Spring MVC

1. Giới thiệu

Spring's Web MVC framework được thiết kế xung quanh một DispatcherServlet t,nó gửi các request đến các handler, với việc cho phép cấu hình các handler mapping, view resolution, locale and theme resolution như là sự hỗ trợ tốt nhất cho việc upload file. Hanler mặc định rất đơn giản “Controller interface”, chỉ đưa ra một phương thức ModelAndView handleRequest(request,response). Cái này đã có thể được sử dụng cho các controller của ứng dụng, nhưng sẽ thích hơn khi bao gồm kiến trúc thực thi có thứ bậc, sự nhất quán, ví dụ AbstractController, AbstractCommandController and SimpleFormController. Các controller của ứng dụng sẽ là các lớp con tiêu biểu của chúng . Chú ý rằng chúng ta có thể chọn một lớp cơ sở nếu chúng ta không có một form, chúng ta không cần một form controller. Đây là điều khác biệt chính so với Strusts.

Spring Web MVC cho phép chúng ta sự dụng vài đối tượng như là một lệnh hoặc đối tượng form – không cần implement một framework-specific interface hoặc base class. Spring's data binding có tính mềm dẻo cao: ví dụ, nó đối xử các kiểu không hợp như là các validation error, điều này có thể được ước lượng bởi ứng dụng, không như các system error. Với tất cả những điều này có nghĩa là chúng ta không cần sao chép các property của đối tượng bussiness, không cần phải gõ các chuỗi vào trong form đối tượng chỉ để xử lí yêu cầu không phù hợp, hoặc chuyển đổi thành các chuỗi. Thay vào đó nó thường đưa ra sự liên kết trực tiếp đến đối tượng business của chúng ta. Đây là điểm khác biệt chính so với Struts, Strust được xây dựng xung quanh yêu cầu các lớp cơ sở như là Action và ActionForm.

So sánh với WebWork , spring có thêm các vai trò khác của đối tượng. Nó hỗ trợ thể sự thông báo của một controller, một lệnh tùy chọn hoặc một đối tượng form, và một mô hình để thông đến view. Mô hình sẽ đơn giản bao gồm lệnh và đối tượng form nhưng cũng có dữ liệu liên quan tùy ý; Thay vào đó, một WebWork Action kết hợp tất cả những vai trò đó vào trong một đối tượng riêng rẽ. WebWork cho phéo bạn sử dụng những đối tượng business của form, nhưng chỉ làm các thuộc tính bean cho các lớp Action tương ứng. Cuối cùng, những thể hiện Action giống nhau, sự xử lí request của nó được sử dụng cho việc đánh giá và xũa định vị trí của form trong View. Như vậy, dữ liệu liên quan cũng cần được mô hình như các thuộc tính bean của Action. Với những thứ như vậy, người ta có thể cho rằng có quá nhiều vai trò cho một đối tượng.

Giải pháp view của Spring mềm dẻo vô cùng. Một sự thực thi của một Controller có thể thậm chi có thể ghi một view trực tiếp vào response (bằng cách trả về giá trị null null cho ModelAndView). Trong trường hợp đơn giản, một thể hiện ModelAndView bao gồm một tên của view và một mô hình *ánh xạ(*Map), nó chứa đựng tên bean và những đối tượng tương ứng (going một lệnh hoặc một form, chứa đựng dữ liệu liên quan). Giải pháp dùng tên view có thể được cấu hình tốt, hoặc thông qua tên bean, một file properties, hoặc qua thông ViewResolver implementation của bạn. Thật sự, mô hình (Min MVC) được dựa trên Map interface, nó cho phép các công nghệ view hoàn toàn trừu tượng. Một vài renderer có thể được tích hợp trực tiếp, như là JSP, Velocity…. Mô hình Map được chuyển đổi một cách đơn giản thành một định dạng thích hợp, như là các thuộc tính JSP request hoặc một mẫu Velocity.

1.1 Pluggability of other MVC implementations

Có một vài lí do tại sao vài dự án thích sử dụng những MVC implement khác. Nhiều nhóm Nhiều nhóm phát triển muốn nâng cấp những hạng tầng hiện tại gồm các kĩ năng và công cụ. Hơn nữa, có một lượng lớn người hiểu biết và có kinh nghiệm về Strusts framework. Tuy nhiên, nếu bạn quen với kiến trúc Struts, nó vẫn có thể là một lựa chọn vững chắc cho web layer; cùng một kiểu áp dụng cho WebWork và những web MVC framework khác.

Nếu bạn không mốn sử dụng Spring's web MVC, nhưng có ý định sử dụng những giải pháp khác mà Spring đề nghị, bạn có thể tích hợp sự lựa chọn web MVC framework của bạn với Spring một cách dễ dàng. Đơn giản là khởi động một Spring root application context thông qua ContextLoaderListener của nó, và truy cập nó thông qua thuộc tính ServletContext của nó (or hoặc phương thức helper của Spring tương ứng) từ bên trong một Struts hoặc WebWork action. Chú ý, không có một vài "plugins" được gọi, vì thế không có sự tích hợp có tính chuyên môn nào là cần thiết. Từ điểm nhìn của web layer, bạn sẽ sử dụng một cách đơn giản Spring như là một thư viện, với thể hiện root application context như là một entry point(lối vào).

Tất cả những bean bạn đã đăng kí và các service của Spring có thể là các đầu ngón tay của bạn thậm chí không có Spring MVC. Spring không cạnh tranh với Struts or WebWork trong trường hợp này, nó chỉ xác định địa chỉ nhiều khu vực nơi mà sự trong sạch của web MVC frameworks không làm, Từ sự cấu hình bean đến truy cập dữ lieu và giao tác, xử lí giao tác. Vì thế bạn có thể làm giàu ứng dụng của mình với Spring middle tier hoặc/và data access tier, thậm chí nếu bạn chỉ muốn dùng, ví dụ như, transaction abstraction với JDBC hoặc Hibernate.

1.2 Features of Spring Web MVC

Spring WebFlow

Mục đích Spring Web Flow (SWF) là tìm giải pháp tốt nhất cho việc quản lí luồng của trang ứng dụng web (web application page flow).

SWF tích hợp với các framework đang tồn tại như Spring MVC, Struts, và JSF, trong cả môi trường. Nếu bạn có một business process (hoặc processes), nó có ít khi mà một mô hình giao tiếp như là bị đối nghịch thành một mô hình request một cách rõ ràng, SWF có thể là một giải pháp.

SWF cho phép bạn bắt các luồng logic của page như self-contained modules mà nó được sự dụng lại trong các hoàn cảnh khác, và như là ý tưởng cho việc xây dựng các module của ứng dụng web, nó hướng dẫn người dùng xuyên suốt điều khiển các navigation, những navigation này điều khiển business processes.

Tham khảo thêm về SWF tại trang Spring WebFlow site.

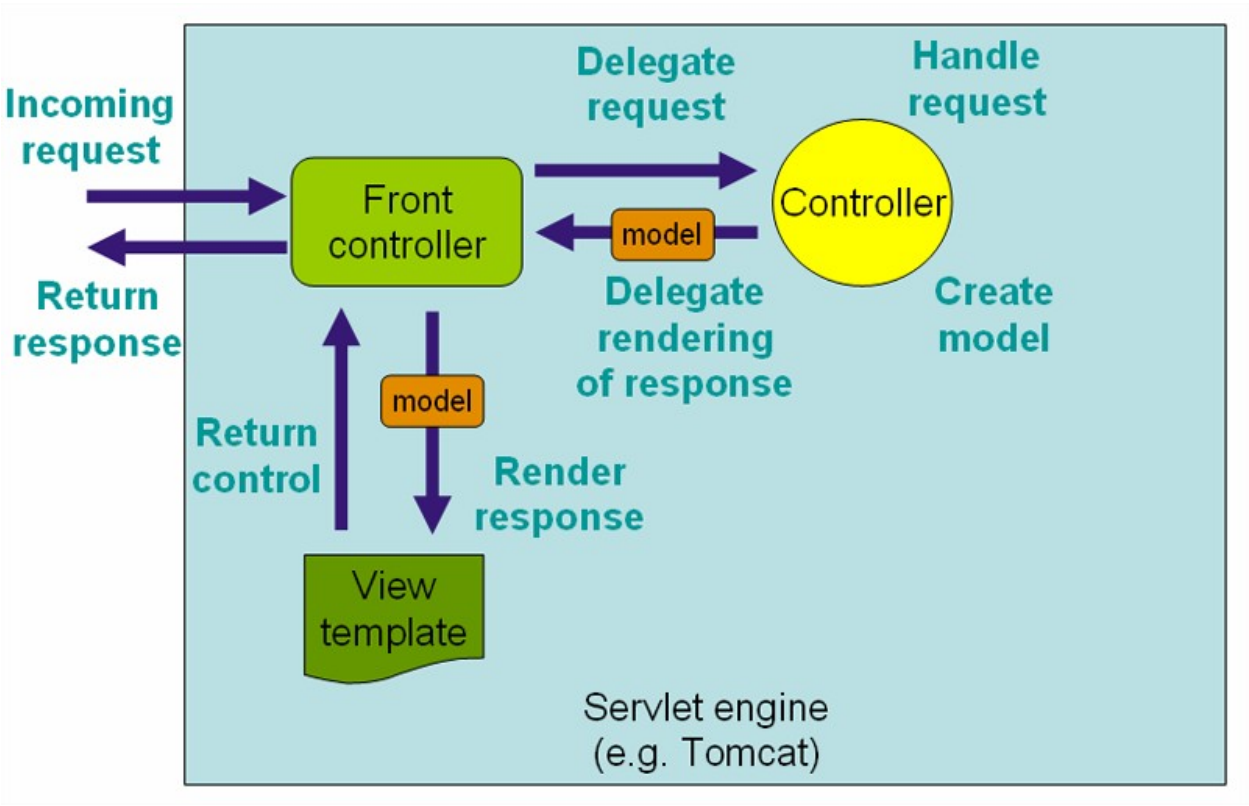
Các module của Spring web cung cấp một sự phong phú các feature duy nhất cho web, bao gồm:

