Spring Mobile Reference Manual

Keith Donald

Spring Mobile Reference Manual

by Keith Donald

1.0.0.M2

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.

2.2 How to get

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>
```

2.3 DeviceResolverHandlerInterceptor

The DeviceResolverHandlerInterceptor detects the device that originated the web request before handler invocation. The detected Device is set as a request attribute named 'currentDevice' and made available to handlers during request processing. This allows handlers to vary their controller and/or presentation logic by device type.

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

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

By default, the interceptor will use a LiteDeviceResolver for device detection. You may plug-in your own DeviceResolver by injecting a constructor argument. See Section 2.5, “Device resolvers” for more information on the implementation options.

2.4 DeviceWebArgumentResolver

The DeviceWebArgumentResolver allows you to inject the detected Device into @Controller handler methods. To use, register this resolver with your DispatcherServlet's AnnotationMethodHandlerAdapter infastructure bean:

```
@Component
public class CustomWebArgumentResolverInstaller {
```

```
    @Inject
    public CustomWebArgumentResolverInstaller(AnnotationMethodHandlerAdapter controllerInvoker) {
        WebArgumentResolver[] resolvers = new WebArgumentResolver[1];
        resolvers[0] = new DeviceWebArgumentResolver();
        controllerInvoker.setCustomArgumentResolvers(resolvers);
    }
}
```

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

```
@Controller
public class HomeController {

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

    /**
     * Declares a {@link Device} parameter to show how you can resolve the model for the device that originated
     * Argument resolution is handled by the {@link DeviceWebArgumentResolver} installed by {@link CustomWebArg
     */
    @RequestMapping("/")
    public void home(Device device) {
        if (device.isMobile()) {
            logger.info("Hello mobile user!");
        } else {
            logger.info("Hello desktop user!");
        }
    }
```

2.5 Device resolvers

The DeviceResolver interface is central the service API for device detection. Spring Mobile provides two DeviceResolver implementations: LiteDeviceResolver and WurflDeviceResolver. You may also implement your own.

LiteDeviceResolver

The default DeviceResolver implementation based on the "lite" detection algorithm [http://plugins.trac.wordpress.org/browser/wordpress-mobile-pack/trunk/plugins/wpmp_switcher/lite_detection.php] implemented as part of the Wordpress Mobile Pack [http://wordpress.org/extend/plugins/wordpress-mobile-pack]. This resolver only detects the presence of a mobile device and does not detect specific capabilities.

WurflDeviceResolver

A DeviceResolver implementation that delegates to WURFL for device detection. WURFL provides a large database of devices and their capabilities. It is useful when you need to know more about the Device that originated the request, such as its specific screen size, manufacturer, model, preferred markup, or other capabilities.

To use, first make sure WURFL is in your classpath:

```
<dependency>
    <groupId>net.sourceforge.wurfl</groupId>
    <artifactId>wurfl</artifactId>
    <version>${net.sourceforge.wurfl-version}</version>
</dependency>
```

To enable the resolver, include the spring-mobile-device XML namespace and use the wurfl-device-resolver tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:device="http://www.springframework.org/schema/mobile/device"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spr
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.
        http://www.springframework.org/schema/mobile/device http://www.springframework.org/schema/mobile/device

    <interceptors>
        <!-- On pre-handle, use WURFL to detect the device that originated the web request -->
        <beans:bean class="org.springframework.mobile.device.mvc.DeviceResolverHandlerInterceptor">
            <beans:constructor-arg>
                <device:wurfl-device-resolver root-location="/WEB-INF/wurfl/wurfl-2.0.25.zip" patch-locations="
            </beans:constructor-arg>
        </beans:bean>
    </interceptors>

</beans:beans>
```

Read more about WURFL at http://wurfl.sourceforge.net. Checkout the wurfl-showcase [http://git.springsource.org/spring-mobile/samples] for a live demonstration. This sample illustrates WURFL-based device detection as well as the use of the WNG tag library to perform multi-serving from common page markup.

If you implement your own DeviceResolver, please consider contributing your implementation back to the community at http://www.springsource.org/spring-mobile.

2.6 SiteSwitcherHandlerInterceptor

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:

```
<interceptors>

    <!-- Redirects mobile users to "m.myapp.com".  The order of this interceptor is significant (it should be c
```

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

</interceptors>
```

dotMobi SiteSwitcher

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

```
<interceptors>

    <!-- Redirects mobile users to "myapp.mobi".  The order of this interceptor is significant (it should be de
    <beans:bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor" factory-metho
        <beans:constructor-arg value="myapp.com" />
    </beans:bean>

</interceptors>
```

Indicating a site preference

The user may indicate a site preference by submitting the site_preference query parameter:

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

The indicated site preference is saved for the user in a SitePreferenceRepository, and also made available as a request attribute named 'currentSitePreference'. The default SitePreferenceRepository implementation used by the mDot and dotMobi SiteSwitcher factory methods is cookie-based. The cookie value is shared across the normal and mobile site domains.

SitePreferenceWebArgumentResolver

The SitePreferenceWebArgumentResolver allows you to inject the user's indicated SitePreference into @Controller handler methods. To use, add this resolver to the list of custom WebArgumentResolvers registered with the DispatcherServlet's AnnotationMethodHandlerAdapter infastructure bean:

```
@Component
public class CustomWebArgumentResolverInstaller {

    @Inject
    public CustomWebArgumentResolverInstaller(AnnotationMethodHandlerAdapter controllerInvoker) {
        WebArgumentResolver[] resolvers = new WebArgumentResolver[2]
        resolvers[0] = new DeviceWebArgumentResolver();
        resolvers[1] = new SitePreferenceArgumentResolver();
        controllerInvoker.setCustomArgumentResolvers(resolvers);
    }

}
```

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";
        }
    }

}
```

See the spring-mobile samples [http://git.springsource.org/spring-mobile/samples] repository for runnable SiteSwitcher examples.