Spring Android Reference Manual

Roy Clarkson

Spring Android Reference Manual

by Roy Clarkson

1.0.0.M2

Table of Contents

1. Spring Android Overview

1.1 Introduction

The Spring Android project supports the usage of the Spring Framework in an Android environment. This includes the ability to use RestTemplate as the REST client for your Android applications.

2. Spring Android Rest Template Module

2.1 Introduction

Spring's RestTemplate is a robust, popular Java-based REST client. The Spring Android Rest Template Module provides a version of RestTemplate that works in an Android environment.

2.2 Overview

The RestTemplate object is the heart of the Spring Android Rest Template library. When you create a new RestTemplate instance, the constructor sets up several supporting objects that make up the RestTemplate functionality.

HttpComponents HttpClient 4.x

The HttpComponents HttpClient [http://hc.apache.org/httpcomponents-client-ga/index.html] is the native http client available on the Android platform. Within Spring Android Rest Template this HttpClient is made available through the HttpComponentsClientHttpRequestFactory. The HttpComponentsClientHttpRequestFactory is set as the default RequestFactory when you create a new RestTemplate instance, so you are not required to set it manually.

Please note that the M1 release of Spring Android included Commons HttpClient 3.x support. This support is now deprecated in M2, and will not be available in the next milestone release. Because the HttpComponents 4.x client is native to Android and available without an additional dependency, it should be used instead.

Object to JSON Marshaling

Object to JSON marshaling in Spring Android Rest Template requires the use of a third party JSON mapping library. The Jackson JSON Processor [http://jackson.codehaus.org] is used within the MappingJacksonHttpMessageConverter to provide this marshaling functionality. The media type supported by this message converter is "application/json".

The MappingJacksonHttpMessageConverter is conditionally loaded when you create a new RestTemplate instance. If the Jackson dependencies are found in your classpath, the message converter will be automatically added and available for use in REST operations. See the How to Get section for more details on including Jackson in your project. Additionally, the Usage Examples section provides code samples.

Object to XML Marshaling

Object to XML marshaling in Spring Android Rest Template requires the use of a third party XML mapping library. The Simple XML serializer [http://simple.sourceforge.net] is used within the SimpleXmlHttpMessageConverter to provide this marshaling functionality. The media types supported by this message converter are "application/xml", "text/xml", and "application/*+xml".

The SimpleXmlHttpMessageConverter is conditionally loaded when you create a new RestTemplate instance. If the Simple dependency is found in your classpath, the message converter will be automatically added and

available for use in REST operations. See the How to Get section for more details on including Simple in your project. Additionally, the Usage Examples section provides code samples.

RSS and Atom Support

RSS and Atom feed support in Spring Android Rest Template requires the use of a third party feed reader library. The Android ROME Feed Reader [http://code.google.com/p/android-rome-feed-reader] is used within the SyndFeedHttpMessageConverter, RssChannelHttpMessageConverter, and the AtomFeedHttpMessageConverter to provide this functionality. The media types supported by these message converters are "application/rss+xml" and "application/atom+xml".

The SyndFeedHttpMessageConverter is conditionally loaded when you create a new RestTemplate instance. If the Android ROME dependencies are found in your classpath, the message converter will be automatically added and available for use in REST operations. See the How to Get section for more details on including Android ROME in your project. Additionally, the Usage Examples section provides code samples.

Because the SyndFeedHttpMessageConverter provides a higher level abstraction around RSS and Atom feeds, the RssChannelHttpMessageConverter, and AtomFeedHttpMessageConverter are not automatically added when you create a new RestTemplate instance. If you prefer to use one of these message converters then you have to manually add it to the RestTemplate instance.

2.3 How to get

Add the spring-android-rest-template artifact to your classpath:

```
<dependency>
    <groupId>org.springframework.android</groupId>
    <artifactId>spring-android-rest-template</artifactId>
    <version>1.0.0.BUILD-SNAPSHOT</version>
</dependency>
```

Unfortunately, Google's provided Android toolset does not include dependency management support. However, through the use of third party tools, you can use Maven to manage dependencies and build your Android app. See the Spring Android and Maven section for more information.

