExceptionResolver in JavaConfiguration

```
@ComponentScan(basePackages={"com.way2learnonline.controllers"})
@Configuration
@EnableWebMvc
public class MvcConfig extends WebMvcConfigurerAdapter {


    @Override
    public void configureHandlerExceptionResolvers(List<HandlerExceptionResolver> exceptionResolvers) {
        SimpleMappingExceptionResolver simpleMappingExceptionResolver= new SimpleMappingExceptionResolver();

        Properties exceptionMappings= new Properties();
        exceptionMappings.put("com.way2learnonline.exceptions.CourseNotFoundException", "coursenotfound");
        exceptionMappings.put("com.way2learnonline.exceptions.InvalidRequestException", "invalidrequest");
        simpleMappingExceptionResolver.setExceptionMappings(exceptionMappings);
        simpleMappingExceptionResolver.setExceptionAttribute("ex");
        simpleMappingExceptionResolver.setDefaultErrorView("defaultErrorPage");

        exceptionResolvers.add(simpleMappingExceptionResolver);
        super.configureHandlerExceptionResolvers(exceptionResolvers);
    }

}
```

ResponseStatusExceptionResolver

```
@ResponseStatus(value=HttpStatus.NOT_FOUND,reason="Course Not Found!!!!")
public class CourseNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 5898035931475595958L;

    public CourseNotFoundException() {}

    public CourseNotFoundException(String message) {
        super(message);
    }
}
```


Sivaprasad.valluru@gmail.com

ExceptionHandlerExceptionResolver

```
@ExceptionHandler(value={CourseNotFoundException.class})
public ModelAndView handleCourseNotFound(){

    ModelAndView mav= new ModelAndView("coursenotfound");
    mav.addObject("someData", "If you are seeing this, then this "
            + "exception is mapped by @ExceptionHandler(value={CourseNotFoundException.class})");

    return mav;
}


@ExceptionHandler(value={InvalidRequestException.class})
public ModelAndView handleInvalidRequest(){

    ModelAndView mav= new ModelAndView("invalidrequest");
    mav.addObject("someData", "If you are seeing this, then this "
            + "exception is mapped by @ExceptionHandler(value={InvalidRequestException.class})");

    return mav;
}
```

Using @ControllerAdvice

```
@ControllerAdvice
public class MyControllerAdvice {

    @ExceptionHandler(value={CourseNotFoundException.class})
    public ModelAndView handleCourseNotFound(){

        ModelAndView mav= new ModelAndView("coursenotfound");
        mav.addObject("someData", "If you are seeing this, then this exception "
                + "is mapped by @ExceptionHandler(value={CourseNotFoundException.class})");

        return mav;
    }

    @ExceptionHandler(value={InvalidRequestException.class})
    public ModelAndView handleInvalidRequest(){

        ModelAndView mav= new ModelAndView("invalidrequest");
        mav.addObject("someData", "If you are seeing this, then this exception "
                + "is mapped by @ExceptionHandler(value={InvalidRequestException.class})");

        return mav;
    }
}
```

LAB – EXCEPTION HANDLING IN SPRING MVC

Sivaprasad.valluru@gmail.com

INTERNATIONALIZATION

Sivaprasad.valluru@gmail.com

- **Configure a MessageSource as below**

```
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages" />
</bean>
```

- **Then write messages.properties and messages_es.properties with key value pairs**
- **In JSP, use tags as below**

```
<spring:message code="browsecourses"/>
```

Reloadable Message Source

- **Will reload after cache-seconds**
  - ✓ Never reload = -1 (default)
  - ✓ Always reload = 0 (never in production!)

```xml
<bean id="messageSource"
    class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basename" value="messages" />
    <property name="cacheSeconds" value="60" />
</bean>
```

LocaleResolver

```
<mvc:interceptors>
    <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" />
</mvc:interceptors>

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
    <property name="cookieName" value="localecookie" />
    <property name="defaultLocale" value="en" />
</bean>
```

LAB- INTERNATIONALIZATION

Sivaprasad.valluru@gmail.com

Configuring Using JavaConfiguration

```
@Bean(name="localeResolver")
public LocaleResolver LocaleResolver(){
    CookieLocaleResolver cookieLocaleResolver= new CookieLocaleResolver();
    cookieLocaleResolver.setCookieName("mylocalecookie");
    cookieLocaleResolver.setDefaultLocale(new Locale("en"));
    cookieLocaleResolver.setCookieMaxAge(3600);
    return cookieLocaleResolver;
}

@Bean
public LocaleChangeInterceptor localeChangeInterceptor(){
    return new LocaleChangeInterceptor();
}



@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(localeChangeInterceptor());
}
```

Configuring MessageSource Using Boot

- **See** MessageSourceAutoConfiguration
- **The default basename is configured as** message
- **So, if messages.properties is present in classpath, a messageSource will be automatically configured by spring boot.**
- If you want to change the base name, configure following in application.properties
  - ✓ Spring.message.basename=mybasename

Sivaprasad.valluru@gmail.com

LAB- INTERNATIONALIZATION USING BOOT

Sivaprasad.valluru@gmail.com

PERSONALIZATION USING THEMES

Sivaprasad.valluru@gmail.com

What Is A Theme?