- Phân chia rõ ràng các vai trò của controller (roles – controller), validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Mỗi vai trò có thể được thỏa mãn đầy đủ bởi một đối tượng chuyên dụng..
- Mạnh mẽ và sự cấu hình không phức tạp của cả framework và các lớp ứng dụng (application classes) như các JavaBean, bao gồm dễ dàng tham khảo (referencing) thông qua các context, Chẳng hạn như từ các web controller đến các đối tượng business (business object) và validators.
- Khả năng thích ứng, không xâm phạm (non-intrusiveness). Sử dụng bất cứ controller subclass gì bạn cần (plain, command, form, wizard, multi-action, hoặc một custom controller) để cung cấp một kịch bản thay vào thu được từ một controller đơn độc cho mọi thứ.
- Những business code có thể được sử dụng lại - không cần phải sao chép. Bạn có thể sử dụng những business object như là lệnh (command) hoặc form object thay vì phản ánh chúng theo để mở rộng một base class framework đặc biệt.
- Binding và validation có thể chỉnh sửa được – những sự không phù hợp kiểu (type mismatches) như là application-level validation errors, nó giữ những giá trị lỗi, localized date và number binding…v.v Thay vì chỉ là String form objects với sự phân tícch và chuyển đổi thành business object bằng tay.
- Handler mapping có thể tùy chỉnh và sự giải quyết bằng view – Các chiến lược handler mapping và sự giải quyết bằng view nhằm sắp xếp URL-based đơn giản thành phức tạp, các chiến lược cho mục đích xây dựng. Điều này làm cho nó mềm dẻo hơn những web MVC framework khách This is more flexible than some web MVC frameworks.
- Tính linh động của model transfer - model transfer thông qua tên/giá trị Map hỗ trợ cho việc dễ dàng tích hợp với một vài công nghệ view khác.
- Locale có thể chỉnh sửa và sự giải quyết bằng theme, hỗ trợ cho JSP với thư viện thẻ của Spring hoặc không, hỗ trợ cho JSTL, hỗ trợ cho Velocity không cần thiết phải có một cây cầu bắc qua công nghệ này, v.v.
- Một thư viện tag JSP còn mạnh mẽ được biết như là thư viện tag Spring, nó cung cấp sự hỗ trợ cho các tính năng như là data binding và themes. Các tag tự do được cho phép tối đa trong giới hạn mã đánh dấu(markup). Tìm hiểu thêm tag thư viện tag tại Appendix D, *spring.tld*
- Một thư viện tag JSP form được giới thiệu trong Spring 2.0, nó làm cho việc soạn thảo các form trong trang JSP dễ dàng hơn. Tìm hiểu thêm tại Appendix E, *spring-form.tld*
- Lifecycle của lifecycle is được xác định trong HTTP request hiện tại hoặc HTTP Session. Đây không phải là đặc tình của Spring MVC, nhưng thích WebApplicationContext container(s) hơn, Spring MVC sự dụng. Tìm hiểu thêm tại Section 3.4.4, “The other scopes”

DispatcherServlet

Spring's web MVC framework giống như nhiều web MVC frameworks, request-driven, được thiết kế xung quanh một servlet trung tâm, nó gửi các request đến các controller và đưa ra những chức năng dễ dàng cho sự phát triển của ứng dụng web. Spring DispatcherServlet bất cứ như thế nào , làm nhiều hơn là chỉ có vậy. Nó hoàn toàn được tích hợp với Spring IoC container và như vậy cho phép bạn sự dụng bất cứ đặc tính nào mà Spring có.

Luồng xử lí request của Spring Web MVC DispatcherServlet được minh họa trong lược đồ bên dưới. Người đọc có hiểu biết về mô hình sẽ nhận thấy rằng that the DispatcherServlet là một sự diễn tả của mẫu thiết kế “Front Controller” (đây là một mẫu mà Spring Web MVC chia sẽ với nhiều web frameworks dẫn đầu khác).

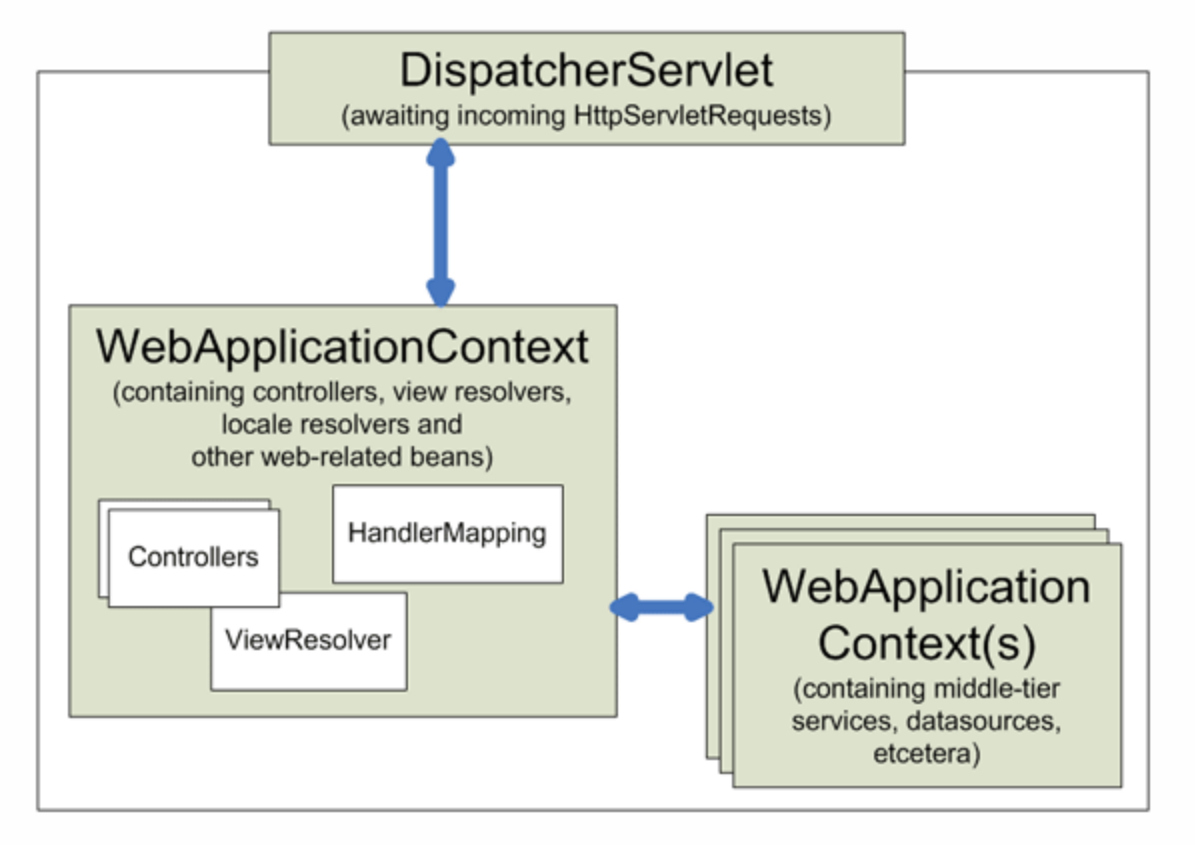


DispatcherServlet là một Servlet thật (nó kế thừa từ lớp cơ sở HttpServlet), và như vậy được khai báo trong web.xml của ứng dụng web của bạn. Những request mà bạn muốn DispatcherServlet xử lí sẽ được ánh xạ, sử dụng một URL mapping trong cùng file web.xml. Đây là chuẩn cấu hình của J2EE servlet; một ví dụ như là một sự khai báo của DispatcherServlet và mapping có thể được tìm thấy bên dưới.

```
<web-app>

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

   <servlet-mapping>
      <servlet-name>example</servlet-name>
      <url-pattern>*.form</url-pattern>
   </servlet-mapping>

</web-app>
```

Trong ví dụ trên, tất cả các request kết thúc với .form sẽ được xử lí bởi 'example' DispatcherServlet. Đây chỉ là bước khởi đầu để cấu hình trong sự thiết lập Spring Web MVC...Những bean khác nhau được sử dụng bởi Spring Web MVC framework (trên và DispatcherServlet trên của chính nó) cầu được cấu hình ngay.

