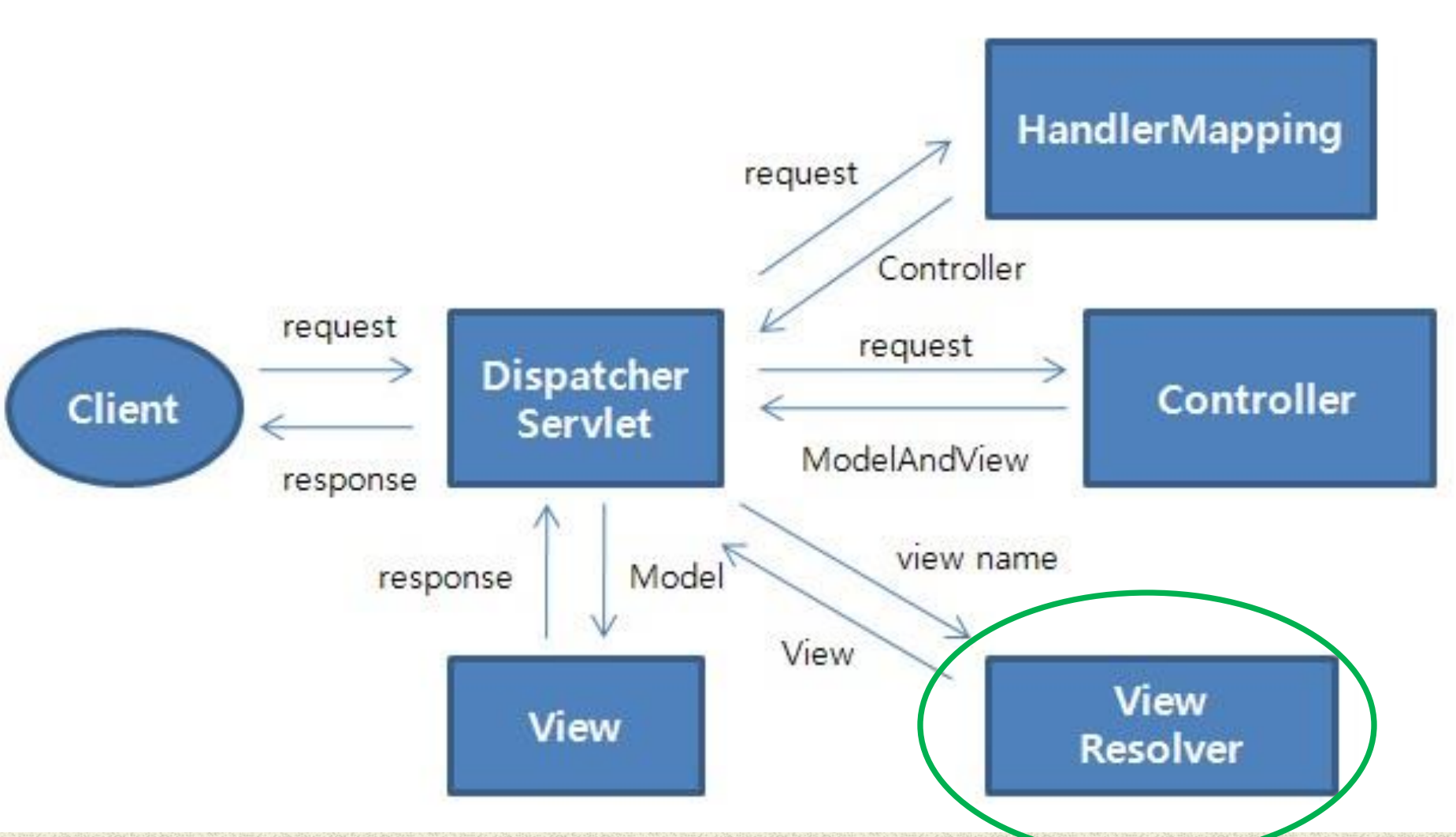
View Resolvers & Views & Upload & Exceptions & Internationalization & Tiles
Harmonizing Diversity

Spring MVC Flow



Spring MVC View Resolvers

- Flexible View Resolving Mechanism
- Resolve logical String-based view names to View types.
- Many out-of-the-box implementations[examples]:

1. **UrlBasedViewResolver**-This directly resolves a view name to a URL without any explicit mapping. The view names can be the URL themselves or a prefix or suffix can be added to get the URL from the view name.
2. **InternalResourceViewResolver**-This is a subclass of **UrlBasedViewResolver**. Out-of-the-box support for JSP
3. **FreeMarkerViewResolver**-This is a subclass of **UrlBasedViewResolver** that supports FreeMarkerView and its subclasses.
4. **VelocityViewResolver**-This is a subclass of **UrlBasedViewResolver** that supports VelocityView and its subclasses.
5. **ContentNegotiatingViewResolver**-This is an implementation of a view resolver based on the request file name or Accept header – mime-type. This class delegates view resolution to other view resolvers that are configured.

Multiple View Resolvers Configuration

```
<!-- lower order value has a higher priority -->
<bean id="internalViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/views/" />
<property name="suffix" value=".jsp" />
<property name="order" value="3" />
</bean>

<bean id="viewResolver"
class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
<property name="prefix" value=""/>
    <property name="suffix" value=".ftl"/>
<property name="order" value="2" />
</bean>
```

See webstoreFreVelTym

Spring MVC Views

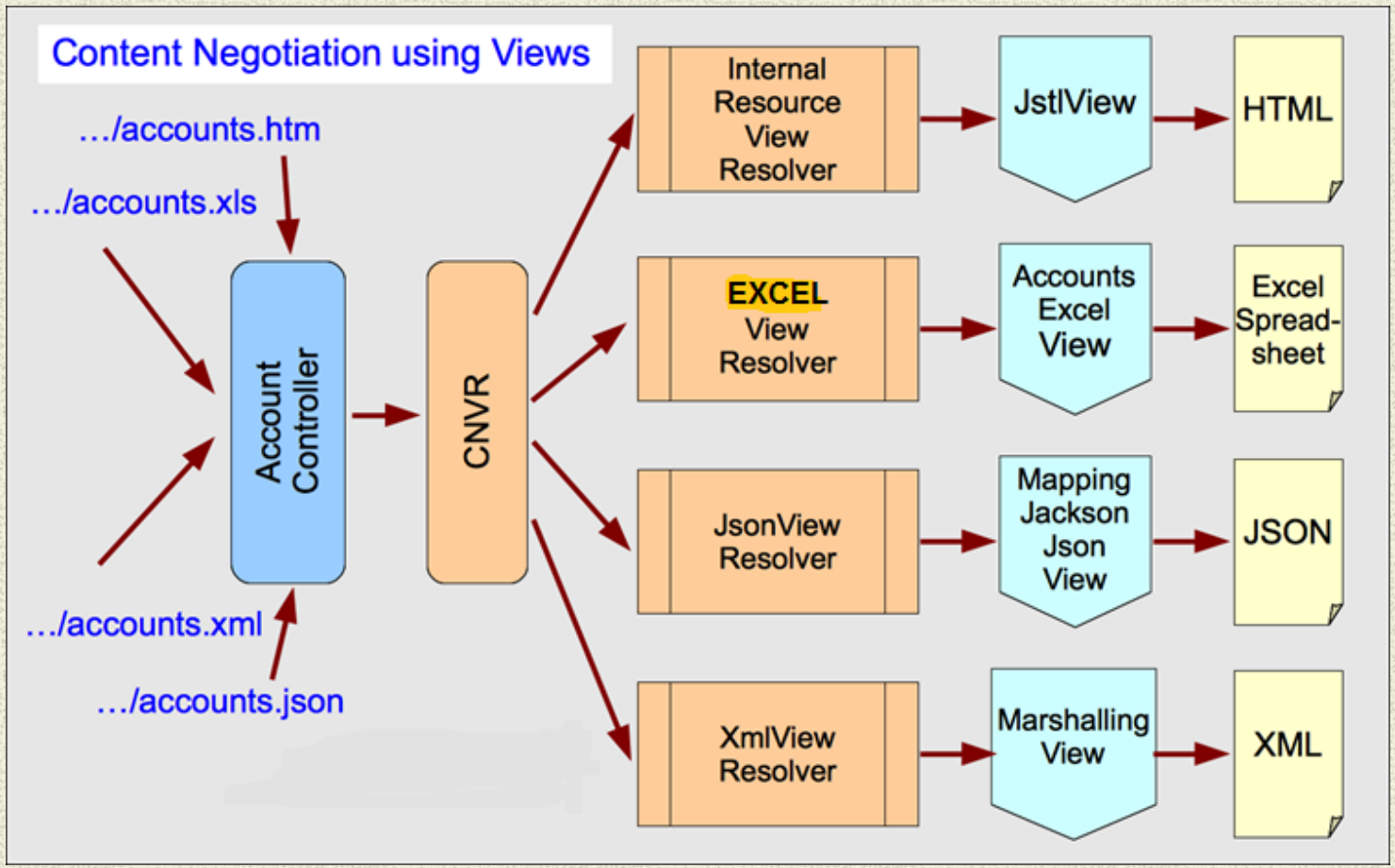
- Spring has flexible view support through the View Interface class
- Out-of-the-box view support for:
  - JspView - InternalResourceViewResolver
  - JSON - ContentNegotiatingViewResolver **
  - XML - ContentNegotiatingViewResolver **
  - PDF - ContentNegotiatingViewResolver **
  - Excel - ContentNegotiatingViewResolver **
  - Tiles - TilesViewResolver
  - Velocity - VelocityViewResolver
  - FreeMarker - FreeMarkerViewResolver

