Spring Mobile Reference Manual

1.0.2.RELEASE

Keith Donald , Roy Clarkson

Table of Contents

1. Spring Mobile Overview

1.1 Introduction

Spring Mobile contains extensions to Spring MVC for developing mobile web applications. This includes a module for server-side mobile device detection.

2. Spring Mobile Device Module

2.1 Introduction

Device detection is useful when requests by mobile devices need to be handled differently from requests made by desktop browsers. The Spring Mobile Device module provides support for server-side device detection. This support consists of a device resolution framework, site preference management, and site switcher.

2.2 How to get

To get the module, add the spring-mobile-device artifact to your classpath:

```
<dependency>
    <groupId>org.springframework.mobile</groupId>
    <artifactId>spring-mobile-device</artifactId>
    <version>${org.springframework.mobile-version}</version>
</dependency>
```

If you are developing against a milestone version, such as 1.0.0.RC2, you will need to add the following repository in order to resolve the artifact:

```
<repository>
    <id>springsource-milestone</id>
    <name>SpringSource Milestone Repository</name>
    <url>http://repo.springsource.org/milestone</url>
</repository>
```

If you are testing out the latest nightly build version (e.g. 1.0.0.BUILD-SNAPSHOT), you will need to add the following repository:

```
<repository>
    <id>springsource-snapshot</id>
    <name>SpringSource Snapshot Repository</name>
    <url>http://repo.springsource.org/snapshot</url>
</repository>
```

2.3 Device resolution

Device resolution is the process of introspecting a HTTP request to determine the device that originated the request. It is typically achieved by analyzing the User-Agent header and other request headers.

At the most basic level, device resolution answers the question: "Is the client using a mobile or tablet device?". This answer enables your application to respond differently to mobile devices that have small screens, or tablet device that has a touch interface. More sophisticated device resolvers are also capable of identifying specific device capabilities, such as screen size, manufacturer, model, or preferred markup.

In Spring Mobile, the DeviceResolver interface defines the API for device resolution:

```
public interface DeviceResolver {

    Device resolveDevice(HttpServletRequest request);

}
```

The returned Device models the result of device resolution:

```
public interface Device {

    /**
     * True if this device is not a mobile or tablet device.
     */
    boolean isNormal();

    /**
     * True if this device is a mobile device such as an Apple iPhone or an Nexus One
 Android.
     * Could be used by a pre-handle interceptor to redirect the user to a dedicated
 mobile web site.
     * Could be used to apply a different page layout or stylesheet when the device is a
 mobile device.
     */
    boolean isMobile();

    /**
     * True if this device is a tablet device such as an Apple iPad or a Motorola Xoom.
     * Could be used by a pre-handle interceptor to redirect the user to a dedicated
 tablet web site.
     * Could be used to apply a different page layout or stylesheet when the device is a
 tablet device.
     */
    boolean isTablet();

}
}
```

As shown above, Device.isMobile() can be used to determine if the client is using a mobile device, such as a smart phone. Similarly, Device.isTablet() can be used to determine if the client is running on a tablet device. Depending on the DeviceResolver in use, a Device may support being downcast to access additional properties.

When to perform

Web applications should perform device resolution at the beginning of request processing, before any request handler is invoked. This ensures the Device model can be made available in request scope before any processing occurs. Request handlers can then obtain the Device instance and use it to respond differently based on its state.

By default, a LiteDeviceResolver is used for device resolution. You may plug-in another DeviceResolver implementation by injecting a constructor argument.

DeviceResolverHandlerInterceptor

Spring Mobile ships a HandlerInterceptor that, on preHandle, delegates to a DeviceResolver. The resolved Device is indexed under a request attribute named 'currentDevice', making it available to handlers throughout request processing.