Như được diễn tả chi tiết trong phần Section 3.8, “The ApplicationContext”, những thể hiện ApplicationContext trong Spring có thể được xác định phạm vi. Trong web MVC framework, mỗi DispatcherServlet có WebApplicationContext của chính nó, nó kế thừa tất cả các bean đã được định nghĩa trong root WebApplicationContext. Những thứ này được kế thừa từ các bean được định nghĩa sẵn, các bean này có thể bị overridden trong phạm vi servlet rõ ràng, và một phạm vi rõ ràng của các bean có thể được định nghĩa sẵn local để cho thể hiện của servlet .



Hệ thống ngữ cảnh Spring Web MVC

Framework sẽ, trên sự khởi tạo của một DispatcherServlet, tìm kiếm một tên file *[servlet-name]-servlet.xml* trong thư mục WEB-INF của ứng dụng web của bạn và tạo các bean được định nghĩa ở đó (overriding những sự định nghĩa của một vài bean được định nghĩa cùng tên trong phạm vi global).

Cân nhắc các DispatcherServlet servlet sau (trong file 'web.xml')

```
<web-app>
    ...
    <servlet>
        <servlet-name>golfing</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>golfing</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

Với vị trí trong sự cấu hình servlet trên, bạn sẽ cần có một file gọi là '/WEB-INF/**golfing**-servlet.xml' trong ứng dụng của bạn; file này sẽ chứa đựng tất cả các mô tả của các component(bean) của *Spring Web MVC*. Vị trí chính xác của các file cấu hình có thể bị thay đôi qua một tham số khởi tạo của servlet (Xem bên dưới để rõ hơn).

WebApplicationContext là một sự mở rộng của plain ApplicationContext, nó có một vài tính năng phụ cần thiết cho các web. Nó khác với một ApplicationContext bình thường, ở đó nó có khả năng giải quyết các theme (xem Section 13.7, “Using themes”), và rằng nó biết serlet nào nó được liên kết (bởi có một liên kết tới ServletContext). WebApplicationContext được bao bên trong ServletContext, và sử dụng những phương thức tĩnh trên lớp RequestContextUtils, bạn có thể luôn luôn tìm thấy WebApplicationContext trong trường hợp bạn cần truy cập đến nó.

Spring DispatcherServlet có một sự ghép nối với các bean đặc biệt, nó dùng để có thể xử lí các request và tạo ra những view thích hợp. Những bean đó được bao gồm bên trong Spring framework và có thể được cấu hình trong WebApplicationContext, chỉ như một vài bean khác cần được cấu hình. Mỗi cái bean này được mô tả chi tiết hơn bên dưới. Bây giờ, chúng ta sẽ chỉ đề cập đến chúng, vừa để bạn biết chúng tồn tại và có thể cho phép chúng ta nói về DispatcherServlet. Hầu hết các bean, có thể thấy được mặc định được cung cấp vì thế bạn không (initially) phải lo lắng về làm sao cấu hình chúng.

Table 13.1. Sự đặc tả bean trong the WebApplicationContext

Bean type	Explanation
Controllers	Controllers là các thành phần được tạo thành từ phần 'C' của MVC.
Handler mappings	Handler mappings xử lí sự thực thi của một danh sách các pre- và post-processors và các controllers sẽ được thực thi nếu chúng hợp với tiêu chuẩn chắc chắn (cho đối tượng một sự xứng hợp của một URL đặc biệt với controller)
View resolvers	View resolvers là các thành phần có khả năng giải quyết các tên view đến các view
Locale resolver	Một locale resolver là một thành phần có khả năng giải quyết vị trí một client là đang sử dụng, để có thể đưa đến tính quốc tế hóa các view (internationalized view)
Theme resolver	Một theme resolver có khả năng giải quyết các theme, ứng dụng dụng web của bạn có thể dùng, ví dụ, để đưa đến sự cá nhân hóa layout
multipart file resolver	Một multipart file resolver đưa ra các chức năng để xử lí file upload từ HTML form
Handler exception resolver(s)	Handler exception resolvers đưa ra chức năng ánh xạ các ngoại lệ đến các views hoặc thực hiện các xử lí ngoại lệ phức tạp hơn.

Khi một DispatcherServlet được cài đặt cho việc sử dụng và một request đến DispatcherServlet đặt biệt, người ta bảo rằng DispatcherServlet bắt đầu xử lí các request. Danh sách các mô tả sau mô tả đầy đủ một request đến khi được sử lí bởi một DispatcherServlet:

1. WebApplicationContext được tìm kiếm và bao trong request như một thuộc tính để cho controller và những nguyên tố khác trong quá trình xử lí sử dụng. Nó được bao mặc định dưới từ khóa DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE.
2. Vị trí của locale resolver được bao với request để các nguyên tố trong trong quá trình xử lí giải quyết locale để sử dụng khi xử lí request (tạo view, chuẩn bị dữ liệu,v.v..) nếu bạn không sử dụng resolver, nó sẽ không gây ảnh hưởng gì cả, vì thế nếu bạn không cần locale giải quyết , bạn không phải xử dụng nó.
3. Theme resolver được bao với request để các yếu tố như là các views xác đinh theme nào được sử dụng. Theme resolver không ảnh gì cả nếu bạn không dùng nó, vì thế bạn không cần theme thì chỉ cần bạn làm ngơ với nó.
4. Nếu một multipart resolver được đặc tả, request được duyệt cho các multipart; nếu các multipart được tìm thấy, request được bao phủ trong một MultipartHttpServletRequest cho quá trình xử lí xa hơn nữa bởi các yếu tố bên trong xử lí. (Xem Section 13.8.2, “Using the MultipartResolver” thêm thông tin về multipart handling).
5. Một handler thích hợp được tìm kiếm. Nếu một handler được tìm thấy, sự xử lí buộc liên kết với handler (các preprocessor, các postprocessor, và controller) sẽ được thực thi để chuẩn bị một a model (cho sự phát sinh [render]).
6. Nếu một model được trả về, view được phát sinh (render). Nếu không có mô hình nào được trả về (cái có thể được quy vào một pre- or postprocessor, phân hoạch request, ví dụ, cho lí do bảo mật), không có view được phát sinh (render), từ khi request có thể đã được đầy đủ.

Những ngoại lệ được quăng ra trong quá trình xử lí request nhận được một cách tình có bởi một vài handler exception resolver mà chúng được khai báo trong WebApplicationContext. Sự dụng những exception resolver đó cho phép bạn định nghĩa những hành vi một cách tùy biến trong trường hợp như các exception có được bởi sự quăng ném.

Spring DispatcherServlet cũng hỗ trợ trả về *last-modification-date*, như là được đặc tả bởi Servlet API. Tiến trình của sự xác định ngày chỉnh sửa cuối cùng cho một request đặc biệt thì không phức tạp: DispatcherServlet sẽ tìm kiếm một handler mapping thích

hợp và kiểm thử nếu handler được tìm thấy interface *implements the interface LastModified*. Nếu thế, giá trị của phương thức getLastModified(request) của interface LastModified được trả về cho client.

Bạn có thể tùy chỉnh Spring's DispatcherServlet bằng cách thêm các tham số context vào file web.xml hoặc sự khởi tạo các tham số của servlet. Những khả năng có thể được liệt kê như sau.

Table 13.2. Sự khởi tạo các tham số DispatcherServlet.

Parameter	Explanation
contextClass	Lớp thực thi ApplicationContext, nó sẽ được dùng để khởi tạo ngữ cảnh được sử dụng bởi servlet này. Nếu tham số này không được đặc tả, XmlWebApplicationContext sẽ được sử dụng.
contextConfigLocation	String được chuyển đến ngữ cảnh của thể hiện (context instance) (được đặc tả bởi contextClass) để chỉ ra nơi context(s) có thể được tìm thấy. Chuỗi có khả năng phân chia thành nhiều chuỗi (sự dụng một dấu phẩy để tách) để hỗ trợ cho nhiều context (trong trường hợp nhiều khu vực của context, của các bean được định nghĩa hai lần, cái sau cùng được ưu tiên).
namespace	namespace của WebApplicationContext. Mặc đinh là [servlet-name]-servlet.

13.3. Controllers

Sự thông báo của một controller là một phần trong thiết kế của mô hình MVC (Chính xác hơn nó là *'C' trong* MVC. Các Controller cung cấp chế độ truy cập vào ứng dụng mà nó được định nghĩa tiêu biểu bởi một service interface. Các Controller giải thích user input và chuyển đổi thành dạng input có thể hiểu được mà nó sẽ được trình diễn cho user bằng view. Spring đã thực thi sự thông báo của một controller trong một cách chung chung trừu tượng, cho phép một lượng lớn các loại controller được tao tạo ra. Spring chứa đựng đặc tả form (form-specific) của các controller, các command-based controller, và các controller mà nó thực hiện wizard-style logic, thành tên nhưng một ít.

Cơ sở của Spring cho kiến trúc controller là interface org.springframework.web.servlet.mvc.Controller, Source code cho nó được liệt kê bên dưới.