  - Redirect - InternalResourceViewResolver
  - Forward - InternalResourceViewResolver

** "Candidates"

ContentNegotiatingViewResolver

- Scenario based on “file extension” in URL:



- Single Controller method..multiple views…

CNVR

Resolving Image files for uploading

- HTTP multipart request is used by browsers to upload files/data to the server
- Spring good support HTTP multipart request:
  - CommonsMultipartResolver

DispatcherServlet XML config:

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <property name="maxUploadSize" value="10240000"/>
</bean>
```

JSP:

```
<form:form  modelAttribute="newProduct" class="form-horizontal"
    enctype="multipart/form-data">
```

Content-Type: multipart/form-data

```
<form:input id="productImage" path="productImage" type="file"
class="form:input-large" />
```

Resolving Image files for uploading[Cont.]

Saving image in Controller:

```
MultipartFile productImage = productToBeAdded.getProductImage();
String rootDirectory = servletContext.getRealPath("/");
```

rootDirectory == application root on Server

```
if (productImage!=null && !productImage.isEmpty()) {
  try {
       File outputfile =
         new File(rootDirectory+"\\resources\\images\\"
                          + newProduct.getProductId() + ".png");
      BufferedImage bufferedImage = ImageIO.read(productImage.getInputStream());
      ImageIO.write(bufferedImage, "PNG", outputfile);
} catch (Exception e) {
  throw new RuntimeException("Product Image saving failed", e);
```

Main Point

- Spring MVC Views and View Resolvers offers a variety of ways to manage the presentation of data.
- *Life is the expression of the field of all possibilities resulting in an explosion of variety in nature*

Handler Exception Resolver

- HandlerExceptionResolver interface
  - Used to resolve exceptions during Controller mapping & execution

    Two Default implementations [”out of the box”]:
    - ResponseStatusExceptionResolver – supports @ResponseStatus
    - ExceptionHandlerExceptionResolver – supports @ExceptionHandler

Exceptions can be handled EITHER individually OR Globally across ALL Controllers with @ControllerAdvice

ResponseStatusExceptionResolver

- Marks a method or exception class with the status code and reason that should be returned. “Customizes” exceptions as HTTP status codes
- The status code is applied to the HTTP response when the handler method is invoked, or whenever said exception is thrown.
- Could reside on Exception OR in @ControllerAdvice

```
@ResponseStatus Example Generates a 404 message:
@ResponseStatus(value=HttpStatus.NOT_FOUND,
                    reason="No products found under this category")
public class NoProductsFoundUnderCategoryException extends
RuntimeException{

ProductController “Test”:
if (products == null || products.isEmpty()) {
        throw new NoProductsFoundUnderCategoryException();
}
```

ExceptionHandlerExceptionResolver

- **Method identified as ExceptionHandler for exception resolution**

- **Could reside in EITHER ProductController OR @ControllerAdvice**

```
@ExceptionHandler(ProductNotFoundException.class)
public ModelAndView handleError(HttpServletRequest req,
        ProductNotFoundException exception) {
        ModelAndView mav = new ModelAndView();
        mav.addObject("invalidProductId", exception.getFullMessage());
        mav.setViewName("productNotFound");
        return mav;
}
```

- **JSP: productNotFound.jsp**