- **Themes allow dynamic determination of look-and-feel related values**
  - ✓ Paths to CSS files, images, etc.
- **Each theme has a name and is backed by a dedicated MessageSource instance**
- **A user can change an application's look-and-feel at runtime by switching the theme**

The ThemeResolver

- **An abstraction for determining the current theme name**
  - ✓ The DispatcherServlet looks for a ThemeResolver
    - ✓ The bean id is expected to be "themeResolver”
- **FixedThemeResolver is configured by default**
  - ✓ Default theme name of "theme”
  - ✓ Expects a theme.properties file

Sivaprasad.valluru@gmail.com

ThemeResolver Types

FixedThemeNameResolver (enabled by default)	• Uses a single theme
CookieThemeResolver	• Reads/writes the theme name to a cookie
SessionThemeResolver	• Reads/writes the theme name to the HTTP Session

Sivaprasad.valluru@gmail.com

Cookie Theme Resolver

```
<mvc:interceptors>
    <bean class="org.springframework.web.servlet.theme.ThemeChangeInterceptor"/>
</mvc:interceptors>

<bean id="themeResolver" class="org.springframework.web.servlet.theme.CookieThemeResolver">
    <property name="defaultThemeName" value="blue"/>
    <property name="cookieName" value="clientTheme" />
    <property name="cookieMaxAge" value="3600" />
</bean>
```


Sivaprasad.valluru@gmail.com

blue.properties

```
layout.css=/resources/css/layout.css
```

red.properties

```
layout.css=/resources/css/layout-red.css
```

In JSP

```
<spring:theme var="layoutCSS" code="layout.css"/>
<c:url var="layoutCssURL"  value="${layoutCSS}" />

<link rel="stylesheet" type="text/css" href="${layoutCssURL}">
```

LAB : MVC-THEMES

Sivaprasad.valluru@gmail.com

Request Context

- **Context holder for request-specific state**
  - ✓ current web application context
  - ✓ current locale (and timezone from Spring 4.0)
  - ✓ current theme
  - ✓ binding errors
- **Provides easy access to localized messages and Errors instances.**

Sivaprasad.valluru@gmail.com

Define A RequestContext

- **Not available by default**
  - ✓ Need to expose a **requestContext** attribute
  - ✓ *Option 1: Directly on View class*
    - ✓ Almost all views extend AbstractView which provides setRequestContextAttribute(String name)
  - ✓ *Option 2: Via any UrlBasedViewResolver subclass such as ...*

```
<bean class="org.springframework.web.
                servlet.view.InternalResourceViewResolver">
   <property name="requestContextAttribute"
             value="requestContext"/>
</bean>
```


Sivaprasad.valluru@gmail.com

Usage: In JSP

- **Access request data in a JSP page**
  - ✓ Accessible in other views also

```
<c:out value="${requestContext.locale.language}" />

<c:out value="${requestContext.theme.name}" />
```

Usage: In code

- **Locale can be injected directly**
  - ✓ Can only get the theme from the RequestContext

```
@Controller
public class MusicController {

  @RequestMapping(value="/fetchForCurrentGenre")
  public String suggest(Model model, Locale locale,
                                HttpServletRequest req) {
    RequestContext context = new RequestContext(req);
    String themeName = context.getTheme().getName();

    // Suggest some music for current type of user
    model.addAttribute("genre",
            musicManager.lookupFor(themeName, locale));

    return "songs/list";  // Suggest for current genre
  }
```

VIEW RESOLVERS IN DETAILS

Sivaprasad.valluru@gmail.com

View Resolvers

- **Separation of concerns between controller and View**
  - ✓ Controller does not need to know view implementation
  - ✓ Just deals in *logical view names*
- **Encapsulate View selection strategy**
  - ✓ Does *not render output*
  - ✓ Several built-in implementations supplied

```
public interface ViewResolver {
    View resolveViewName(String viewName, Locale locale)
        throws Exception;
}
```


Sivaprasad.valluru@gmail.com

What is a View?

- **An implementation of the View interface**
  - ✓ Implementations use a variety of rendering techniques
    - ✓ View *'knows' the type of output it can produce*
    - ✓ The render() method generates output
  - ✓ Default is JstlView for rendering JSP
    - ✓ Generates text/html

```
public interface View {

    String getContentType();    // Mime type created by render()

    void render(Map<String, ?> model, HttpServletRequest request,
                HttpServletResponse response) throws Exception;
}
```

Resource (URL) Based Views

- ***A logical view name typically corresponds to a file***
  - ✓ JSP
  - ✓ Velocity or FreeMarker template
  - ✓ XSLT stylesheet
  - ✓ …
- **These files are typically stored under /WEB-INF**
  - ✓ Hidden from direct browser access
- **Rendering requires a model**
  - ✓ The view name is not *always an actual file*
    - ✓ For example: a Tiles definition name