To enable, add the DeviceResolverHandlerInterceptor to the list of interceptors defined in your DispatcherServlet configuration:

```
<mvc:interceptors>
    <!-- On pre-handle, resolve the device that originated the web request -->
    <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor"
 />
</mvc:interceptors>
```

DeviceResolverRequestFilter

Spring Mobile also ships with a Servlet Filter that delegates to the same DeviceResolver. As with the HandlerInterceptor, the resolved Device is indexed under a request attribute named 'currentDevice', making it available to handlers throughout request processing.

To enable, add the DeviceResolverRequestFilter to your web.xml:

```
<filter>
    <filter-name>deviceResolverRequestFilter</filter-name>
    <filter-class>org.springframework.mobile.device.DeviceResolverRequestFilter</filter-
class>
</filter>
```

Obtaining a reference to the current device

When you need to lookup the current Device in your code, you can do so in several ways. If you already have a reference to a ServletRequest or Spring WebRequest, simply use DeviceUtils:

```
Device currentDevice = DeviceUtils.getCurrentDevice(servletRequest);
```

If you'd like to pass the current Device as an argument to one of your @Controller methods, configure a DeviceWebArgumentResolver:

```
<mvc:annotation-driven>
    <mvc:argument-resolvers>
        <beans:bean class="org.springframework.mobile.device.DeviceWebArgumentResolver" />
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

You can then inject the Device into your @Controllers as shown below:

```
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    @RequestMapping("/")
    public void home(Device device) {
        if (device.isMobile()) {
            logger.info("Hello mobile user!");
        } else {
            logger.info("Hello desktop user!");
        }
    }
```

Supported DeviceResolver implementations

Spring Mobile allows for the development of different DeviceResolver implementations that offer varying levels of resolution capability. The first, and the default, is a *LiteDeviceResolver* that detects the presence of a mobile device but does not detect specific capabilities.

LiteDeviceResolver

The default DeviceResolver implementation is based on the "lite" detection algorithm implemented as part of the Wordpress Mobile Pack. This resolver only detects the presence of a mobile or tablet device, and does not detect specific capabilities. No special configuration is required to enable this resolver, simply configure a default DeviceResolverHandlerInterceptor and it will be enabled for you.

It is possible that the LiteDeviceResolver incorrectly identifies a User-Agent as a mobile device. The LiteDeviceResolver provides a configuration option for setting a list of User-Agent keywords that should resolve to a "normal" device, effectively overriding the default behavior. These keywords take precedence over the mobile and tablet device detection keywords. The following example illustrates how to set the normal keywords in the configuration of the DeviceResolverHandlerInterceptor by injecting a constructor argument. In this case, User-Agents that contain "iphone" and "android" would no longer resolve to a mobile device.

```
<mvc:interceptors>
        <!-- Detects the client's Device -->

  <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor">
            <beans:constructor-arg>
                <beans:bean class="org.springframework.mobile.device.LiteDeviceResolver">
                    <beans:constructor-arg>
                        <beans:list>
                            <beans:value>iphone</beans:value>
                            <beans:value>android</beans:value>
                        </beans:list>
                    </beans:constructor-arg>
                </beans:bean>
            </beans:constructor-arg>
        </beans:bean>
</mvc:interceptors>
```

Alternatively, you may subclass the LiteDeviceResolver, and either set these values in the constructor, or by calling the getNormalUserAgentKeywords() method.

2.4 Site preference management

Device resolution is often used to determine which "site" will be served to the user. For example, a mobile user may be served a "mobile site" that contains content optimized for display on a small screen, while a desktop user would be served the "normal site". Support for multiple sites can be achieved by introspecting Device.isMobile() and varying controller and view rendering logic based on its value.

However, when an application supports multiple sites, allowing the user to switch between them, if desired, is considered a good usability practice. For example, a mobile user currently viewing the mobile site may wish to access the normal site instead, perhaps because some content he or she would like to access is not available through the mobile UI.

Building on the device resolution system is a facility for this kind of "user site preference management". This facility allows the user to indicate if he or she prefers the mobile site or the normal site. The indicated SitePreference may then be used to vary control and view rendering logic.

The SitePreferenceHandler interface defines the core service API for site preference management:

