Features

Introduction

Restlet Framework is mature and scalable, based on a small core and many optional extensions, making it suitable for any kind of web API development, including cross-channel web sites and applications.

Web API support

- Core REST and HTTP concepts have equivalent Java artifacts (Resource, Representation, Connector or Component classes for example).
- Suitable for both client-side and server-side web applications. The innovation is that that it uses the same Java API, reducing the learning curve and the software footprint.
- Concept of "URIs as UI" supported based on the URI Templates standard. This results in a very flexible yet simple routing with automatic extraction of URI variables into request attributes.
- Tunnelling service lets browsers issue any HTTP method (PUT, DELETE, PATCH, etc.) through a simple HTTP POST. This service is transparent for Restlet applications.
- Easy API documentation thanks to WADL support and Swagger integration.

Complete Web server

Contrary to the Servlet API, the Restlet API gives you extensive control on the URI mapping and on the virtual hosts configuration. It includes a powerful Directory class to server static files in a way similar to Apache Web Server. For example, our Restlet.org web site is directly powered by Restlet Framework on a regular JVM!

Here is a more complete list of features provided by the internal Web server: - Static file serving similar to Apache HTTP Server, with metadata association based on file extensions. - Transparent content negotiation based on client preferences. - Conditional requests automatically supported for resources. - Remote edition of files based on PUT and DELETE methods (aka mini-WebDAV mode). - Decoder service transparently decodes compressed or encoded input representations. This service is transparent for Restlet applications. - Log service writes all accesses to your applications in a standard Web log file. The log format follows the W3C Extended Log File Format and is fully customizable. - Powerful URI based redirection support similar to Apache Rewrite module. - Extensive and flexible security support for both authentication and authorization.

Presentation and persistence agnostic

By staying open to all presentation environments and technologies (AngularJS, Android, iOS, Eclipse RCP, GWT, etc.) and all persistence technologies (JDBC, Hibernate, Spring IO, Cassandra, MongoDB, etc.), your investment in Restlet is secured. With very little work, your Restlet applications can be made portable from one environment to the other.

Multiple editions

REST principles have no limit, they can be applied everywhere the Web is and even in places where there is no Internet but needs for communication or effective decoupling. Currently, the Restlet Framework is available in several editions: - Edition for Java SE, to run your Restlet applications in regular JVMs. - Edition for Java EE, to run your Restlet applications in Servlet containers. - Edition for GAE, to run your Restlet applications in Google App Engine cloud platform. - Edition for GWT, to run your Web browser clients, without plugins. - Edition for Android, letting you deploy Restlet applications on mobile Android devices. - Edition for OSGi, letting you deploy Restlet applications on dynamic and embedded OSGi environments.

Servlet compatible

Restlet was an attempt to build a better Servlet API, aligned with the true Web architecture (REST) and standards (HTTP, URI). Therefore the Restlet API has no dependency on the Servlet API, it only depends on the Java SE. However, it is perfectly possible to deploy a Restlet application into Java EE application servers or just Servlet containers. This is possible using an adapter Servlet provided as an extension.

Available Connectors

- Multiple server HTTP connectors available, based on either Eclipse Jetty or the [Simple framework (http://www.simpleframework.org/).
- AJP server connector available to let you plug behind an Apache HTTP server or Microsoft IIS. It is based on Jetty's connector.
- Multiple client HTTP connectors available, based on either Apache HTTP Client or on an NIO-based extension (preview).
- Compact internal HTTP client and server for development and light deployments based on HTTPUrlConnection class. No external dependency needed.
- Client SMTP, SMTPS, POP v3 and POPS v3 connectors are provided based on JavaMail and a custom email XML format.
- Client JDBC connector based on the JDBC API, a custom request XML format and the JDBC WebRowSet interface for XML responses.

- Client FILE connector supports GET, PUT and DELETE methods on files and directories. In addition, it is able to return directory listings.
- Client CLAP connector to access to the Classloader resources.
- Client and server RIAP connectors to access to the Restlet internal resources, directly inside the JVM, relatively to the current application or virtual host or component.
- Client SOLR connector to call embedded Apache Lucene Solr search and indexing engine.