URL-Based View Example

- **JstlView**
  - ✓ exposes the model attributes as request attributes
  - ✓ adds attributes for JSTL format/message tags
  - ✓ forwards to JSP page
  - ✓ default View implementation

```
JstlView view = new JstlView
                        ("/WEB-INF/rewards/show.jsp");
view.render(model, request, response);
```


Sivaprasad.valluru@gmail.com

UrlBasedViewResolver

- **Typically interprets a view name as a resource location**
- **Many subclasses**
  - ✓ InternalResourceViewResolver (JSP/Servlets)
  - ✓ VelocityViewResolver
  - ✓ FreeMarkerViewResolver
  - ✓ ThymeleafViewResolver
  - ✓ XsltViewResolver
- **Supports “redirect:” prefix**

Content-Generating Views

- **Extends from a base class**
  - ✓ AbstractExcelView, AstractPdfView, etc
- **Creates view content using an API**
  - ✓ POI, iText
- **The base class writes generated content to the response stream**

Excel View Example

```
public class CoursesExcelView extends AbstractExcelView {

    @Override
    @SuppressWarnings("unchecked")
    protected void buildExcelDocument(Map<String, Object> model,
            HSSFWorkbook workbook, HttpServletRequest request,
            HttpServletResponse response) throws Exception {

        List<Course> courses = (List<Course>)
                request.getSession().getServletContext().getAttribute("courses");


        HSSFSheet sheet = workbook.createSheet();


        for (short i = 0; i < courses.size(); i++) {
            Course course = courses.get(i);
            HSSFRow row = sheet.createRow(i);
            addStringCell(row, 0, course.getCourseId());
            addStringCell(row, 1, course.getName());
            addStringCell(row, 2, course.getPrice()+"");
        }
    }
```

BeanNameViewResolver

- **Interprets a view name as a bean name**
  - ✓ Bean *must be a View instance*

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

<bean id="browseCourses.xls" class="in.co.way2learn.controllers.CoursesExcelView"/>
```

```
@RequestMapping(value = { "/browseCourses"}, method = RequestMethod.GET)
public String browseCourses(Map<String, Object> modelMap) {

    modelMap.put("courses",courseService.getAll(false));
    return "browseCourses.xls";
}
```

XmlViewResolver

- **Picks up View beans from a given location file**
  - ✓ Works just like BeanNameViewResolver
    - ✓ But matches only Views defined in its bean file
- **Reduces bean-file clutter**
- **Keeps View beans separate**

```
<bean class="...web.servlet.view.XmlViewResolver">
    <property name="location">
        <value>/WEB-INF/spring/views.xml</value>
    </property>
</bean>
```

Standard Spring Bean file containing *only* View beans

Sivaprasad.valluru@gmail.com

ViewResolver Chain

- **The DispatcherServlet discovers ViewResolver beans by *type***
  - ✓ Multiple ViewResolver beans are possible
    - ✓ Each is given a “chance” to match
    - ✓ You can specify the order (via a property)
  - ✓ The first resolver to return a View “wins”

ViewResolver Chain Example

```
<bean class="...web.servlet.view.BeanNameViewResolver">
    <property name="order" value="1">
</bean>

<bean class="...web.servlet.view.InternalResourceViewResolver">
    <property name="order" value="2">
    <property name="prefix" value="/WEB-INF/”>
    <property name="suffix" value=".jsp">
</bean>
```

Simplified Since Spring 4.1

```
<!-- Beans chained in order listed -->
<mvc:view-resolvers>
  <bean-name/>
  <mvc:jsp prefix="/WEB-INF/" suffix=".jsp"/>
</mvc:view-resolvers>
```


Sivaprasad.valluru@gmail.com

ViewResolver Chain Order

- **All ViewResolver beans implement Ordered**
- **Some UrlBasedViewResolvers can be anywhere in the chain**
  - ✓ Depends on the View type served
  - ✓ Tiles, Velocity, Freemarker view resolvers check their resources really exist
    - ✓ return null if they don't
- **Others must be last**
  - ✓ JSTL/JSP, XSLT, JSON *always forward, never return null*
  - ✓ These resolvers *must be last*
    - ✓ Or sub-class to implement a resource check and return null

CONTENT NEGOTIATION

Sivaprasad.valluru@gmail.com

Content Type Negotiation

- **Clients may request different content types for the *same resource***
  - ✓ via file extension
  - ✓ and/or request parameter
  - ✓ and/or request header
- **The process of determining which type to render is known as *Content Type Negotiation***

Sivaprasad.valluru@gmail.com

Content Type Negotiation Example

- **Example requirement**
  - ✓ Provide Excel link on Account Search Results page
- **Sample requests (if using extensions)**
  - ✓ HTML:
    - ✓ GET /accounts/list.html
  - ✓ Excel:
    - ✓ GET /accounts/list.xls
- **If using the format parameter**
  - ✓ GET /accounts/list?format=html
  - ✓ GET /accounts/list?format=xls

Option #1: Separate Methods

- **Use a different controller method for each view type**