```
public interface SitePreferenceHandler {

    /**
     * The name of the request attribute that holds the current user's site preference
 value.
     */
    final String CURRENT_SITE_PREFERENCE_ATTRIBUTE = "currentSitePreference";

    /**
     * Handle the site preference aspect of the web request.
     * Implementations should first check if the user has indicated a site preference.
     * If so, the indicated site preference should be saved and remembered for future
 requests.
     * If no site preference has been indicated, an implementation may derive a default
 site preference from the {@link Device} that originated the request.
     * After handling, the user's site preference is returned and also available as a
 request attribute named 'currentSitePreference'.
     */
    SitePreference handleSitePreference(HttpServletRequest request, HttpServletResponse
 response);

}
```

The resolved SitePreference is an enum value:

```
public enum SitePreference {

    /**
     * The user prefers the 'normal' site.
     */
    NORMAL,

    /**
     * The user prefers the 'mobile' site.
     */
    MOBILE {
        public boolean isMobile() {
            return true;
        }
    };

    /**
     * Tests if this is the 'mobile' SitePreference.
     * Designed to support concise SitePreference boolean expressions e.g. <c:if
 test="${currentSitePreference.mobile}"></c:if>.
     */
    public boolean isMobile() {
        return false;
    }

}
```

Spring Mobile provides a single SitePreferenceHandler implementation named StandardSitePreferenceHandler, which should be suitable for most needs. It supports query-parameter-based site preference indication, pluggable SitePreference storage, and may be enabled in a Spring MVC application using a HandlerIntercepor. In addition, if no SitePreference has been explcitly indicated by the user, a default will be derived based on the user's Device (MOBILE for mobile devices, and NORMAL otherwise).

Indicating a site preference

The user may indicate a site preference by activating a link that submits the site_preference query parameter:

```
Site: <a href="${currentUrl}?site_preference=normal">Normal</a> | <a href="${currentUrl}?
site_preference=mobile">Mobile</a>
```

The indicated site preference is saved for the user in a SitePreferenceRepository, and made available as a request attribute named 'currentSitePreference'.

Site preference storage

Indicated site preferences are stored in a SitePreferenceRepository so they are remembered in future requests made by the user. CookieSitePreferenceRepository is the default implementation and stores the user's' preference in a client-side cookie.

Enabling site preference management

To enable SitePreference management before requests are processed, add the SitePreferenceHandlerInterceptor to your DispatcherServlet configuration:

```
<mvc:interceptors>
    <!-- On pre-handle, resolve the device that originated the web request -->
    <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor"
 />
    <!-- On pre-handle, manage the user's site preference (declare after
 DeviceResolverHandlerInterceptor) -->

  <beans:bean class="org.springframework.mobile.device.site.SitePreferenceHandlerInterceptor"
 />
</mvc:interceptors>
```

By default, the interceptor will delegate to a StandardSitePreferenceHandler configured with a CookieSitePreferenceRepository. You may plug-in another SitePreferenceHandler by injecting a constructor argument. After the interceptor is invoked, the SitePreference will be available as a request attribute named 'currentSitePreference'.

Obtaining a reference to the current site preference

When you need to lookup the current SitePreference in your code, you can do so in several ways. If you already have a reference to a ServletRequest or Spring WebRequest, simply use SitePreferenceUtils:

```
SitePreference sitePreference =
 SitePreferenceUtils.getCurrentSitePreference(servletRequest);
```

If you'd like to pass the current SitePreference as an argument to one of your @Controller methods, configure a SitePreferenceWebArgumentResolver:

```
<mvc:annotation-driven>
    <mvc:argument-resolvers>
        <beans:bean class="org.springframework.mobile.device.DeviceWebArgumentResolver" />