Available Representations

- Built-in support for XML representations (JAX, JibX, DOM or SAX based) with a simple XPath API based on JDK's built-in XPath engine.
- Integration with the FreeMarker template engine
- Integration with the Velocity template engine
- Integration with [Apache FileUpload] http://jakarta.apache.org/commons/fileupload/) to support multi-part forms and easily handle large file uploads from browsers to support multi-part forms and easily handle large file uploads from browsers
- Transformer filter to easily apply XSLT stylesheets on XML representations. It is based on JDK's built-in XSLT engine.
- Extensible set of core representations based on NIO readable or writable channels, BIO input or output streams.
- Support for Atom and JSON standards.
- Integration with Apache Lucene Tika to support metadata extraction from any representation.

Flexible configuration

- Complete configuration possible in Java via the Restlet API
- Configuration possible via Restlet XML and WADL files
- Implementation of the JAX-RS 1.1 standard API (based on JSR-311).
- Deployment as native services is possible and illustrated using the powerful Java Service Wrapper.
- Extensive integration with popular Spring Framework.

- Deployment to Oracle 11g embedded JVM supported by special extension.

Security

- Supports HTTP Basic and Digest authentication (client and server side)
- Supports HTTPS (HTTP over SSL)
- Supports OAuth 2.0 authentication (preview mode)
- Supports Amazon S3 authentication
- Supports Microsoft Shared Key and Shared Key Lite authentication (client side)
- Supports SMTPS (SMTP over SSL) and SMTP-STARTTLS
- Supports POPS (POP over SSL)

Scalability

- Fully multi-threaded design with per-request Resource instances to reduce thread-safety issues when developing applications.
- Intentional removal of Servlet-like HTTP sessions. This concept, attractive as a first sight, is one of the major issue for Servlet scalability and is going against the stateless exchanges promoted by REST.
- Supports non-blocking NIO modes to decouple the number of connections from the number of threads.
- Supports asynchronous request processing, decoupled from IO operations. Unlike the Servlet API, the Restlet applications don't have a direct control on the outputstream, they only provide output representation to be written by the server connector.

Upcoming features

Is something important for you missing? Maybe we are already working on it or are planning to do so.

We suggest that you have a look at our public roadmap or at our issue tracker on GitHub.

Feel free to create some new ones if needed! # First application

Introduction

This first application illustrates how to develop a Restlet application that combines several editions of the Restlet Framework : GAE, GWT, Android and Java SE. It explains the benefits of annotated Restlet

interfaces and of the ConverterService that offers transparent serialization between Restlet representations and Java objects, usable between a server application and several kind of clients.

Table of contents

Requirements

It is based on the following editions of the Restlet Framework : Java SE (JSE), Google App Engine (GAE), Google Web Toolkit (GWT) and Android which must be downloaded separately from this page. It has been tested with the following environments:

- Restlet Framework 2.2 RC 4
- Google App Engine (GAE) 1.7.0
- Google Web Toolkit (GWT) 2.5
- Android 4

GAE doesn't support HTTP chunked encoding, therefore serialized object can't be sent (via POST or PUT) to a GAE server. Since Restlet Framework version 2.1 M4 we have a workaround available that buffers the HTTP entity to prevent chunk encoding. To use it, call the ClientResource.setRequestEntityBuffering(boolean) method with a "true" value. Note that this workaround isn't required for the GWT edition.

Scenario

The server application is hosted on the Google App Engine (GAE) platform. For the sake of simplicity it serves only one resource named "contact", with the following characteristics:

- its relative URI is "/contacts/123"
- it supports the GET, PUT and DELETE methods.

- it represents a simple "contact" object.

The "contact" object has the following attributes:

- firstname
- lastname
- age
- home address (actually an instance of a "Address" class): line1, line2, zipcode, city and country

This resource will be requested for several kind of clients:

- GWT client page
- Android application
- Java SE client

Archive content

The full source code (without the required archives) is available here: firstapplication.zip (application/zip, 1.0 MB)

It contains the full source code of three Eclipse projects with:

1. Project that contains both the GAE server and the GWT client code
2. Project that contains the source code of the Android client
3. Project that contains the source code of the Java SE client

Common classes

The following classes are available on the three project. They are used by the server and the clients in order to produce the serialized representation of the Contact object and to deserialize incoming representations.

- Contact
- Address
- ContactResource.