```
@RequestMapping("/accounts.htm")
public String listHtml(HttpServletRequest rq, Model model) {
    model.addAttribute(accountManager.findAllAccounts());
    return "accounts/list";
}

@RequestMapping("/accounts.xls")
public String listXcel(HttpServletRequest rq, Model model) {
    model.addAttribute(accountManager.findAllAccounts());
    return "accounts/list.xls";
}
```

Violates DRY principle

Option #2: Controller Logic

- **Controller detects URL extension (for example) and selects view name**

```
@RequestMapping("/accounts")
public String list(HttpServletRequest req, Model model) {
   model.addAttribute(accountManager.findAllAccounts());

   if (req.getRequestURI().endsWith(".xls"))
      return "accounts/list.xls";
   else
      return "accounts/list";
}
```

Extension ignored.
Equivalent to: `/accounts.*`

Works, but if-else logic will likely be replicated to other controllers!

Sivaprasad.valluru@gmail.com

Option #3: Special View Resolver

- ***ContentNegotiatingViewResolver (CNVR)***
  - ✓ Introduced in Spring 3.0
  - ✓ *Configuration changed significantly in 3.2*
- **Does not do view resolution itself**
  - ✓ Delegates to other view resolvers
  - ✓ Finds View generating requested content type
- **Each View class has an associated content type**
  - ✓ Existing views already define their content types

`AbstractPdfView` → application/pdf
`AbstractExcelView` → application/vnd.ms-excel
`JstlView` → text/html

Sivaprasad.valluru@gmail.com

ContentNegotiatingViewResolver

```
<bean class="org.springframework.web.servlet.
                        view.ContentNegotiatingViewResolver">
  <!-- Configuration delegated to manager since Spring 3.2 -->
  <property name="contentNegotiationManager" ref="cnManager" />

  <property name="viewResolvers">
    <list>
      <bean class="org.springframework.web.servlet.
                        view.BeanNameViewResolver"/>

      <bean class="org.springframework.web.servlet.
                        view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/"/>
        <property name="suffix" value=".jsp"/>
      </bean>
     </list>
   </property>
</bean>
```

The view resolvers to negotiate between

Simplifying CNVR Configuration

- **CNVR configuration can be *much simpler***
  - ✓ Configure view resolvers separately from the CNVR
    - ✓ No need for viewResolvers property
- **Extensions can be mapped to MIME types using the *Java Activation Framework***
  - ✓ No need for mediaTypes property
  - ✓ Presumes standard extensions (.pdf, .xls, ...)
  - ✓ *activation.jar must be on the classpath*
  - ✓ Specify useJaf property to disable (on by default)
- **Namespace support since Spring 4.1**

Simplified Configuration – I

```
<bean class="...ContentNegotiatingViewResolver"
      p:content-negotiation-manager-ref="cnManager" />

<bean id="cnManager" p:defaultContentType="text/html"
      class="...ContentNegotiationManagerFactoryBean" />


<bean class="...XmlViewResolver"
      p:location="/WEB-INF/spring/pdf-views.xml" />

<bean class="...XmlViewResolver"
      p:location="/WEB-INF/spring/excel-views.xml" />

<bean class="...InternalResourceViewResolver"
      p:prefix="/WEB-INF/" p:suffix=".jsp" />
```

Types & resolvers are determined automatically

Simplified Configuration – II

```
<!-- As previous slide, but also defines a default view (using JSP).
     Automatically uses default ContentNegotiationManager -->
<mvc:view-resolvers>
  <mvc:content-negotiation use-not-acceptable="true">
    <mvc:default-views>
      <bean class="org.springframework.web.servlet.view.JstlView" />
    </mvc:default-views>
  </mvc:content-negotiation>

  <bean class="...XmlViewResolver"
        p:location="/WEB-INF/spring/pdf-views.xml" />
  <bean class="...XmlViewResolver"
        p:location="/WEB-INF/spring/excel-views.xml" />
  <mvc:jsp prefix="/WEB-INF/" suffix=".jsp"/>
</mvc:view-resolvers>
```

Configuring the CNVR

- **Defines view resolvers to negotiate between**
  - ✓ Must be first view resolver used
  - ✓ All properties (except the three below) now delegated to a manager, and deprecated since 3.2

Property	Details
`order`	• CNVR should be first in any resolver chain • Default is Ordered.HIGHEST_PRECEDENCE
`viewResolvers`	• The view resolvers to delegate to. • If undefined, *all* view resolvers are detected
`useNotAcceptableStatusCode`	• If no suitable view is found, return HTTP Status 406 • Default false – CNVR returns null and view-resolver chaining can be used

Sivaprasad.valluru@gmail.com

ContentNegotiationManager

- **Negotiation functionality moved into new class**
  - ✓ ContentNegotiationManager
    - ✓ Select which formats supported
    - ✓ Configure how format is specified in request
    - ✓ Specify default (fall-back) format

- **A default ContentNegotiationManager is created**
  - ✓ if <mvc:annotation-driven> or *@EnableWebMvc specified*
  - ✓ Or define your own (next slide)

ContentNegotiationManagerFactoryBean