Spring Android Rest Template supports several optional libraries. These optional libraries are used by different Http Message Converters within Rest Template. If you would like to make use of these Message Converters, then you need to include the corresponding libraries in your classpath.

Jackson JSON Processor

The MappingJacksonHttpMessageConverter is used to marshal Objects to JSON. The Jackson [http://jackson.codehaus.org] library provides this functionality.

Add the following Jackson dependencies to your classpath to enable this Message Converter.

```
<dependency>
```

```
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.7.1</version>
</dependency>
```

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-core-asl</artifactId>
    <version>1.7.1</version>
</dependency>
```

Simple XML Serializer

The SimpleXmlHttpMessageConverter is used to marshal Objects to XML. Simple [http://simple.sourceforge.net] is an XML serialization and configuration framework for Java that is compatible with Android.

Add the following Simple dependency to your classpath to enable this Message Converter.

```
<dependency>
    <groupId>org.simpleframework</groupId>
    <artifactId>simple-xml</artifactId>
    <version>2.4.1</version>
</dependency>
```

Android ROME Feed Reader

The RssChannelHttpMessageConverter, AtomFeedHttpMessageConverter, and SyndFeedHttpMessageConverter are used to process RSS and Atom feeds. Android ROME Feed Reader [http://code.google.com/p/android-rome-feed-reader] is a port of the popular ROME library that is compatible with Android.

Add the following Android ROME dependencies to your classpath to enable these Message Converters. This library depends on a forked version of JDOM to work on Android 2.1 and earlier. The JDOM library addresses a bug [http://www.jdom.org/pipermail/jdom-interest/2009-July/016345.html] in the Android XML parser.

```
<dependency>
    <groupId>com.google.code.android-rome-feed-reader</groupId>
    <artifactId>android-rome-feed-reader</artifactId>
    <version>1.0.0-r2</version>
</dependency>
```

```
<dependency>
    <groupId>org.jdom</groupId>
    <artifactId>jdom</artifactId>
    <version>1.1.1-android-fork</version>
</dependency>
```

2.4 Usage Examples

Using Rest Template, it's easy to invoke RESTful APIs. Below are several usage examples that illustrate the different methods for making RESTful requests.

All of the following examples are based on a sample Android application [http://git.springsource.org/spring-mobile/samples]. You can retrieve the source code for the sample app with the following command:

```
$ git clone git://git.springsource.org/spring-mobile/samples.git
```

Basic Usage Example

The following example shows a query to google for the search term "SpringSource".

```
RestTemplate restTemplate = new RestTemplate();
String url = "https://ajax.googleapis.com/ajax/services/search/web?v=1.0&q={query}";
String result = restTemplate.getForObject(url, String.class, "SpringSource");
```

Retrieving JSON data via HTTP GET

Suppose you have defined a Java object you wish to populate from a RESTful web request that returns JSON content.

Define your object based on the JSON data being returned from the RESTful request:

```
public class Event {

    private Long id;

    private String title;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public String setTitle(String title) {
        this.title = title;
    }
}
```

Make the RestTemplate request:

```
String url = "http://mypretendservice.com/events";
RestTemplate restTemplate = new RestTemplate();
Event[] events = restTemplate.getForObject(url, Event[].class);
```

You can also set the Accept header for the request:

```
List<MediaType> acceptableMediaTypes = new ArrayList<MediaType>();
acceptableMediaTypes.add(new MediaType("application","json"));

HttpHeaders requestHeaders = new HttpHeaders();
requestHeaders.setAccept(acceptableMediaTypes);

HttpEntity<?> requestEntity = new HttpEntity<Object>(requestHeaders);

String url = "http://mypretendservice.com/events";

RestTemplate restTemplate = new RestTemplate();
ResponseEntity<Event[]> responseEntity = restTemplate.exchange(url, HttpMethod.GET, requestEntity, Event[].clas
Event[] events = responseEntity.getBody();
```

Retrieving XML data via HTTP GET

Using the same Java object we defined earlier, we can modify the requests to retrieve XML.

Define your object based on the XML data being returned from the RESTful request. Note the annotations used by Simple to marshal the object:

```
@Root
public class Event {

    @Element
    private Long id;