```
public interface Controller {

    /**
     * Process the request and return a ModelAndView object which the DispatcherServlet
     * will render.
     */
    ModelAndView handleRequest(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception;

}
```

Như bạn có thể thấy, Controller interface định nghĩa một phương thức đơn độc chịu trách nhiệm cho việc xử lí một request và trả về một model và view thích hợp. Ba khái niệm đó là cơ sớ cho Spring MVC implementation - ModelAndView và Controller. Trong khi Controller interface là khá trừu tượng, Spring đưa ra một lượng các Controller implementation có thể sử dụng dễ dàng và nó đã chứa đựng một lượng các chức năng mà bạn cần.Controller interface chỉ định nghĩa yêu cầu trách nhiệm cơ bản cho mỗi controller; là xử lí một request và trả về một model và một view.

13.3.1. AbstractController and WebContentGenerator

Để cung cấp một cơ sở hạ tâng, tất cả những Controller khác nhau của Spring kế thừa từ AbstractController, một lớp đưa ra sự hỗ trợ caching và, ví dụe, cài đặt cho mimetype.

Table 13.3. Những đặc tính được đưa ra bởi AbstractController

Feature	Explanation
supportedMethods	Chỉ cho biết những phương thức nào controller sẽ chấp nhận. Luôn luôn nó đặt cho cả hai GET và POST, nhưng bạn có thể chỉnh sửa cái này để mang lại phương thức bạn muốn hỗ trợ. Nếu một request được nhận với một method mà không được hỗ trợ bởi controller, Client sẽ được thông báo về điều này (một ServletException sẽ được ném ra).
requiresSession	Chỉ ra rằng control này yêu cầu một HTTP session để làm việc của nó. Nếu một session không được biểu diễn khi mà một controller nhận một request, người dùng sẽ được thông tin về điều này bằng một ServletException được ném ra.
synchronizeSession	Dùng cái này nếu bạn muốn xử lí bởi controller để đồng bộ với HTTP session của người dùng.
cacheSeconds	Khi bạn muốn một controller phát sinh một chỉ thị caching trong HTTP response, định rõ một con số rõ ràng ở đây (integer). Mặc định giá trị của thuôc tính này là -1 vì thế chỉ thị bảo ràng không có caching phát sinh response.
useExpiresHeader	Chỉnh sửa các controller của bạn để đặc tả HTTP 1.0 compatible *"Expires"* header khi mà tạo response. Mặc định giá trị thuộc tính này là true.
useCacheHeader	Chỉnh sửa các controller của bạn để đặc tả HTTP 1.1 compatible *"Cache-Control"* header khi mà tạo ra response. Mặc định giá trị này là true.

Khi sử dụng AbstractController như là lớp cơ sở cho controller của bạn, bạn chỉ phải override phương thức handleRequestInternal(HttpServletRequest, HttpServletResponse), thi hành theo sự logic của bạn, và trả về một đối tượng ModelAndView. Đây là một ví dụ ngắn gồm có một lớp và được khai báo trong web application context.

```
package samples;

public class SampleController extends AbstractController {

    public ModelAndView handleRequestInternal(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception {

        ModelAndView mav = new ModelAndView("hello");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}
<bean id="sampleController" class="samples.SampleController">
    <property name="cacheSeconds" value="120"/>
</bean>
```

Lớp trên và sự khai báo trongweb application context là tất cả những gì bạn cần bên cạnh sự thiết lập một handler mapping (xem thêm Section 13.4, “Handler mappings”) để làm cho controller rất đơn giản này làm việc. Cotrolller này sẽ phát sinh chỉ thị caching bảo với client rằng cache mọi thứ trong vòng 2 phút trước khi xem lại. Controller nàu cũng trả về một hard-coded view (nó được cân nhắc điển hình cho sự thực hành tệ).

13.3.2. Những controller đơn giản khác

Mặc dù bạn có thể mở rộng AbstractController, Spring cung cấp một số các implementation cụ thể, mà nó đưa ra chức năng chung chung được sử dụng trong ứng dúng MVC đơn giản. ParameterizableViewController là cơ bản, cũng như ví dụ trên, ngoại trừ cho sự thật là bạn có thể đặc tả tên view mà nó sẽ trả về trong web application context (và như vậy xóa đi sự cần thiết để hard-code viewname trong class java).

UrlFilenameViewController kiểm tra URL và lấy tên file của request và sử dụng nó như một tên view. Ví dụ, tên file của http://www.springframework.org/index.html request là index.

13.3.3. MultiActionController

Spring đưa ra một multi-action controller với nó bạn tập hợp nhiều action vào trong một controller, như vậy nhóm các chức năng với nhau. Multi-action controller trong org.springframework.web.servlet.mvc.multiaction – và là khả năng ánh xạ các request đến tên phuong thức và gọi đúng tên phương thức. Sử dụng controller multi-action thì đặc biệt thuận tiện khi bạn có nhiều chức năng phổ biến trong một controller, nhưng muốn có nhiều mục (multiple entry) trỏ vào controller, ví dụ

Table 13.4. Những đặc tính được đưa ra bởi MultiActionController

Feature	Explanation
delegate	Có hai kịch bản sử dụng với MultiActionController. Hoặc là bạn tạo ra một subclass của MultiActionController và đặc tả những phương thưc sẽ được giải quyết bởi MethodNameResolver trên subclass (trong trường hợp này bạn không cần đặt delegate), hoặc bạn định nghĩa một đối tượng, trên phương thức này s resolved by the MethodNameResolver will be invoked. If you choose this scenario, you will have to define the delegate using this configuration parameter as a collaborator.
methodNameResolver	MultiActionController cần một chiến lược giải quyết phương thức nó có gọi, dựa trên các request đi vào. Chiến lược này được định nghĩa bởi interface MethodNameResolver; MultiActionController phơi bày một thuộc tính sp mà bạn gọi thay thế một resolver mà có thể làm công việc này.

Những phương thức được định nghĩa cho một multi-action controller cần tuân theo kiểu sau:

```
// anyMeaningfulName can be replaced by any methodname
public [ModelAndView | Map | void] anyMeaningfulName(HttpServletRequest, HttpServletResponse [, Exception | AnyObject]);
```

Xin chú ý rằng kiểu nạp chồng phương thức là không được cho phép (overloading method) khi đó nó sẽ làm cho MultiActionController nhầm lẫn. Hơn nữa, bạn có thể định nghĩa *exception handlers* có khả năng xử lí các exception mà được ném ra từ phương thức mà bạn đặc tả.

Tham số Exception có thể là một vài exception, xa hơn nữa nó là một subclass của java.lang.Exception or java.lang.RuntimeException. Tham số của AnyObject có thể là một vài class. Những thàm số Request sẽ được bao lại bên trong đối tượng này cho sự sử dụng thích hợp.

Tìm kiếm một vài ví dụ hợp lệ về phương thức MultiActionController.

Kiểu chuẩn (phương thức interface của Controller).

```
public ModelAndView doRequest(HttpServletRequest, HttpServletResponse)
```

Kiểu này chấp nhận một tham số Login mà sẽ được xác định với tham số từ request.

```
public ModelAndView doLogin(HttpServletRequest, HttpServletResponse, Login)
```

Kiểu phương thức có xử lí Exception.

```
public ModelAndView processException(HttpServletRequest, HttpServletResponse, IllegalArgumentException)
```

Kiểu trả về kiểu void (tìm hiểu thêm tại Section 13.11, “Convention over configuration”).

```
public void goHome(HttpServletRequest, HttpServletResponse)
```

Kiểu này trả về một kiểu Map(tìm hiểu thêm tại Section 13.11, “Convention over configuration”).