```
<!-- Configuration delegated to manager since Spring 3.2 -->
<bean id="cnManager" class="org.springframework.web
                .accept.ContentNegotiationManagerFactoryBean">

  <property name="mediaTypes">
    <map>
      <entry key="html" value="text/html" />
      <entry key="json" value="application/json" />
      <entry key="xls" value="application/vnd.ms-excel" />
    </map>
  </property>

  <property name="favorParameter" value="true" />
  <property name="ignoreAcceptHeader" value="true" />
  <property name="defaultContentType" value="text/html" />
</bean>

<mvc:annotation-driven content-negotiation-manager="cnManager" />
```

Map extensions to content type

Favor means *enable*

Configuring the Content Negotiation Manager

- **Manager configuration properties**

Property	Details
`defaultContentType`	• The type to render if a match is not found • No default
`favorParameter`	• Use URL parameter: http://..?**format=pdf** • Default is false
`favorPathExtension`	• Determine content type from file-type in URL • Example: http://…somepage**.pdf** • Default is true
`parameterName`	• Name of URL parameter • Default is "*format*"
`ignoreAcceptHeader`	• Use the accept header? • Default is false

Sivaprasad.valluru@gmail.com

What Format?

- ***Option 1: URL Suffix***
  - ✓ URL ends in a suffix
  - ✓ Example: http://.../accounts.pdf
  - ✓ Use favorPathExtension=true
- ***Option 2: Format parameter***
  - ✓ URL contains *format=xyz*
  - ✓ Example: http://.../accounts?format=pdf
  - ✓ Use favorParameter=true
- ***Option 3: Accepts header***
  - ✓ An HTTP Request header property
  - ✓ Use ignoreAcceptHeader=false

LAB-CONTENT NEGOTIATION

Sivaprasad.valluru@gmail.com

FORM-HANDLING

Sivaprasad.valluru@gmail.com

Using Spring Form Tags

- **Add the taglib directive**

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
```

- **Use the Spring <form> tag**

```
<form method="post" action="...">
```

```
<form:form method="post" action=""
                         modelAttribute="requestInfo">
```

Form object can be *any* attribute from the model

Jsp Form

```
<head>
<title>Add Employee Form</title>
<style>
.error {
color: #ff0000;
font-weight: bold;
}
</style>
</head>
</head>

<body>
    <form:form method="post" modelAttribute="employee">
        <form:errors path="*"  cssClass="error"/>
        <table>
          <tr>
            <td>EMPLOYEE ID</td>
            <td><form:input path="id" /></td>
            <td><form:errors path="id"  cssClass="error"/></td>
          </tr>
          <tr>
            <td>EMPLOYEE NAME</td>
            <td><form:input path="name" /></td>
              <td><form:errors path="name"  cssClass="error"/></td>
          </tr>
```

```
<tr>
  <td>DATE OF BIRTH</td>
  <td> <form:input path="dateOfBirth" /></td>
  <td><form:errors path="dateOfBirth"  cssClass="error"/></td>
</tr>
 <tr>
  <td>CITY</td>
  <td> <form:input path="address.city" /></td>
    <td><form:errors path="address.city"  cssClass="error"/></td>
</tr>
 <tr>
  <td>COUNTRY</td>
  <td> <form:input path="address.country" /></td>
    <td><form:errors path="address.country"  cssClass="error"/></td>
</tr>
<tr>
  <td colspan="3">
      <input type="submit" value="Save" />
  </td>
</tr>
</table>
</form:form>
```

```java
@Controller
@SessionAttributes({"requestInfo","cities"})
public class RequestInfoController {

    @RequestMapping(value={"/requestInfo.htm","/s/requestInfo.htm"},method=RequestMethod.GET)
    public String requestInfo(Map<String, Object>  modelMap){
        modelMap.put("requestInfo", new RequestInfoData());

        return "requestInfo";
    }

    @RequestMapping(value={"/requestInfo.htm","/s/requestInfo.htm"},method=RequestMethod.POST)
    public String requestInfoPost(@ModelAttribute("requestInfo") @Valid RequestInfoData requestInfoData,
                        BindingResult result,SessionStatus  sessionStatus){

        if(result.hasErrors()){
            return "requestInfo";
        }

        // Invoke Business Logic
        sessionStatus.setComplete();

        return "requestInfoSuccess";
    }
```

Control Data Binding Fields

- **Allowed fields (white list)**

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.setAllowedFields("date", "amount");
}
```

Strongly recommended!

- **Disallowed fields (black list)**

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.setDisallowedFields("id", "*Id");
}
```

Wild cards may be used

Customize "Type Mismatch" Error Messages

- **Customize "Type Mismatch" Data Binding Error Messages**

```
# Any "type mismatch" error
typeMismatch=Incorrect value

# Type mismatch errors for a named field
typeMismatch.amount=Incorrect amount

# Type mismatch errors for a named field on a named model
attribute
typeMismatch.diningForm.amount=Incorrect dining amount

# Type mismatch errors for all fields of a specific type
typeMismatch.common.money.MonetaryAmount=Incorrect \
                                          monetary amount
```