```
<h1 class="alert alert-danger"> ${invalidProductId}</h1>
```

Product Not Found Exception

```
public class ProductNotFoundException extends RuntimeException{
private String message = "No product found with the product ID = ";
private String productId;

public ProductNotFoundException(String productId, String message) {
    this.productId = productId;
    if (message != null) this.message = message;
}
public String getFullMessage() {
return (message + productId);
}
public String getProductId() {
      return productId;
}
```

@ControllerAdvice

- Indicates the annotated class assists a "Controller“
- Works across ALL controllers

- It is typically used to define **@ExceptionHandler**, @InitBinder, and @ModelAttribute methods that apply to all @RequestMapping methods.

- @ControllerAdvice
- **public class ControllerExceptionHandler {**

- Handles exceptions….

Demo: Test Exception

Product Controller "dummy" URL:

```
@RequestMapping(value = "/throw", method = RequestMethod.GET)
public String throwException(@ModelAttribute("newProduct") Product
newProduct) {

String productId = "B1234";
Product product = productService.getProductById(productId);
if(product == null) {
throw  new ProductNotFoundException(productId, null);
}
```

Main Point

- A well-defined exception and error handling approach is important for simplifying the development of web applications.
- *The removal of obstacles is an important aspect of the process of growth.*

Internationalization

- i18n – ‘i’+ 18 chars + ‘n’ == internationalization
- Support for multiple languages & data format with code rewrite
- Examples:
  - zh Chinese nl Dutch
  - hi Hindi el Greek
  - ja Japanese fr French
- L10n = ‘l’+10 chars + ‘n’ = localization
- Support locale-specific [geographic/region/country] information
  - Egypt EG Libya LY China CN
  - India IN Taiwan TW
  - Mynmar MM Mongolia MN

Java Locale class

- Locale(String language)
- Locale(String language, String Country)
- Locale(String language, String Country, String variant)

- Variant is browser specific code [windows, MAC, etc.]

- Message are stored in ".properties files indicating Locale
- E.g. messages_zh.properties
- Optionally messages_zh_CN.properties

Locale Resolvers

- Browser's Accept-Language header

```
<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver">
  <property name="defaultLocale" value="en_US"/>
</bean>
```

- Session
  - uses a locale attribute in the user's session

```
<bean id="localeResolver"
  class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
  <property name="defaultLocale" value="en_US"/>
</bean>
```

- Cookie
  - uses a cookie sent back to the user

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

LocaleChangeInterceptor

- Used to handle Cookie or Session locale resolvers AUTOMATICALLY

```
<mvc:interceptors>
   <bean class=
      "org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
    <property name="paramName" value="language"/>
   </bean>
</mvc:interceptors>
```

This is the parameter that the interceptor looks for...

Tiles

- **Composite View Pattern**

  create pages using a consistent structure

  pages share the same layout

  individual pages differ in segments

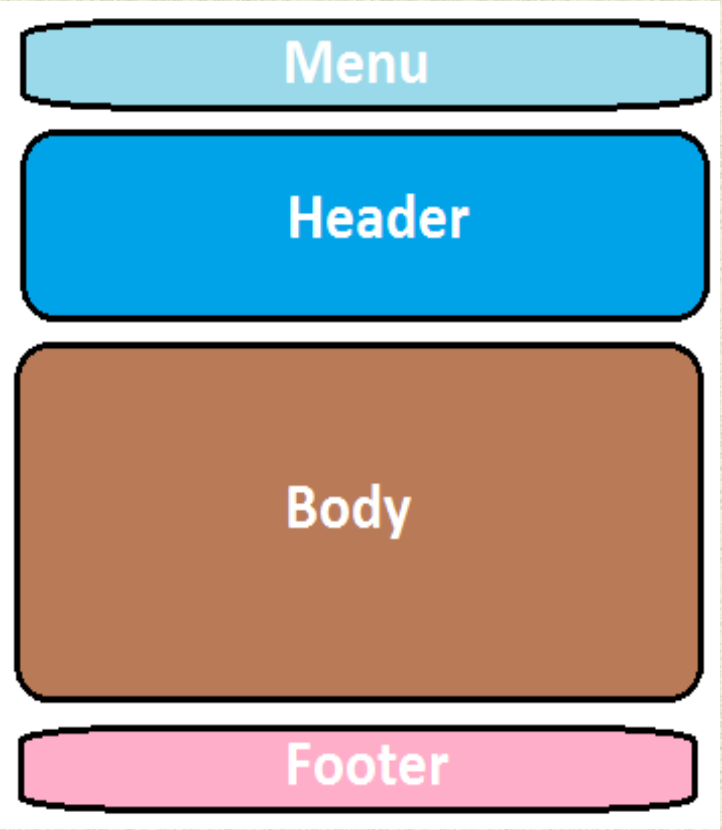
  segment placement maintains positional consistency across all the site.

Tiles View Resolver Configuration