```
public Map doRequest(HttpServletRequest, HttpServletResponse)
```

MethodNameResolver là phương thức chịu trách nhiệm xử lí tên dựa trên request vào. Xem chi tiết bên dưới về 3 kiểu implement của MethodNameResolver mà Spring cung cấp mà không có ràng buộc.

- ParameterMethodNameResolver – có khả năng giải quyết một tham số request và sự dụng nó như tên phương thức (http://www.sf.net/index.view?testParam=testIt sẽ có kết quả trong một phương thức testIt(HttpServletRequest, HttpServletResponse) đang được gọ). Thuộc tính paramName property đặc tả tham số request mà được kiểm tra).
- InternalPathMethodNameResolver – lấy filename từ request path và sử dụng nó như là tên phương thức (http://www.sf.net/testing.view sẽ có kết quả trong một phương thức testing(HttpServletRequest, HttpServletResponse) đang được gọi).
- PropertiesMethodNameResolver – sử dụng một đối tượng user-defined properties với request URLs được ánh xạ đến tên phương thức. Khi properties chứa đựng /index/welcome.html=doIt và một request đến /index/welcome.html vào, phương thức doIt(HttpServletRequest, HttpServletResponse) được gọi. Tên phương thức này của resolver làm việc với PathMatcher, vì thế nếu properties chứa /**/welcom?.html, Nó cũng sẽ làm việc!

Đây là một ví dụ. Trước tiên, một ví dụ hiện thị ParameterMethodNameResolver và delegate property, nó sẽ chấp nhận những request đến URLs với tham số phương thức được lồng vào và đặt để lấy Index:

```
<bean id="paramResolver" class="org....mvc.multiaction.ParameterMethodNameResolver">
  <property name="paramName" value="method"/>
</bean>

<bean id="paramMultiController" class="org....mvc.multiaction.MultiActionController">
  <property name="methodNameResolver" ref="paramResolver"/>
  <property name="delegate" ref="sampleDelegate"/>
</bean>

<bean id="sampleDelegate" class="samples.SampleDelegate"/>

## together with

public class SampleDelegate {

    public ModelAndView retrieveIndex(HttpServletRequest req, HttpServletResponse resp) {

        return new ModelAndView("index", "date", new Long(System.currentTimeMillis()));
    }
}
```

Khi sử dụng delegate được biểu diễn ở trên, chúng ta có thể sử dụng PropertiesMethodNameResolver để kiếm một sự xứng hợp của URLs với phương thức mà chúng ta định nghĩa:

```
<bean id="propsResolver" class="org....mvc.multiaction.PropertiesMethodNameResolver">
  <property name="mappings">
    <value>
       /index/welcome.html=retrieveIndex
       /**/notwelcome.html=retrieveIndex
       /*/user?.html=retrieveIndex
    </value>
  </property>
</bean>

<bean id="paramMultiController" class="org....mvc.multiaction.MultiActionController">
    <property name="methodNameResolver" ref="propsResolver"/>
    <property name="delegate" ref="sampleDelegate"/>
</bean>
```

13.3.4. Lệnh của các controller (command controller)

Spring's *command controllers* là một phần cơ sở của gói Spring Web MVC. Command controllers cung cấp một lối để tương tác với dữ liệu của các đối tượng và liên kết động các tham số (parameter) từ HttpServletRequest đến dữ liệu của đối tượng được đặc tả. Chúng thi hành hơi giống role của Struts ActionForm, nhưng trong Spring, dữ liệu đối tượng của bạn không phải thực thi một interface của framework đặc tả. Trước tiên, chúng ta xem xét command controller nào là sẵn sàng được dùng một cách dễ dàng nhất mà không cần cấu hình cài đặt.

- AbstractCommandController - một command controller, bạn có thể sử dụng tạo ra command controller của riêng bạn, có khả năng liên kết các tham số request vào đối tượng dữ liệu bạn đặc tả. Class này không đưa ra hình dạng của chức năng; nó làm bằng cách nào đó đưa ra tính năng kiểm chứng (validation) và để bạn đặc tả trong controller của chính nó những gì làm với đối tượng command mà đã được xác định với giá trị tham số của request.
- AbstractFormController – một controller trừu tượng đưa ra sự hỗ trợ form submission. Sử dụng controller này bạn có thể mô hình các form và xác định chúng đang sử dụng một đối tượng Command bạn lấy từ controller. Sau khi một user điền đầy form, AbstractFormController liên kết các trường (field), Kiểm chứng (validate) đối tượng command, và điều khiểu các đối tượng trở lại controller để lấy action thích hợp. Các đặc tính được hỗ trợ là: invalid form submission (resubmission), validation, và normal form workflow. Bạn thực thi phương tức để xác định view nào là được sử dụng cho form presentation và thành công. Sử dụng controller nếu bạn cần các form, nhưng không muốn đặc tả view giẽ hiện thị user trong application context.
- SimpleFormController – một form controller cung cấp sự hỗ trợ thậm chí nhiều hơn khi tạo một form với một đối tượng corresponding command. SimpleFormController để cho bạn đặc tả một command object, một viewname cho form, một viewname cho trang bạn muốn hiện thị user khi form submission đã thành công, và thêm nữa.
- AbstractWizardFormController – như tên class được đề nghị, đây là một abstract class - wizard controller của bạn nên kế thừa nó. Điều này có nghĩa là bạn phải thực thi phương thức validatePage(), processFinish() và processCancel().

  Bạn hầu như chắc chắn rằng muốn viết một contractor, nó sẽ là cái tối thiểu nhất để gọi setPages() và setCommandName(). Former lấy tham số như là một mảng kiểu String. Mảng này là danh sách các view mà nó bao gồm wizard của bạn. Sau cùng lấy tham số của nó như một kiểu chuỗi, Nó sẽ được sử dụng để chỉ dẫn đối tượng command từ trong các view của bạn.

  Như với một vài thể hiện của AbstractFormController, bạn sẽ được yêu cầu sử dụng một đối tượng command – một JavaBean mà nó sẽ được định vị với dữ liệu từ những form của bạn. Bạn có thể làm cái này theo một trong hai cách sau: hoặc là gọi setCommandClass() từ constructor với class của đối tượng command, hoặc thực thi phương thức formBackingObject().

  AbstractWizardFormController có một số phương thức cụ thể mà bạn có thể override. Trong số này, cái mà thích tìm kiếm và rất hữu dụng là: referenceData(..) bạn có thể sử dụng để thông qua model data đến view trong form của một Map; getTargetPage() nếu wizard của bạn cần thay đổi thứ tự trang hoặc bỏ sót các trang động; và onBindAndValidate() nếu bạn muốn override built-in binding và validation workflow.

  Cuối cùng, nó đáng giá trỏ ra ngoài setAllowDirtyBack() và setAllowDirtyForward(), mà bạn có thể gọi từ getTargetPage() để cho phép những người dùng backwards và forwards trong wizard thậm chí nếu nếu kiểm chứng được xác nhận là không đạt (validation fail) cho trang hiện tại.

  For a full list of methods, see the Javadoc for AbstractWizardFormController. There is an implemented example of this wizard in the jPetStore included in the Spring distribution: org.springframework.samples.jpetstore.web.spring.OrderFormController.

13.4. Handler mappings

Sự dụng một handler mapping chúng ta có thể ánh xạ một web request đến một handler thích hợp. Có một vài handler mapping chúng ta có thể sử dụng một cách tự nhiên, ví dụ: SimpleUrlHandlerMapping hoặc BeanNameUrlHandlerMapping.

Chức năng của một HandlerMapping cơ bản cung cấp là sự phân phối của một HandlerExecutionChain, Nó cần chứa handler phù hợp với request, và cũng có thể chứa đựng một danh sách các handler chặn mà được áp dụng cho request. Khi một request vào, DispatcherServlet sẽ điều khiển nó thông qua handler mapping để phân tích và kiểm tra request và mang lên với một HandlerExecutionChain thích hợp. Sau đó DispatcherServlet sẽ thực thi handler và các bộ chặn trong chuỗi thực thi (nếu có).

Khái niệm configurable handler mappings là có thể tùy ý chứa đựng các bộ chặn (thực thi trước khi và sau khi handler thực sự được thực thi, hoặc cả hai) là thực sự mạnh mẽ. Khối lượng các chức năng được hỗ trợ có thể được xây dựng trong việc xây dựng các HandlerMapping tùy ý. Cách xử lí của một handler mapping tùy ý là một handler không chị dựa trên URL của request mà nó cũng mô tả trạng thái của session liên kết với request.

Phần này mô tả hai handler mapping thông dụng. Chúng mở rộng từ AbstractHandlerMapping và cùng có những thuộc tính sau:

- interceptors: danh sách các bộ chặn để dùng. Xem thêm Section 13.4.3, “Intercepting requests - the HandlerInterceptor interface”.
- defaultHandler: default handler để dùng, Khi handler mapping này không hợp với handler nào.
- order: dựa trên giá trị của thuộc tính order (xem org.springframework.core.Ordered interface), Spring sẽ sắp xếp tất cả các handler mappings đang có trong context và áp dụng handler đầu tiên.

- alwaysUseFullPath: nếu thuộc tính này được set là true, Spring sẽ sự dụng đường dẫn đầy đủ servlet context hiện tại để tìm handler thích hợp. Nếu thuộc tính này được set là false (mặc định), đường dẫn bên trong servlet mapping hiện tại sẽ được sử dụng . Ví dụ, Nếu một servlet được ánh xạ sử dụng /testing/* và alwaysUseFullPath được set là true, /testing/viewPage.html sẽ được sử dụng, Trái lại nếu thuộc tính được set là false, /viewPage.html sẽ được sử dụng.
- urlPathHelper: sử dụng thuộc tính này, bạn có thể sử dụng UrlPathHelper khi kiểm tra URLs. Bình thường, Bạn khoong nên thay đổi giá trị mặc định.
- urlDecode: Giá trị mặc định của thuộc tính này là false. HttpServletRequest trả về request URLs và URIs mà không được decoded. Nếu bạn không muốn chúng được mã hóa trước khi HandlerMapping sự dụng chúng để tìm một handler thích hợp , bạn phải đặt thuộc tính này là true (nó yêu cầu JDK 1.4). Phương thức decode sử dụng hoặc là encoding được đặc tả bởi request hoặc default ISO-8859-1 encoding .
- lazyInitHandlers: allows for lazy initialization of *singleton* handlers (prototype handlers are always lazily initialized). Default value is false.

13.5.2. Chaining ViewResolvers

Spring hỗ trợ nhiều view resolver, và chúng được sắp xếp một cách thứ tự theo một chuỗi mắc xích. Trong ví dụ bên dưới, chuỗi mắc xích của chúng ta có hai view (InternalResourceViewResolver):

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

<bean id="excelViewResolver" class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="order" value="1"/>
  <property name="location" value="/WEB-INF/views.xml"/>
</bean>

<!-- in views.xml -->

<beans>
  <bean name="report" class="org.springframework.example.ReportExcelView"/>
</beans>
```