ContactResource is an interface annotated with Restlet annotations:

```
public interface ContactResource {
   @Get
   public Contact retrieve();
```

```
    @Put
    public void store(Contact contact);

    @Delete
    public void remove();
}
```

It represents the contract passed between the client and the server.

When using collections of objects as method parameters, you need to use concrete classes if you intend to have GWT clients. For example use ArrayList<Contact> instead of List<Contact>.

GAE server

We propose to host the server application on the GAE platform. The server project relies on the following JAR files:

- org.restlet.jar: core archive (*GAE edition*)
- org.restlet.ext.gwt.jar: GWT server-side extension to convert Java objects to a GWT-specific serialization format (*GAE edition*)
- org.restlet.ext.gae.jar: GAE server-side extension (*GAE edition*)
- org.restlet.ext.servlet.jar: Servlet extension to deploy the Restlet application in GAE (*GAE edition*)
- org.restlet.ext.jackson.jar: Jackson extension used to generate JSON representations of the contact resource (*GAE edition*)
- com.fasterxml.jackson.core.jar, com.fasterxml.jackson.databind.jar, com.fasterxml.jackson.annotations.jar: archives of the Jackson libraries required by the Jackson extension, and available in the GAE edition.

See also the "readme.txt" file located in the sources file. It list also all necessary binaries taken from the GAE platform.

The server-side resource implements the annotated interface.

```
/**
 * The server side implementation of the Restlet resource.
 */
```

```
public class ContactServerResource extends ServerResource implements
ContactResource {


  private static volatile Contact contact =

      new Contact("Scott", "Tiger", new Address("10 bd Google", null,
"20010", "Mountain View",

              "USA"), 40);


   public void remove() {

      contact = null;

   }


   public Contact retrieve() {

      return contact;

   }


   public void store(Contact contact) {

      ContactServerResource.contact = contact;

   }

}
```

This resource is then exposed by the server application:

```
   @Override

   public Restlet createInboundRoot() {

      Router router = new Router(getContext());


      // Serve the files generated by the GWT compilation step.

      router.attachDefault(new Directory(getContext(), "war:///"));

      router.attach("/contacts/123", ContactServerResource.class);
```

```
        return router;
    }
```

GWT client

The GWT client relies only on the core Restlet JAR (org.restlet.jar) provided in the GWT edition.

In order to get the Contact object, a proxy class is required. This is an interface that inherits on a specific interface (delivered by the GWT edition of the Restlet Framework):

```
public interface ContactResourceProxy extends ClientProxy {

    @Get

    public void retrieve(Result<Contact> callback);


   @Put

    public void store(Contact contact, Result<Void> callback);


    @Delete

    public void remove(Result<Void> callback);

}
```

This interface looks like the ContactResource interface, expect that it adds a callback to each declared methods, due to the asynchronous nature of the GWT platform and the underlying AJAX mechanism offered by web browsers.

The type of the callback is not limited to the Result interface of the Restlet Framework, it can also be the usual AsyncCallback interface provided by GWT. Thus it allows you to easily migrate an existing GWT-RPC code base to GWT-REST with Restlet.

Then, the following code allows you to request and handle the Contact resource:

```
ContactResourceProxy contactResource =
GWT.create(ContactResourceProxy.class);


// Set up the contact resource

contactResource.getClientResource().setReference("/contacts/123");
```

```
// Retrieve the contact
contactResource.retrieve(new Result<Contact>() {
   public void onFailure(Throwable caught) {
      // Handle the error
   }

   public void onSuccess(Contact contact) {
      // Handle the contact, for example by updating the GWT interface
      // Contact fields
      cTbFirstName.setText(contact.getFirstName());
      cTbLastName.setText(contact.getLastName());
      cTbAge.setText(Integer.toString(contact.getAge()));
   }
});
```

Here is a screenshot of the GWT client page once the user has clicked on the GET button.

serialization gwt screenshot

serialization gwt screenshot

In order to update the contact, simply complete your contact object and invoke the "store" method as specified by the proxy interface:

```
contactResource.store(contact, new Result<Void>() {
   public void onFailure(Throwable caught) {
      // Handle the error
   }

   public void onSuccess(Void v) {
      // Display a dialog box
      dialogBox.setText("Update contact");
```

```
        textToServerLabel.setText("Contact successfully updated");

        dialogBox.center();

        closeButton.setFocus(true);

    }

});
```

Android client

The Android client project relies only on the core Restlet JAR (org.restlet.jar) provided by the Android edition of the Restlet Framework.