Providing form Reference Data

```
@ModelAttribute("cities")
public List<City> getCties(){
    List<City> cities=Arrays.asList(new City("Hyderabad", "hyd"),
            new City("Bangalore", "blr"),
            new City("Delhi", "del"));
    return cities;
}
```

```
<form:select path="city" items="${cities}"
    itemLabel="cityName"  itemValue="cityCode"/>
```

REDIRECTION AND FLASH SCOPE

Sivaprasad.valluru@gmail.com

POST-Redirect-GET

- **User should never see a page resulting from an HTTP POST**
  - ✓ If the page is refreshed the POST is resubmitted
  - ✓ The effect of the original POST is repeated
    - ✓ Suppose you were buying something online … you just bought *two of them!*
- **Useful Pattern: POST-Redirect-GET**
  - ✓ After a successful POST the user is redirected to the next page via a GET request
  - ✓ A refresh redisplays the success page
    - ✓ No repeat of the POST

Redirect

- **Redirect prefix causes a browser redirect**
  - ✓ Returns an HTTP 302 Redirect response
  - ✓ Browser then loads new URL
  - ✓ This is a *new request – original request data lost*

- **Can be absolute or servlet-context relative**
  - ✓ redirect:http://www.spring.io
  - ✓ redirect:accounts

Redirect Attributes

- **Allow control of data passed on redirect**
  - ✓ *RedirectAttributes extends Model*
    - ✓ Attributes passed via query-string on redirect

```
@RequestMapping(value={"/requestInfo.htm","/s/requestInfo.htm"},method=RequestMethod.POST)
public String requestInfoPost(@ModelAttribute("requestInfo") RequestInfoData requestInfoData,
                    BindingResult result,
                    SessionStatus  sessionStatus,RedirectAttributes redirectAttributes){

    if(result.hasErrors()){
        return "requestInfo";
    }

    redirectAttributes.addAttribute("name",requestInfoData.getName());
    redirectAttributes.addAttribute("email",requestInfoData.getEmail());
    sessionStatus.setComplete();

    return "redirect:requestInfoSuccessRedirect";
}

@RequestMapping("requestInfoSuccessRedirect")
public String requestInfoSuccessRedirect(){

    return  "requestInfoSuccess";
}
```


Sivaprasad.valluru@gmail.com

Flash Scope

- **Flash: Allow for data that survives a redirect**
  - ✓ Use *addFlashAttribute()*
    - ✓ Will appear in Model of *next request for this use*

```
redirectAttributes.addFlashAttribute("requestInfoData", requestInfoData);
```


Sivaprasad.valluru@gmail.com

LAB-MVC FORM HANDLING

Sivaprasad.valluru@gmail.com

FORMATTERS

Sivaprasad.valluru@gmail.com

Using Formatters

- **Annotations: inside the bean class**

```
public class Account {
  @DateTimeFormat(iso=ISO.DATE)
  private java.util.Date endDate;
}
```

Used by *all* JSPs displaying info for this bean

- **JSP tags (using the <fmt> tag library)**

```
<fmt:formatDate value="${account.endDate}"
                pattern="MM/dd/yyyy" />
```

Each JSP can have its *own* formatting pattern

- **Register custom Formatters**
- **Usually used for custom business beans (such as SocialSecurityNumber, BankAccount …)**

Formatting Annotations Example

```
public class Account {

  private final String name;
  private final String number;

  @NumberFormat(style=Style.CURRENCY)
  private final BigDecimal amount;

  @NumberFormat(style=Style.NUMBER, pattern="#,###.###")
  private final BigDecimal interestAmount;

  @DateTimeFormat(iso=ISO.DATE)
  private final Date endDate;
}
```


Sivaprasad.valluru@gmail.com

Default Configuration

- **The mvc namespace registers a set of formatters by default**

```
<mvc:annotation-driven />
```

Registers formatters for numbers and Dates

- **Register a conversion service to add your own**

```
<mvc:annotation-driven
     conversion-service="conversionService" />
```

Creating a Custom Formatter

```
public class SSNFormatter implements Formatter<SocialSecurityNumber> {

    @Override
    public String print(SocialSecurityNumber ssn, Locale locale) {

        return ssn.getArea()+"-"+ssn.getGroup()+"-"+ssn.getSerial();
    }

    @Override
    public SocialSecurityNumber parse(String text, Locale locale)
            throws ParseException {
        SocialSecurityNumber ssn= new SocialSecurityNumber();
        ssn.setArea(text.substring(0,3));
        ssn.setGroup(text.substring(4,6));
        ssn.setSerial(text.substring(7,11));
        return ssn;
    }

}
```

Registering a Custom Formatter

```
<mvc:annotation-driven  conversion-service="conversionService"/>

<bean id="conversionService"
    class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
    <property name="formatters">
        <list>
            <bean class="in.co.way2learn.controllers.SSNFormatter" />
        </list>

    </property>
</bean>
```