Nếu một view resolver cụ thể không tìm thấy view nào thích hợp, Spring sẽ phân tích trong context để xem có view resolver nào được cấu hình không. Nếu có, nó sẽ tiếp tục phân tích yêu cầu, trái lại nó sẽ trả về một ngoại lệ.

Khi viết một view resolver, chúng ta cần phải xác tính đến các ngoại lệ mà view resolver trả về

13.5.3. Redirecting to views

Với JSP, nó được xử lí thông qua Servlet/JSP engine, điều này được xử lí đơn giản thông qua InternalResourceViewResolver / InternalResourceView, nó sử dụng một trong các phương thức sau RequestDispatcher.forward(..) hoặc RequestDispatcher.include().

13.5.3.1. RedirectView

Để sử dụng cơ chế redirect, Spring cung cấp RedirectView . Nó nhận kết quả của phương thức HttpServletResponse.sendRedirect() để xử lí

The RedirectView simply ends up issuing an HttpServletResponse.sendRedirect() call, which will come back to the client browser as an HTTP redirect. All model attributes are simply exposed as HTTP query parameters. This does mean that the model must contain only objects (generally Strings or convertible to Strings) which can be readily converted to a string-form HTTP query parameter.

13.5.3.2. The redirect: prefix

Nếu controller tạo RedirectView trên chính nó, thì sẽ controller sẽ không nhận biết được là có redirect. Controller chỉ giải quyết trên tên của view đã được chèn vào.

Redirect cụ thể: prefix cho phép điều này đạt được. Nếu tên view được trả về có tiền tố là redirect, khi đó UrlBasedViewResolver (và tất cả các lớp con) sẽ nhận ra đây là là một redirect. View đó sẽ được thiết đãi như là một redirect URL.

13.5.3.3. The forward: prefix

Forward: prefix được xử lí bởi UrlBasedViewResolver và subclasses. Tất cả những gì nó làm là tạo một InternalResourceView (băng cách gọi RequestDispatcher.forward()) . vì vậy không phải sử dụng prefix này khi sử dụng InternalResourceViewResolver / InternalResourceView.

Như redirect prefix, controller sẽ không phải nhận biết rằng có một redirect nếu đã chèn tên view vào controller

13.6. Sử dụng locales

Các phần của Spring đều hỗ trợ internationalization. DispatcherServlet cho phép bạn xử lí các message tự động bằng cách sử dụng vị trí của client. Nó được làm bởi LocaleResolver objects.

Khi request đến, DispatcherServlet tìm kiếm một locale resolver và nếu nó tìm thấy nó cố gắng sử dụng locale resolver này để đặt locale. Sử dụng phương thức RequestContext.getLocale().

Bên cạch việc giải quyết các locale tự động bạn cũng có thể đính kèm một bộ chặn đến handler mapping để thay đổi locale theo một tình huống đặc biệt

Locale resolvers và interceptors được định nghĩa trong gói org.springframework.web.servlet.i18n, và được cấu hình trong application context theo cách đơn giản

13.6.1. AcceptHeaderLocaleResolver

Locale resolver phân tích accept-language header trong request.

13.6.2. CookieLocaleResolver

Locale resolver phân tích một Cookie. Nếu cookie được sử dụng nó sử dụng local mà cookie đã đặc tả

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver">

    <property name="cookieName" value="clientlanguage"/>

    <!-- in seconds. If set to -1, the cookie is not persisted (deleted when browser shuts down) -->
    <property name="cookieMaxAge" value="100000">

</bean>
```

Table 13.6. CookieLocaleResolver properties

Property	Default	Description
cookieName	classname + LOCALE	Tên của cookie
cookieMaxAge	Integer.MAX_INT	Thời gian tối đa mà cookie có hiệu lực trên client. Nếu là -1 thì cookie sẽ không được lưu lại lâu. Nó chỉ có hiệu lực cho tới khi người dùng tắt browser.
cookiePath	/	Sử dụng tham số này chúng ta có thể giới hạn tính chất riêng tư của cookie. Khi cookiePath được đặt tả, Cookie chỉ được thể hiện trong được trông thấy ở đường dẫn này.

13.6.3. SessionLocaleResolver

SessionLocaleResolver cho phép bạn lấy các locale từ session.

13.6.4. LocaleChangeInterceptor

Bạn có thể xây dựng sử thay đổi của locale bằng cách sử dụng LocaleChangeInterceptor. Bộ chặn này cần được thêm vào handler mappings, trong request và thay đổi locale (nó gọi setLocale() của LocaleResolver, mà nó cũng tồn tại trong ngữ cảnh này).

```
<bean id="localeChangeInterceptor"
     class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
   <property name="paramName" value="siteLanguage"/>
</bean>

<bean id="localeResolver"
     class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>

<bean id="urlMapping"
     class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="interceptors">
      <list>
         <ref bean="localeChangeInterceptor"/>
      </list>
   </property>
   <property name="mappings">
      <value>/**/*.view=someController</value>
   </property>
</bean>
```

Tất gọi đến tất cả *.view chứa một tham số có tên là siteLanguage.Ví dụ: http://www.sf.net/home.view?siteLanguage=nl sẽ thay đổi ngôn ngữ của site thành Dutch.

13.7. Sử dụng themes

13.7.1. Giới thiệu

T*heme* được Spring web MVC framework cung cấp, cho phép bạn nâng cấp kinh nghiệm của người dùng bằng việc làm cho ứng dụng của bạn look and feel bởi theme. Theme cơ bản là một tập hợp các tài nguyên tĩnh như là hình, style sheet….

13.7.2. Định nghĩa themes

Cấu hình org.springframework.ui.context.ThemeSource. WebApplicationContext interface kế thừa ThemeSource nhưng ủy nhiệm nó chịu trách nhiệm thực thi. Mặc định sự ủy nhiệm này sẽ là org.springframework.ui.context.support.ResourceBundleThemeSource, lớp này lấy các file properties từ classpath gốc. Nếu muốn tạo một ThemeSource implementation hoặc cần cấu hình basename prefix của ResourceBundleThemeSource, bạn có thể đăng kí một bean trong application context với tên là "themeSource". Web application context sẽ tự động nhận biết bean và khởi đông nó.

Ví dụ:

```
styleSheet=/themes/cool/style.css
background=/themes/cool/img/coolBg.jpg

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
  <head>
    <link rel="stylesheet" href="<spring:theme code="styleSheet"/>" type="text/css"/>
  </head>
  <body background="<spring:theme code="background"/>">
    ...
  </body>
</html>
```

Mặc định ResourceBundleThemeSource sử dụng một basename prefix rỗng. Các file properties sẽ được tải từ gốc của classpath, vì thế chúng ta phải nhập vào cool.properties theme của chúng ta được định nghĩa trong một thư mục tại gốc của classpath, e.g. trong /WEB-INF/classes. Chú ý rằng ResourceBundleThemeSource sử dụng cơ chế tải của chuẩn Java resource bundle, cho phép internationalization trên toàn bộ các theme.

13.7.3. Theme resolvers

Chúng ta đã có theme và bây giờ việc còn lại là chúng ta sẽ quyết định sử dụng theme nào. The DispatcherServlet sẽ tìm kiếm tên bean "themeResolver" để xác định ThemeResolver. ThemeResolver làm việc tương tự như là LocaleResolver. Nó có thể dò ra theme nào sẽ được sử dụng cho request đặc thù và có thể chỉnh sửa theme của request. Những theme resolver sau được cung cấp bởi Spring:

Table 13.7. ThemeResolver implementations

Class	Description
FixedThemeResolver	Sử dụng một theme cố định, chọn thuộc tính "defaultThemeName".
SessionThemeResolver	Theme được duy trì trong HTTP session. Nó chỉ cần được đặt trong mỗi session, nhưng không bền vững giữa các session.
CookieThemeResolver	Theme được chọn được chứa trong cookie trên máy user.

Spring cũng cung cấp ThemeChangeInterceptor, nó cho phép thay đổi thêm trên mỗi request bằng cách chèn vào một tham số request cơ bản.

13.8. Multipart support của Spring (fileupload)

13.8.1. Introduction

Spring đã xây dựng multipart support để xử lí fileupload trong ứng dụng web. MultipartResolver được định nghĩa trong gói org.springframework.web.multipart. Spring cung cấp MultipartResolvers cho việc sử dụng cùng với *Commons FileUpload* (http://jakarta.apache.org/commons/fileupload) và *COS FileUpload* (http://www.servlets.com/cos). Làm thế nào uppload files sẽ được mô tả ngay bên dưới

13.8.2. Using the MultipartResolver

Ví dụ với CommonsMultipartResolver:

```
<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

    <!-- one of the properties available; the maximum file size in bytes -->
    <property name="maxUploadSize" value="100000"/>
</bean>
```

This is an example using the CosMultipartResolver:

```
<bean id="multipartResolver" class="org.springframework.web.multipart.cos.CosMultipartResolver">

    <!-- one of the properties available; the maximum file size in bytes -->
    <property name="maxUploadSize" value="100000"/>
</bean>
```

Trong trường hợp CommonsMultipartResolver, bạn cần sự dụng commons-fileupload.jar; trong trường hợp của CosMultipartResolver, sử dụng cos.jar.

13.8.3. Handling a file upload in a form

Sau khi MultipartResolver đã hoàn thành công việc của nó, request sẽ được xử lí như những request khác.

```
<html>
   <head>
      <title>Upload a file please</title>
   </head>
   <body>
      <h1>Please upload a file</h1>
      <form method="post" action="upload.form" enctype="multipart/form-data">
         <input type="file" name="file"/>
         <input type="submit"/>
      </form>
   </body>
</html>
```

enctype="multipart/form-data là cần thiết để browser biết cách mã các trường của multipart.

Những thuộc tính khác sẽ không tự động chuyển qua string hoặc kiểu dữ liệu cơ bản (primitive type), để có thể đặt dữ liệu binary vào trong các đối tượng của bạn object bạn phải đăng kí một Custom editor với ServletRequestDatabinder. Chúng ta có hai editor cho việc xử lí file là StringMultipartEditor (có thể chuyển các file thành String) và ByteArrayMultipartEditor (chuyển các file thành dãy các byte).

So, to be able to upload files using a (HTML) form, declare the resolver, a url mapping to a controller that will process the bean, and the controller itself.

```
<beans>
                <!-- lets use the Commons-based implementation of the MultipartResolver interface -->
   <bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>

   <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
      <property name="mappings">
         <value>
            /upload.form=fileUploadController
         </value>
      </property>
   </bean>

   <bean id="fileUploadController" class="examples.FileUploadController">
      <property name="commandClass" value="examples.FileUploadBean"/>
      <property name="formView" value="fileuploadform"/>
      <property name="successView" value="confirmation"/>
   </bean>

</beans>
```

After that, create the controller and the actual class to hold the file property.

```
public class FileUploadController extends SimpleFormController {

   protected ModelAndView onSubmit(
      HttpServletRequest request,
      HttpServletResponse response,
      Object command,
      BindException errors) throws ServletException, IOException {

       // cast the bean
      FileUploadBean bean = (FileUploadBean) command;

       let's see if there's content there
      byte[] file = bean.getFile();
      if (file == null) {
          // hmm, that's strange, the user did not upload anything
      }
```

```
        // well, let's do nothing with the bean for now and return
        return super.onSubmit(request, response, command, errors);
    }

    protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
        throws ServletException {
        // to actually be able to convert Multipart instance to byte[]
        // we have to register a custom editor
        binder.registerCustomEditor(byte[].class, new ByteArrayMultipartFileEditor());
        // now Spring knows how to handle multipart object and convert them
    }

}

public class FileUploadBean {

    private byte[] file;

    public void setFile(byte[] file) {
        this.file = file;
    }

    public byte[] getFile() {
        return file;
    }
}
```

FileUploadBean có một kiểu của thuộc tính là byte[], nó nắm giữ file. Controller đăng kí một custom editor để Spring biết lầm thể nào để chuyển đổi các đối tượng multipart, resolver đã tìm thấy các thuộc tính được đặc tả bởi bean. Trong ví dụ này, không có gì được làm với thuộc tính byte[] của bean, nhưng trong bài thực hành này bạn có thể làm bất cứ cái gì bạn muốn (lưu nó vào cơ sở dữ liệu, gửi nó bằng mail đến ai đó, v.v…).

Một ví dụ tương đương, một file được giới hạn với thuộc tính String-typed trên một (form backing) đối tượng:

```
public class FileUploadController extends SimpleFormController {

    protected ModelAndView onSubmit(
        HttpServletRequest request,
        HttpServletResponse response,
        Object command,
        BindException errors) throws ServletException, IOException {

        // cast the bean
        FileUploadBean bean = (FileUploadBean) command;

        let's see if there's content there
        String file = bean.getFile();
        if (file == null) {
            // hmm, that's strange, the user did not upload anything
        }

        // well, let's do nothing with the bean for now and return
        return super.onSubmit(request, response, command, errors);
    }

    protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder)
        throws ServletException {
        // to actually be able to convert Multipart instance to a String
        // we have to register a custom editor
        binder.registerCustomEditor(String.class, new StringMultipartFileEditor());
        // now Spring knows how to handle multipart object and convert them
    }

}

public class FileUploadBean {
```

```
    private String file;

    public void setFile(String file) {
        this.file = file;
    }

    public String getFile() {
        return file;
    }
}
```

Ví dụ cuối là upload một file text, trong trường hợp này không cần đăng kí một custom editor

```
public class FileUploadController extends SimpleFormController {


    protected ModelAndView onSubmit(
        HttpServletRequest request,
        HttpServletResponse response,
        Object command,
        BindException errors) throws ServletException, IOException {

        // cast the bean
        FileUploadBean bean = (FileUploadBean) command;

        let's see if there's content there
        MultipartFile file = bean.getFile();
        if (file == null) {
            // hmm, that's strange, the user did not upload anything
        }

        // well, let's do nothing with the bean for now and return
        return super.onSubmit(request, response, command, errors);
    }
}

public class FileUploadBean {

    private MultipartFile file;

    public void setFile(MultipartFile file) {
        this.file = file;
    }

    public MultipartFile getFile() {
        return file;
    }
}
```

13.9. Using Spring's form tag library

Spring cung cấp một bộ các tag toàn diện cho việc xử lí form khi sử dụng JSP và Spring Web MVC. Mỗi tag cung cấp sự hỗ trở cho một bộ các thuộc tính tương đương trong HTML tag , làm cho các tags thân thiện và trực quan hơn. Tag-generated HTML là HTML 4.01/XHTML 1.0 compliant.

Không giống như các thư viện form/input tag khác, thư viện Spring form tag được tích hợp với Spring Web MVC, Cho các tag truy cập đến các đối tượng command và những dữ liệu liên quan mà controller của bạn cần. Ví dụ, form tags làm cho các JSP dễ dàng để phát triển, đọc và bảo trì.

Chúng ta đã included các generated HTML nơi mà các tag yêu cầu được chú giải.

13.9.1. Configuration

Thư viện form tag được đóng gói trong spring.jar. Library descriptor được gọi là spring-form.tld.

Để sử dụng thư viện các tag phải thêm sử chỉ dẫn ở đầu trang JSP của bạn:

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

13.9.2. form tag

```
<form:form>
   <table>
      <tr>
         <td>First Name:</td>
         <td><form:input path="firstName" /></td>
      </tr>
      <tr>
         <td>Last Name:</td>
         <td><form:input path="lastName" /></td>
      </tr>
      <tr>
         <td colspan="3">
            <input type="submit" value="Save Changes" />
         </td>
      </tr>
   </table>
</form:form>
```

Giá trị firstName và lastName được lấy từ đối tượng command ở PageContext của page controller.

Form HTML chuẩn:

```
<form method="POST">
   <table>
     <tr>
        <td>First Name:</td>
        <td><input name="firstName" type="text" value="Harry"/></td>
        <td></td>
     </tr>
     <tr>
        <td>Last Name:</td>
        <td><input name="lastName" type="text" value="Potter"/></td>
        <td></td>
     </tr>
     <tr>
        <td colspan="3">
          <input type="submit" value="Save Changes" />
        </td>
     </tr>
   </table>
</form>
```