The contact object will be serialized between the GAE server and the Android client (in both directions) using the standard Java serialization process. No additional interface is required except the ContactResource interface furnished by the server.

```
// Initialize the resource proxy.

ClientResource cr = new ClientResource("http://restlet-example-serialization.appspot.com/contacts/123");

// Workaround for GAE servers to prevent chunk encoding

cr.setRequestEntityBuffering(true);


ContactResource resource = cr.wrap(ContactResource.class);


// Get the remote contact

Contact contact = resource.retrieve();
```

In order to update the contact, simply use this instruction:

```
// Update the remote contact

resource.store(contact);
```

The internal HTTP client has been rewritten using the java.nio.package. This may lead, on some android devices, to encounter this kind of exception: **java.net.SocketException: Bad address family**. In this case, you can turn off the IPv6 preference as follow: System.setProperty("java.net.preferIPv6Addresses", "false");

Here is a screenshot of the Android user interface.

serialization android screenshot

serialization android screenshot

Java SE client

Get the full Contact object

The same code used on the Android application allows you to get the full Contact object:

```
ClientResource cr = new
ClientResource("http://restlet-example-serialization.appspot.com/contacts/12
3");

// Get the Contact object

ContactResource resource = cr.wrap(ContactResource.class);

Contact contact = resource.retrieve();


if (contact != null) {

    System.out.println("firstname: " + contact.getFirstName());

    System.out.println(" lastname: " + contact.getLastName());

    System.out.println("     nage: " + contact.getAge());

}
```

This code produces the following ouput on the console:

```
firstname: Scott

 lastname: Tiger

      age: 40
```

Get a JSON representation

In case the Contact class is not available, you can still retrieve a JSON representation by setting the client preferences when retrieving the resource's representation:

```
cr.get(MediaType.APPLICATION_JSON).write(System.out);
```

which produces the following output:

```
{"age":40,"firstName":"Scott","homeAddress":
{"country":"USA","city":"Mountain View","line1":"10 bd
Google","line2":null,"zipCode":"20010"},

"lastName":"Tiger"}
```

First client

Introduction

As we mentioned in the introduction, the Restlet Framework is at the same time a client and a server framework. For example, you can easily work with remote resources using its HTTP client connector.

A connector in REST is a software element that enables the communication between components, typically by implementing one side of a network protocol. Restlet provides several implementations of client connectors based on existing open-source projects. The connectors section lists all available client and server connectors and explain how to use and configure them.

Here we will retrieve the representation of an existing resource and output it in the JVM console:

```
// Outputting the content of a Web page

new ClientResource("http://restlet.org").get().write(System.out);
```

If you are running your client behind a proxy, please check this page to pick an HTTP client that can be configured. The internal HTTP client doesn't support proxies at the moment.

The next example sets some preferences in your client call, like a referrer URI:

```
// Create the client resource

ClientResource resource = new ClientResource("http://restlet.org");


// Customize the referrer property

resource.setReferrerRef("http://www.mysite.org");


// Write the response entity on the console

resource.get().write(System.out);
```

After those first two steps, let's now develop a more complete Restlet application, taking advantage of the various editions of the Restlet Framework.

First server

Introduction
Let's first see how the Restlet Framework can listen to client requests and reply to them. We will use the internal Restlet HTTP server connector (even though it is possible to switch to others such as the one based on Jetty) and return a simple string representation "hello, world" as plain text. Note that the FirstServerResource class extends the base org.restlet.resource.ServerResource class provided by the Restlet API:

```
public class FirstServerResource extends ServerResource {


  public static void main(String[] args) throws Exception {
    // Create the HTTP server and listen on port 8182
    new Server(Protocol.HTTP, 8182, FirstServerResource.class).start();
  }


  @Get
  public String toString() {
    return "hello, world";
  }


}
```

If you run this code and launch your server, you can open a Web browser and hit the http://localhost:8182. Actually, any URI will work, try also http://localhost:8182/test/tutorial. Note that if you test your server from a different machine, you need to replace "localhost" by either the IP address of your server or its domain name if it has one defined.