LAB- WRITING CUSTOM FORMATTER

Sivaprasad.valluru@gmail.com

VALIDATIONS

Sivaprasad.valluru@gmail.com

Validating Form Objects

- **Spring 3.0 supports JSR 303 (Bean Validation) for validating form objects**
  - ✓ Hibernate Validator is the reference implementation
  - ✓ Annotation-driven
  - ✓ Both API and implementation must be on the classpath
- **Custom validation**
  - ✓ Write own validator
  - ✓ Implement org.springframework.validation.Validator

JSR 303 Validation Annotations

Annotation	Purpose
`@NotNull`	The field cannot be null
`@NotEmpty`	Field is not null and not empty – valid for String and Collection type *(not standard - see note)*
`@Size(min="", max="")`	Field must have a size in the range (min, max) – valid for String or Collection type
`@Pattern(regexp="")`	String field is non-null and matches the regular expression
`@Min(n)  @Max(n)`	Min / max value for a numeric field

Sivaprasad.valluru@gmail.com

Example

```
public class Account {

    @Pattern(regexp="\\d{16}")     // 16 digits
    private String creditCardNumber;

    @Size(min=10, max=10)          // Exactly 10 chars
    private String merchantNumber;

    @Min(0)                        // Must be positive
    private BigDecimal monetaryAmount;

    @NotNull                       // Must exist
    private Date date;

}
```


Sivaprasad.valluru@gmail.com

Invoking Validation

- **@Valid inside controller method**

```
@RequestMapping(value={"/requestInfo.htm","/s/requestInfo.htm"},method=RequestMethod.POST)
public String requestInfoPost(@ModelAttribute("requestInfo") @Valid RequestInfoData requestInfoData,
        BindingResult result,SessionStatus  sessionStatus){

    if(result.hasErrors()){
        return "requestInfo";
    }

    sessionStatus.setComplete();


    return "requestInfoSuccess";
}
```

- **Errors are registered in the BindingResult**
  - ✓ Combined with binding errors

Configuring Validation

- **Use <mvc:annotation-driven>**
  - ✓ Enables JSR-303 globally within Spring MVC
  - ✓ JSR-303 dependencies must be on the classpath!
- **Register a LocalValidatorFactoryBean**
  - ✓ Optional since *Spring 3.0.1*

```
<!-- Enables JSR-330 validation by default if on classpath -->
<mvc:annotation-driven validator="validator"/>

<!-- Optionally define the validator to Spring MVC - not
     necessary since Spring 3.0.1 unless you wish to customize -->
<bean id="validator" class="org.springframework.
        validation.beanvalidation.LocalValidatorFactoryBean"/>
```

Customize Validation Error Messages

- **Error messages defined in MessageSource override default text**

```
# Any "field not null" error
NotNull=Required value

# "field not null" errors for a named field
NotNull.amount=Required amount

# "field not null" errors for a named field on a named model
attribute
NotNull.diningForm.amount=Required dining amount

# "field not null" errors for all fields of a specific type
NotNull.common.money.MonetaryAmount=Required monetary amount
```

Custom Validations I

```
@Constraint(validatedBy=MySizeValidator.class)
public @interface MySize {

    String message() default " Invalid!!";
    Class<?> []groups() default {};

    int min();
    int max();
}
```


Sivaprasad.valluru@gmail.com

Custom Validator

```
public class MySizeValidator  implements ConstraintValidator<MySize, Integer>{

    int min;
    int max;

    @Override
    public void initialize(MySize mySize) {
        min=mySize.min();
        max=mySize.max();

    }

    @Override
    public boolean isValid(Integer value, ConstraintValidatorContext context) {
        System.err.println("MySizeValidator.isValid()");
        if(value<min || value>max){
            return false;
        }
        return true;
    }

}
```

Custom Validations II

```
public class DiningForm {

  public void validate(Errors errors) {
    // Custom validation checks - always check for null
    if (merchantNumber != null &&
              !merchantNumber.matches("[1-9][0-9]*"))
      errors.rejectValue("merchantNumber", "numberExpected");
  }
}
```

```
@RequestMapping(method=RequestMethod.PUT)
public String reward(DiningForm diningForm,
                                    BindingResult result) {
    diningForm.validate(result);
    // process failure or success normally ...
}
```


Sivaprasad.valluru@gmail.com

Custom Validations III-a

- **Write a Spring MVC validator**
  - ✓ *Always called – never assume any field is non-null*

```
class DiningValidator implements Validator {
  public void validate(Object target, Errors errors) {
    DiningForm form = (DiningForm)target;
    if (form.merchantNumber != null &&  // always check for null
        form.merchantNumber.matches("[1-9][0-9]*") )
      return;    // valid
    else
      errors.rejectValue("merchantNumber", "numberExpected");
  }

  // Which form types can we validate?
  public boolean supports(Class<?> clazz) {
    return clazz instanceof Dining.class;
  }
}
```