```
<form:form commandName="user">
   <table>
      <tr>
         <td>First Name:</td>
         <td><form:input path="firstName" /></td>
      </tr>
      <tr>
         <td>Last Name:</td>
         <td><form:input path="lastName" /></td>
      </tr>
      <tr>
         <td colspan="3">
            <input type="submit" value="Save Changes" />
         </td>
      </tr>
   </table>
</form:form>
```

13.9.3. input tag

13.9.4. checkbox tag

```
public class Preferences {

    private boolean receiveNewsletter;

    private String[] interests;

    private String favouriteWord;

    public boolean isReceiveNewsletter() {
        return receiveNewsletter;
    }

    public void setReceiveNewsletter(boolean receiveNewsletter) {
        this.receiveNewsletter = receiveNewsletter;
    }

    public String[] getInterests() {
        return interests;
    }

    public void setInterests(String[] interests) {
        this.interests = interests;
    }

    public String getFavouriteWord() {
        return favouriteWord;
    }

    public void setFavouriteWord(String favouriteWord) {
        this.favouriteWord = favouriteWord;
    }
}
```

```
<form:form>
    <table>
        <tr>
            <td>Subscribe to newsletter?:</td>
            <%-- Approach 1: Property is of type java.lang.Boolean --%>
            <td><form:checkbox path="preferences.receiveNewsletter"/></td>
            <td> </td>
        </tr>

        <tr>
            <td>Interests:</td>
            <td>
                <%-- Approach 2: Property is of an array or of type java.util.Collection --%>
                Quidditch: <form:checkbox path="preferences.interests" value="Quidditch"/>
                Herbology: <form:checkbox path="preferences.interests" value="Herbology"/>
                Defence Against the Dark Arts: <form:checkbox path="preferences.interests"
                    value="Defence Against the Dark Arts"/>
            </td>
            <td> </td>
        </tr>
        <tr>
            <td>Favourite Word:</td>
            <td>
                <%-- Approach 3: Property is of type java.lang.Object --%>
                Magic: <form:checkbox path="preferences.favouriteWord" value="Magic"/>
            </td>
            <td> </td>
        </tr>
    </table>
</form:form>
```

```
<tr>
   <td>Interests:</td>
   <td>
      Quidditch: <input name="preferences.interests" type="checkbox" value="Quidditch"/>
      <input type="hidden" value="1" name="_preferences.interests"/>
      Herbology: <input name="preferences.interests" type="checkbox" value="Herbology"/>
      <input type="hidden" value="1" name="_preferences.interests"/>
      Defence Against the Dark Arts: <input name="preferences.interests" type="checkbox"
         value="Defence Against the Dark Arts"/>
      <input type="hidden" value="1" name="_preferences.interests"/>
   </td>
   <td> </td>
</tr>
```

13.9.5. radiobutton tag

```
<tr>
   <td>Sex:</td>
   <td>Male: <form:radiobutton path="sex" value="M"/> <br/>
      Female: <form:radiobutton path="sex" value="F"/> </td>
   <td> </td>
</tr>
```

13.9.6. password tag

```
<tr>
   <td>Password:</td>
   <td>
      <form:password path="password" />
   </td>
</tr>


<tr>
   <td>Password:</td>
   <td>
      <form:password path="password" value="^76525bvHGq" showPassword="true" />
   </td>
</tr>
```

13.9.7. select tag

```
<tr>
   <td>Skills:</td>
   <td><form:select path="skills" items="${skills}"/></td>
   <td></td>
</tr>


<tr>
   <td>Skills:</td>
   <td><select name="skills" multiple="true">
      <option value="Potions">Potions</option>
      <option value="Herbology" selected="true">Herbology</option>
      <option value="Quidditch">Quidditch</option></select></td>
   <td></td>
</tr>
```

13.9.8. option tag

```
<tr>
   <td>House:</td>
   <td>
      <form:select path="house">
         <form:option value="Gryffindor"/>
         <form:option value="Hufflepuff"/>
         <form:option value="Ravenclaw"/>
         <form:option value="Slytherin"/>
      </form:select>
   </td>
</tr>
```

```
<tr>
   <td>House:</td>
   <td>
      <select name="house">
         <option value="Gryffindor" selected="true">Gryffindor</option>
         <option value="Hufflepuff">Hufflepuff</option>
         <option value="Ravenclaw">Ravenclaw</option>
         <option value="Slytherin">Slytherin</option>
      </select>
   </td>
 </tr>
```

13.9.9. options tag

```
<tr>
   <td>Country:</td>
   <td>
      <form:select path="country">
         <form:option value="-" label="--Please Select"/>
         <form:options items="${countryList}" itemValue="code" itemLabel="name"/>
      </form:select>
   </td>
   <td></td>
</tr>
```

If the User lived in the UK, the HTML source of the 'Country' row would look like:

```
 <tr>
   <td>Country:</td>
   <tr>
      <td>Country:</td>
      <td>
         <select name="country">
            <option value="-">--Please Select</option>
            <option value="AT">Austria</option>
            <option value="UK" selected="true">United Kingdom</option>
            <option value="US">United States</option>
         </select>
      </td>
      <td></td>
   </tr>
   <td></td>
</tr>
```

As the example shows, the combined usage of an option tag with the options tag generates the same standard HTML, but allows you to explicitly specify a value in the JSP that is for display only (where it belongs) such as the default string in the example: "--Please Select".

13.9.10. The textarea tag

This tag renders an HTML 'textarea'.

```
<tr>
   <td>Notes:</td>
   <td><form:textarea path="notes" rows="3" cols="20" /></td>
   <td><form:errors path="notes" /></td>
</tr>
```

13.9.11. The hidden tag

```
<form:hidden path="house" />

<input name="house" type="hidden" value="Gryffindor"/>
```

13.9.12. errors tag

```
public class UserValidator implements Validator {

   public boolean supports(Class candidate) {
      return User.class.isAssignableFrom(candidate);
   }

   public void validate(Object obj, Errors errors) {
      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "firstName", "required", "Field is required.");
      ValidationUtils.rejectIfEmptyOrWhitespace(errors, "lastName", "required", "Field is required.");
   }
}
```

The form.jsp would look like:

```
<form:form>
   <table>
      <tr>
         <td>First Name:</td>
         <td><form:input path="firstName" /></td>
         <%-- Show errors for firstName field --%>
         <td><form:errors path="firstName" /></td>
      </tr>

      <tr>
         <td>Last Name:</td>
         <td><form:input path="lastName" /></td>
         <%-- Show errors for lastName field --%>
         <td><form:errors path="lastName"  /></td>
      </tr>
      <tr>
         <td colspan="3">
            <input type="submit" value="Save Changes" />
         </td>
      </tr>
   </table>
</form:form>
```

If we submit a form with empty values in the firstHame and lastName fields, this is what the HTML would look like:

```
<form method="POST">
   <table>
      <tr>
         <td>First Name:</td>
         <td><input name="firstName" type="text" value=""/></td>
         <%-- Associated errors to firstName field displayed --%>
         <td><span name="firstName.errors">Field is required.</span></td>
      </tr>

      <tr>
         <td>Last Name:</td>
         <td><input name="lastName" type="text" value=""/></td>
         <%-- Associated errors to lastName field displayed --%>
         <td><span name="lastName.errors">Field is required.</span></td>
      </tr>
      <tr>
         <td colspan="3">
            <input type="submit" value="Save Changes" />
         </td>
      </tr>
   </table>
</form>
```

What if we want to display the entire list of errors for a given page? The example below shows that the errors tag also supports some basic wildcarding functionality.

- path="*" - displays all errors
- path="lastName*" - displays all errors associated with the lastName field

The example below will display a list of errors at the top of the page, followed by field-specific errors next to the fields:

```
<form:form>
   <form:errors path="*" cssClass="errorBox" />
   <table>
      <tr>
         <td>First Name:</td>
         <td><form:input path="firstName" /></td>
         <td><form:errors path="firstName" /></td>
      </tr>
      <tr>
         <td>Last Name:</td>
         <td><form:input path="lastName" /></td>
         <td><form:errors path="lastName"  /></td>
      </tr>
      <tr>
         <td colspan="3">
            <input type="submit" value="Save Changes" />
         </td>
      </tr>
   </table>
</form:form>
```

The HTML would look like:

```
<form method="POST">
   <span name="*.errors" class="errorBox">Field is required.<br/>Field is required.</span>
   <table>
      <tr>
         <td>First Name:</td>
         <td><input name="firstName" type="text" value=""/></td>
         <td><span name="firstName.errors">Field is required.</span></td>
      </tr>

      <tr>
         <td>Last Name:</td>
         <td><input name="lastName" type="text" value=""/></td>
         <td><span name="lastName.errors">Field is required.</span></td>
      </tr>
      <tr>
         <td colspan="3">
            <input type="submit" value="Save Changes" />
         </td>
      </tr>
</form>
```