So far, we have mostly showed you the highest level of abstraction in the Restlet API, with the ServerResource classes. But as we move forward, you will discover that this class is supported by a broad Java API, mapping all REST and HTTP concepts to a set of Java classes, interfaces and annotations.

Let's now illustrate how to use this API on the client-side. # First steps

This section will give you a first taste of the Restlet Framework:

- First server
- First client
- First application # Part I - Introduction

User guide overview

This is the official User Guide for version 2.3 of the Restlet Framework (http://restlet.org).

This document is edited in a collaborative way via this GitHub repository as a set of Markdown pages.

We are very happy to have you as a new user and hope that you will have as much fun learning and using this technology as we had designing and developing it. Be ready to radically change the way you think and build web applications. With Restlet in our backpack, we are certain that you will enter the REST and Web API world with the best toolkit available!

As a truly open source project, we not only consider you as a user but as a potential contributor. You will soon find ways to contribute back to the project, by filing a bug or enhancement reports, by submitting documentation or code improvements or by helping other users or developers in the mailing lists. Every contribution is valuable to the community and we give credit back to our team by listing all the contributors on our team page.

Let's get started and again: welcome to the Restlet Team !

Framework overview

Restlet is a comprehensive yet lightweight RESTful web API framework for Java that lets you embrace the architecture style of the Web (REST) and benefit from its simplicity and scalability. By using our innovative framework, you can start blending your web services, web sites and web clients into uniform web applications!

Restlet has a light core but thanks to its pluggable extension, it is also a comprehensive REST framework for Java. It supports all REST concepts (Resource, Representation, Connector, Component, etc.) and is suitable for both client and server Web applications.

It supports major Web standards like HTTP, SMTP, XML, JSON, OData, OAuth, RDF, RSS, WADL, and Atom.

Many extensions are also available to integrate with Servlet, Spring, Jetty, Simple, JAXB, JAX-RS, JiBX, Velocity, FreeMarker, XStream, Jackson, SLF4J, SDC and many more!

Special editions for Android, GWT, GAE, Java SE, Java EE and OSGi are also available and kept synchronized with an automated porting process.

Community support

If you can't find an answer to your question in this document, please use our discussion lists and read the recommended books.

We especially recommend our Restlet in Action book published by Manning.

Professional support

If you can't wait to get an answer or need to keep your questions confidential, you can also obtain private support by buying a professional support plan from the creator of the Restlet Framework.

What's new in version 2.3

Introduction

In the next sections, you will get a synthesis of the major changes done to the Restlet Framework in version 2.3.

Main changes

- Java 7 requirement
- Many bug fixes

Migration guide from version 2.2 to 2.3

This section intends to explain the main differences between the Restlet 2.2 and 2.3 releases and to help you migrate your existing applications. Both releases are meant to be compatible at the API level, so you should at most observe deprecate features while upgrading.

Note that if you intend to migrate directly from 1.1 to 2.3, you should really consider migrating first from 1.1 to 2.0. For migration instructions between 1.1 and 2.0, you can check this page.

Replace all JAR files

Restlet JARs and dependencies

Deprecated API features

The next step is to look at each deprecated feature and look in the Javadocs at the preferred alternative in version 2.3. API improvements
==========

Packages restructuring

In order to simplify even more the learning and deployment of Restlet applications, the Restlet API and its implementation (ie. the Restlet Engine) have been merged into a single module (ie. JAR, bundle). All engine classes were moved to org.restlet.engine.

In addition, all extensions are now located under a org.restlet.ext root package. This means that all extensions previously under "com.noelios.restlet.ext" have been moved. Note that some classes from the core Restlet API have been moved to the Restlet Engine such as:

- org.restlet.util.Engine moved to org.restlet.engine.Engine
- org.restlet.util.Helper moved to org.restlet.engine.Helper
- org.restlet.util.ByteUtils moved to org.restlet.engine.io.BioUtils
- org.restlet.util.DateUtils moved to org.restlet.engine.util.DateUtils

Note also, that there were two Spring extensions in Restlet 1.1. Now they are merged under the org.restlet.ext.spring extension.