```
<bean id="tilesViewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver">
  <property name="viewClass"
     value="org.springframework.web.servlet.view.tiles3.TilesView" />
  <property name="order" value="-2" />
</bean>
```

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

Sample Template [layoutTemplate.jsp]




```
<title><tiles:insertAttribute name="title" />
<body>
    <ul class="nav nav-pills pull-right">
     <tiles:insertAttribute name="menu" />
    </ul>
    <h3 class="text-muted">Web Store</h3>

    <h1>
     <tiles:insertAttribute name="header" />
    </h1>
    <p>
       <tiles:insertAttribute name="subHeader" />
    </p>

    <div class="row">
      <tiles:insertAttribute name="body" />
    </div>

    <div class="footer">
     <tiles:insertAttribute name="footer" />
    </div>
</body>
```

Example Tiles Definition File

Takes template file & “initializes” it

Sets up navigation bar & footer

```
<tiles-definitions>
    <definition name="baseLayout"
       template="/WEB-INF/tiles/template/layoutTemplate.jsp">
    <put-attribute name="title" value="Sample Title" />
    <put-attribute name=“menu"
          value="/WEB-INF/tiles/template/navigation.jsp" />
    <put-attribute name="header" value="" />
    <put-attribute name="subHeader" value="" />
    <put-attribute name=“body" value="" />
    <put-attribute name="footer"
          value="/WEB-INF/tiles/template/footer.jsp" />
</definition>
```

Example Tiles Pages

Welcome Page details

```
<definition name="welcome" extends="baseLayout">
    <put-attribute name="title" value="Welcome" />
    <put-attribute name="header" value="Internationalization" />
    <put-attribute name=“body" value="/WEB-INF/views/welcome.jsp" />
</definition>
```

Products Page details

```
<definition name="products" extends="baseLayout">
    <put-attribute name="title" value="Products" />
    <put-attribute name="header" value="Products" />
    <put-attribute name=“subHeader" value="Available Products" />
    <put-attribute name=“body" value="/WEB-INF/views/products.jsp" />
</definition>
```

Tiles Challenge: definition per page

Adding a new page requires a new definition in the tile-definition.xml file

```
<definition name="welcome" extends="baseLayout">
<definition name="products" extends="baseLayout">
<definition name="product" extends="baseLayout">
<definition name="addProduct" extends="baseLayout">
<definition name="login" extends="baseLayout">
<definition name="cart" extends="baseLayout">

etc., etc., etc...
```

Solution: Tile Wildcards

```
<!- This Wild Card works for multiple controller JSP references...
<definition name="*/*" extends="baseLayout">
     <put-attribute name="title" value="{2}.title" />
     <put-attribute name="heading" value="{2}.heading" />
     <put-attribute name="tagline" value="{2}.tagline" />
     <put-attribute name="body" value="/WEB-INF/views/{1}/{2}.jsp" />
</definition>
```

Property file references

ALSO in LayoutTemplate we can access attribute value

AND externalize the contents

```
<c:set var="heading"><tiles:getAsString name="heading" /></c:set>
```

Converts attribute value to a String

```
<spring:message code="${heading}"/>
```

Wildcard matches

Controller JSP name matches with wildcard in definition

- `<definition name="*" extends="baseLayout">`
- `return "welcome";`

- `<definition name="*/*" extends="baseLayout">`
- `return "/welcome";`
- `return “product/product";`

- `<definition name="*/*/*" extends="baseLayout">`
- `return "/welcome/";`
- `return "/welcome/foo";`

Main Point

- All good websites have something in common: they have an organized look & feel.
- *A facet of SCI is that Order is found everywhere*