 <beans:bean class="org.springframework.mobile.device.site.SitePreferenceWebArgumentResolver"
 />
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

You can then inject the indicated SitePreference into your @Controllers as shown below:

```
@Controller
public class HomeController {

    @RequestMapping("/")
    public String home(SitePreference sitePreference, Model model) {
        if (sitePreference == SitePreference.MOBILE) {
            // prepare mobile view for rendering
            return "home-mobile";
        } else {
            // prepare normal view for rendering
            return "home";
        }
    }

}
```

2.5 Site switching

Some applications may wish to host their "mobile site" at a different domain from their "normal site". For example, Google will switch you to m.google.com if you access google.com from your mobile phone.

In Spring Mobile, you may use the SiteSwitcherHandlerInterceptor to redirect mobile users to a dedicated mobile site. Users may also indicate a site preference; for example, a mobile user may still wish to use 'normal' site. Convenient static factory methods are provided that implement standard site switching conventions.

mDot SiteSwitcher

Use the "mDot" factory method to construct a SiteSwitcher that redirects mobile users to m.${serverName}; for example, m.myapp.com:

```
<mvc:interceptors>
    <!-- On pre-handle, resolve the device that originated the web request -->
    <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor"
 />
    <!-- On pre-handle, redirects mobile users to "m.myapp.com" (declare after
 DeviceResolverHandlerInterceptor) -->

  <beans:bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor" factory-
method="mDot">
        <beans:constructor-arg value="myapp.com" />
    </beans:bean>
</mvc:interceptors>
```

dotMobi SiteSwitcher

Use the "dotMobi" factory method to construct a SiteSwitcher that redirects mobile users to ${serverName - lastDomain}.mobi; for example, myapp.mobi:

```
<mvc:interceptors>
    <!-- On pre-handle, resolve the device that originated the web request -->
    <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor"
 />
    <!-- On pre-handle, redirects mobile users to "myapp.mobi" (declare after
 DeviceResolverHandlerInterceptor) -->

  <beans:bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor" factory-
method="dotMobi">
        <beans:constructor-arg value="myapp.com" />
    </beans:bean>
</mvc:interceptors>
```

urlPath SiteSwitcher

Use the "urlPath" factory method to construct a SiteSwitcher that redirects mobile users to ${serverName}/${mobilePath}; for example, myapp.com/m/:

```
<mvc:interceptors>
    <!-- On pre-handle, resolve the device that originated the web request -->
    <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor"
 />
    <!-- On pre-handle, redirects mobile users to "myapp.com/m" (declare after
 DeviceResolverHandlerInterceptor) -->

  <beans:bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor" factory-
method="urlPath">
        <beans:constructor-arg value="/m" />
    </beans:bean>
</mvc:interceptors>
```

You can also specify the root path of the application in the "urlPath" factory method. The following sample constructs a SiteSwitcher that redirects mobile users to ${serverName}/${rootPath}/${mobilePath}; for example, myapp.com/showcase/m/:

```
<mvc:interceptors>
    <!-- On pre-handle, resolve the device that originated the web request -->
    <beans:bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor"
 />
    <!-- On pre-handle, redirects mobile users to "myapp.com/showcase/m" (declare after
 DeviceResolverHandlerInterceptor) -->

  <beans:bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor" factory-
method="urlPath">
        <beans:constructor-arg value="/m" />
        <beans:constructor-arg value="/showcase" />
    </beans:bean>
</mvc:interceptors>
```

The "mDot", "dotMobi" and "urlPath" factory methods configure cookie-based SitePreference storage. The cookie value will be shared across the mobile and normal site domains. Internally,

the interceptor delegates to a SitePreferenceHandler, so there is no need to register a SitePreferenceHandlerInterceptor when using the switcher.

See the JavaDoc of SiteSwitcherHandlerInterceptor for additional options when you need more control. See the spring-mobile samples repository for runnable SiteSwitcher examples.