In order to ensure a cleaner separation between resource and representation artifacts, we have added a new “org.restlet.representation” package and moved all representation classes to it. The “org.restlet.resource” package is now more focused, allowing us to welcome our new resource API and to move in related classes such as Finder and Handler. We have also added a “org.restlet.routing” package where we moved Filter, Redirector and Router related classes.

Root package

The org.restlet.Uniform abstract class has been refactored into an interface with a single handle(Request, Response) method. Its logic has been moved to the org.restlet.Client class. This allows us to leverage it for asynchronous calls, as we already do in the GWT edition.

We moved Message, Request and Response classes from the "org.restlet.data" to the "org.restlet" root package as they are core artifacts of the API used in Uniform interface. Requests and responses can now be aborted, useful to save bandwidth where undesired calls are handled.

In order to support both inbound and outbound routing and filtering of calls for Restlet applications, we have added two properties: “inboundRoot : Restlet” and “outboundRoot : Restlet” to the Application class and deprecated the “root : Restlet” property. This will allow you to easily provide features such as preemptive authentication with the recently added ClientResource class.

The component XML configuration has been extended to support all existing properties and features available in the Java API.

Data package

Nearly all standard HTTP headers are now fully supported by the Restlet API including “Age”, “Authentication-Info”, “Date”, "Expect", “If-Range”, “Retry-After”, "Via", “Warning”, and the frequently asked “Cache-Control” header! For a detailled mapping of those headers to the Restlet API, please consult this page of the user guide.

Character sets support was enhanced, fixing an issue with Macintosh and adding new constants in the CharacterSet class for all common ones defined by IANA. Also, the default language used for representation is now dynamically retrieved based on the JVM setting instead of English/US.

Representation package

Added AppendableRepresentation for dynamic generation of StringRepresentation instances.

Resource package

After a long experience with our class-driven Resource API introduced in Restlet 1.0 and the recent introduction of the annotation-driven JAX-RS API (that we support as a special Restlet extension), we felt it was time for us to step-back and propose a better solution, taking the best of both experiences.

We initiated a specification effort in our developers wiki at the beginning of this year, taking into account all the enhancement requests, issues and feed-back made by our community. Here is the overall architecture:

The new design uses three foundation classes (UniformResource, ClientResource and ServerResource) which support just four annotations by default: @Get, @Post, @Put and @Delete. New ones can be defined to support extension methods like @Copy and @Move for WebDAV. Note that the support for annotations can be turned off and is not necessary to develop resources.

This new design provides the best of both worlds, the power and flexibility of Restlet 1.1 and the expressiveness and additional abstraction offered by JAX-RS. In addition, it goes beyond those existing APIs by offering a uniform support for both server and client resources !

The ServerResource class cleanly integrates the annotation-based and method-based call processing : the annotation-based processing extends the usual method-base processing.

ClientResource also support for annotated interfaces via the creation of dynamic proxies. It also automatically follows redirections when possible. Access to the underlying ClientResource instance for dynamic proxies is possible via a ClientProxy interface automatically implemented, like for the GWT edition. Proxies for child or parent resource can also easily be obtained, reusing the current resource like a prototype resource!

Asynchronous call handling is now available on both on the server-side and the client-side. The callback mechanism used was inspired from GWT and is fully consistent with our Restlet edition for GWT, ensuring a greater portability.

Routing package

The "org.restlet.routing" package has been enhanced with new Validator and Extractor filters containing logic found in the new deprecated Route class, replaced with a more specific TemplateRoute class. Those changes should be transparent for most applications using the Router class to attach Restlets and resources using URI templates.

We also moved Template and Variable classes from "org.restlet.util" to "org.restlet.routing" package.

Security package

The refactoring of the Restlet security model has been the most requested change since the Restlet 1.0 release. Fortunately, after a long maturation period, it has finally made a huge step forward, materialized by the addition of an “org.restlet.security” package. The new design is the synthesis of many contributions and discussions from the community. Care has been taken to keep a separation of concern between Components and Applications regarding security.

Security realms now have a lifecycle allowing the initialization from a relational database, a file or a LDAP directory.

The API is also extensible and offers a good foundation for your new security efforts.

Service package

Refactored the services to facilitate the addition of new ones by users in their applications or their components.

The TunnelService now supports “X-HTTP-Method-Override” header.