    @Element
    private String title;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public String setTitle(String title) {
        this.title = title;
    }
}
```

To marshal an array of events from xml, we need to define a wrapper class for the list:

```
@Root(name="events")
public class EventList {

   @ElementList(inline=true)
   private List<Event> events;

   public List<Event> getEvents() {
      return events;
   }

   public void setEvents(List<Event> events) {
       this.events = events;
   }
}
```

Make the RestTemplate request:

```
String url = "http://mypretendservice.com/events";
RestTemplate restTemplate = new RestTemplate();
EventList eventList = restTemplate.getForObject(url, EventList.class);
```

You can also specify the Accept header for the request:

```
List<MediaType> acceptableMediaTypes = new ArrayList<MediaType>();
acceptableMediaTypes.add(new MediaType("application","xml"));

HttpHeaders requestHeaders = new HttpHeaders();
requestHeaders.setAccept(acceptableMediaTypes);

HttpEntity<?> requestEntity = new HttpEntity<Object>(requestHeaders);

String url = "http://mypretendservice.com/events";

RestTemplate restTemplate = new RestTemplate();
ResponseEntity<EventList> responseEntity = restTemplate.exchange(url, HttpMethod.GET, requestEntity, EventList.
EventList eventList = responseEntity.getBody();
```

Send JSON data via HTTP POST

POST a Java object you have defined to a RESTful service that accepts JSON data.

Define your object based on the JSON data expected by the RESTful request:

```
public class Message
{
    private long id;
```

```
    private String subject;

    private String text;

    public void setId(long id) {
        this.id = id;
    }

    public long getId() {
        return id;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public String getSubject() {
        return subject;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getText() {
        return text;
    }
}
```

Make the RestTemplate request. In this example, the request responds with a string value:

```
Message message = new Message();
message.setId(555);
message.setSubject("test subject");
message.setText("test text");

String url = "http://mypretendservice.com/sendmessage";
RestTemplate restTemplate = new RestTemplate();
String response = restTemplate.postForObject(url, message, String.class);
```

You can also specify the Content-Type header in your request:

```
Message message = new Message();
message.setId(555);
message.setSubject("test subject");
message.setText("test text");

HttpHeaders requestHeaders = new HttpHeaders();
requestHeaders.setContentType(new MediaType("application","json"));

HttpEntity<Message> requestEntity = new HttpEntity<Message>(message, requestHeaders);

String url = "http://mypretendservice.com/sendmessage";

RestTemplate restTemplate = new RestTemplate();
ResponseEntity<String> responseEntity = restTemplate.exchange(url, HttpMethod.POST, requestEntity, String.class
String result = responseEntity.getBody();
```

Retrieve RSS or Atom feed

The following is a basic example of loading an RSS feed:

```
String url = "http://mypretendservice.com/rssfeed";
RestTemplate restTemplate = new RestTemplate();
SyndFeed = restTemplate.getForObject(url, SyndFeed.class);
```

It is possible that you need to adjust the Media Type associated with the SyndFeedHttpMessageConverter. By default, the converter is associated with "application/rss+xml" and "application/atom+xml". An RSS feed might instead have a media type of "text/xml", for example. The following code illustrates how to set the media type.

```
String url = "http://mypretendservice.com/rssfeed";

SyndFeedHttpMessageConverter converter = new SyndFeedHttpMessageConverter();
List<MediaType> mediaTypes = new ArrayList<MediaType>();
mediaTypes.add(new MediaType("text","xml"));
converter.setSupportedMediaTypes(mediaTypes);

List<HttpMessageConverter<?>> messageConverters = new ArrayList<HttpMessageConverter<?>>();
messageConverters.add(converter);