Custom Validations III-b

- **Register per controller**
  - ✓ Use @InitBinder method
- **Why Use?**
  - ✓ Form object is a domain-object
    - ✓ Don't want to add validator methods – but consider DTOs
- **Can reuse a validator**
  - ✓ Such as credit card or account number formats
  - ✓ BUT: only one validator per form supported

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.setValidator(new CreditCardNumberValidator());
}
```


Sivaprasad.valluru@gmail.com

MVC-TESTING

Sivaprasad.valluru@gmail.com

Why Spring MVC testing?

- **How can you test if**
  - ✓ The @RequestMapping will result in a correct HandlerMapping?
  - ✓ The @Valid is active? Expected validation checks happen?
  - ✓ The framework redirects where you expect?
  - ✓ The ViewResolver chain translates logical view name to the correct View?
  - ✓ Any exception results in a display of the error page?

```
@RequestMapping(value={"/requestInfo.htm","/s/requestInfo.htm"},method=RequestMethod.POST)
public String requestInfoPost(@ModelAttribute("requestInfo") @Valid RequestInfoData requestInfoData,
        BindingResult result){

    if(result.hasErrors()){
        return "requestInfo";
    }

    return "requestInfoSuccess";
}
```

Example

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes={MvcConfig.class,TestConfig.class})
@WebAppConfiguration
public class CourseControllerTests {

    private MockMvc mockMvc;

    @Autowired private ITestimonialService testimonialService;
    @Autowired private ICourseService courseService;

    @Autowired
    private WebApplicationContext webApplicationContext;

    private Course course;

    @Before
    public void setUp(){
        Mockito.reset(testimonialService,courseService);
        mockMvc=MockMvcBuilders.webAppContextSetup(webApplicationContext).build();

        Testimonial testimonial1= new Testimonial(5,"Hadoop Trainer was Excellent","Very Good Training");
        Testimonial testimonial2= new Testimonial(4,"Hadoop Trainer was good","Excellent Training");
         course= new Course("hadoop","Hadoop Course",40,6000);
        when(courseService.get("hadoop")).thenReturn(course);
        when(testimonialService.getTestimonialsByCourse("hadoop")).thenReturn(Arrays.asList(testimonial1,testimonial2));
    }
```

```
@Test
public void testViewIndividualCourseWithInvalidCourseId() throws Exception{
    MockHttpServletRequestBuilder requestBuilder=MockMvcRequestBuilders.get("/viewIndividualCourse.htm")
                                                        .param("courseId", "invalidCourseId");

    mockMvc.perform(requestBuilder)
    .andExpect(status().isOk())
    .andExpect(view().name("coursenotfound"))
    .andExpect(model().attribute("ex", hasProperty("message",
                    is("Course is not found!! Who asked you to type courseID in URL?")) ));


    verify(courseService,times(1)).get("invalidCourseId");
    verifyNoMoreInteractions(courseService);
    verifyZeroInteractions(testimonialService);
}
```

```java
@Test
public void testViewIndividualCourse() throws Exception{

    MockHttpServletRequestBuilder requestBuilder=MockMvcRequestBuilders.get("/viewIndividualCourse.htm")
            .param("courseId", "hadoop");

    mockMvc.perform(requestBuilder)
    .andExpect(status().isOk())
    .andExpect(view().name("viewIndividualCourse"))
    .andExpect(model().attribute("course", course))
    .andExpect(model().attribute("testimonials", hasSize(2)))
    .andExpect(
                model().attribute(
                        "testimonials", hasItem(
                                                allOf(
                                                        hasProperty("rating",is(5)),
                                                        hasProperty("title",is("Hadoop Trainer was Excellent")),
                                                        hasProperty("description",is("Very Good Training"))
                                                    )
                                                )
                                )
                );


    verify(courseService,times(1)).get("hadoop");
    verifyNoMoreInteractions(courseService);
}
```

How to add various attributes or params?

```
MockMvcRequestBuilders.get("/viewIndividualCourse.htm")
.param("courseId", "invalidCourseId")
.requestAttr("somereqAttr", "123")
.sessionAttr("somesessionAttr", 456)
.header("someheader", "xyz")
.locale(new Locale("es"))
.accept(MediaType.APPLICATION_JSON).
contentType(MediaType.APPLICATION_XML);
```

Checking for Validation and Binding Errors

```
MockHttpServletRequestBuilder requestBuilder=
        MockMvcRequestBuilders.post("/requestInfo.htm")
                        .contentType(MediaType.APPLICATION_FORM_URLENCODED)
                        .param("name", "siva").param("age", "111");

mockMvc.perform(requestBuilder)
        .andExpect(status().isOk())
        .andExpect(view().name("requestInfo"))
        .andExpect(model().attributeHasErrors("requestInfo"))
        .andExpect(model().attributeErrorCount("requestInfo", 1))
        .andExpect(model().attributeHasFieldErrors("requestInfo", "age"))
        .andExpect(model().attribute("requestInfo",
                                hasProperty("name",is("siva"))));
```