RestTemplate restTemplate = new RestTemplate();
restTemplate.setMessageConverters(messageConverters);
SyndFeed feed = restTemplate.getForObject(url, SyndFeed.class);
```

3. Spring Android and Maven

3.1 Introduction

An alternative to downloading the individual library JARs yourself is to use Maven for dependency management. The Maven Android Plugin [http://code.google.com/p/maven-android-plugin] allows developers to utilize Maven's dependency management capabilities within an Android application. Additionally, the Maven Integration for Android Development Tools [http://code.google.com/a/eclipselabs.org/p/m2eclipse-android-integration/] bridges the Maven Android Plugin and the Android Development Tools (ADT) [http://developer.android.com/sdk/eclipse-adt.html] to allow the use of dependency management within Eclipse.

3.2 Example POM

The following Maven POM file [http://maven.apache.org/guides/introduction/introduction-to-the-pom.html] illustrates how to configure the Maven Android Plugin [http://code.google.com/p/maven-android-plugin] and associated dependencies for use with Spring Android Rest Template.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework.android</groupId>
    <artifactId>showcase</artifactId>
    <version>1.0.0.BUILD-SNAPSHOT</version>
    <packaging>apk</packaging>
    <name>spring-android-showcase-client</name>
    <url>http://www.springsource.org</url>
    <organization>
        <name>SpringSource</name>
        <url>http://www.springsource.org</url>
    </organization>

    <build>
        <sourceDirectory>src</sourceDirectory>
        <finalName>${project.artifactId}</finalName>
        <plugins>
            <plugin>
                <groupId>com.jayway.maven.plugins.android.generation2</groupId>
                <artifactId>maven-android-plugin</artifactId>
                <version>2.8.4</version>
                <configuration>
                    <sdk>
                        <platform>3</platform>
                    </sdk>
                    <emulator>
                        <avd>3</avd>
                    </emulator>
                    <deleteConflictingFiles>true</deleteConflictingFiles>
                    <undeployBeforeDeploy>true</undeployBeforeDeploy>
                </configuration>
                <extensions>true</extensions>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
```

```
                <version>2.3.2</version>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>com.google.android</groupId>
            <artifactId>android</artifactId>
            <version>1.5_r4</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.android</groupId>
            <artifactId>spring-android-rest-template</artifactId>
            <version>1.0.0.BUILD-SNAPSHOT</version>
        </dependency>
        <dependency>
            <!-- Using Jackson for JSON marshaling -->
            <groupId>org.codehaus.jackson</groupId>
            <artifactId>jackson-mapper-asl</artifactId>
            <version>1.7.1</version>
        </dependency>
        <dependency>
            <!-- Using Simple for XML marshaling -->
            <groupId>org.simpleframework</groupId>
            <artifactId>simple-xml</artifactId>
            <version>2.4.1</version>
            <exclusions>
                <exclusion>
                    <artifactId>stax</artifactId>
                    <groupId>stax</groupId>
                </exclusion>
                <exclusion>
                    <artifactId>stax-api</artifactId>
                    <groupId>stax</groupId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <!-- Using Android ROME for RSS and ATOM feeds -->
            <groupId>com.google.code.android-rome-feed-reader</groupId>
            <artifactId>android-rome-feed-reader</artifactId>
            <version>1.0.0-r2</version>
        </dependency>
    </dependencies>

    <repositories>
        <!-- For developing with Android ROME Feed Reader -->
        <repository>
            <id>android-rome-feed-reader-repository</id>
            <name>Android ROME Feed Reader Repository</name>
            <url>https://android-rome-feed-reader.googlecode.com/svn/maven2/releases</url>
        </repository>
        <!-- For testing against latest Spring snapshots -->
        <repository>
            <id>org.springframework.maven.snapshot</id>
            <name>Spring Maven Snapshot Repository</name>
            <url>http://maven.springframework.org/snapshot</url>
            <releases><enabled>false</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
```

```
        <!-- For developing against latest Spring milestones -->
        <repository>
            <id>org.springframework.maven.milestone</id>
            <name>Spring Maven Milestone Repository</name>
            <url>http://maven.springframework.org/milestone</url>
            <snapshots><enabled>false</enabled></snapshots>
        </repository>
    </repositories>

</project>
```

3.3 Maven Commands

Once you have configured a Maven POM in your Android project you can use the following Maven command to clean and assemble your Android APK file. Additional goals [http://maven-android-plugin-m2site.googlecode.com/svn/plugin-info.html] are available for use with the Maven Android Plugin.

```
$ mvn clean install
```

The following command starts the emulator specified in the Maven Android Plugin section of the POM file

```
$ mvn android:emulator-start
```

Deploys the application package to the emulator

```
$ mvn android:deploy
```