More powerful ConverterService which can automatically serialize POJOs into XML, JSON and other serialization formats such as regular Java serialization or long term bean persistence. We leverage XStream for automatic XML/JSON marshalling and unmarshalling. Note that the JAX-RS extension now also relies on this improved conversion service! Converters were added for all relevant Restlet extensions, allowing usage of high-level classes in annotated Restlet interfaces for example.

Editions

Introduction

The packaging has been reworked to provide **separate distributions for each edition** that we support: Java SE, Java EE (with Servlet support), Google Web Toolkit, Google App Engine and Android.

In addition, the porting of the main source code base to each edition is now fully automated, ensuring a constant synchronization in term of features and bug fixes. All extensions and API features aren’t (or can’t be) supported in all editions, but the API is consistent. As a side effect, the “org.restlet.gwt” package has been moved to the regular “org.restlet” one.

Java SE edition (JSE)

This is the usual distribution previously available in version 1.1, without the Servlet related extensions.

Java EE edition (JEE)

This is the usual distribution previously available in version 1.1, without the standalone connectors such as Jetty, Grizzly, Netty or Simple.

Google App Engine edition (GAE)

This edition is based on the Java EE edition as GAE requires the deployment of web applications in a constrained Servlet container.

Google Web Toolkit edition (GWT)

GWT object serialization support, based on annotated Restlet interfaces was also added for the GWT edition, leveraging GWT’s deferred binding mechanism and GWT-RPC serialization format! See this related post for details. This achieves the same level of productivity than GWT-RPC, in a RESTful way.

Android edition
Port of the Crypto, Apache HTTP Client, JAAS, JSON, Net extension (without FTP client). # Extensions for version 2.0

Crypto extension
New extension that contains support for Amazon S3 and Windows Azure client HTTP authentication (Shared Key and Shared Key Lite) schemes.

The support for HTTP DIGEST has been vastly improved, especially on the client-side, with proper mapping of its properties to AuthenticationInfo (new), ChallengeMessage (new), ChallengeRequest and ChallengeResponse classes.

FreeMarker extension
FreeMarker templates can now be loaded via the Context's client dispatcher and relatively to a base URI.

GWT extension (server-side)
Added an ObjectRepresentation class to the GWT edition and to the "org.restlet.ext.gwt" server extension. This allows transparent serialization of Java objects leveraging GWT-RPC serialization mechanism, but using your REST APIs.

HTTP Client extension (Apache)
The extension has been updated from 3.1 to 4.0 version. Note that some parameters have been changed, so be sure to verify your configuration. In addition, it is now possible to specify a different proxy server to use for each Restlet connector.

JAAS extension
Following the security API enhancements and refactorings, the classes relying on the javax.security.auth package have been moved to a new JAAS extension**.**

Jackson extension
New Jackson extension added, offering a nice alternative to the existing XStream extension for JSON object serialization (based on Jettison).

JAX-RS extension
Now leverages the new Security API.

Jetty extension
Updated to leverage the recent Jetty 7.0 version now hosted at Eclipse.

Lucene extension
In addition, a Lucene extensionhas been created to host the Solr client connector contributed by Rémi Dewitte who will lead this extension.

There is also a TikaRepresentation available to leverage Lucene Tika subproject when extracting metadata from representations.

Net extension

A new FTP client connector was added in the “org.restlet.ext.net” extension, based on the JDK’s URLConnection class. It is limited and only support GET methods.

Netty extension

New extension leveraging the new NIO framework from JBoss. Provides HTTP and HTTPS server connectors.

OData extension

A new extension for Microsoft ADO.NET Data Services technology (previously known as “project Astoria”) was added, later renamed to WCF Data Services then OData.

It provides a high-level client API based on the ClientResource class that lets you access remote OData services, typically hosted in an ASP.NET servers or on the Windows Azure cloud computing platform. The extension contains both a code generator for the representation beans and a runtime layer.

Advanced features such as projections, blobs, server-side paging, row counts, customizable feeds or version headers are supported.

This extension is in the "org.restlet.ext.odata" package and depends on "org.restlet.ext.atom" and "org.restlet.ext.xml" extensions. The extension is also available on the Android edition of the Restlet Framework.

RDF extension

As announced when we presented the roadmap, we want to make Restlet a great framework for building applications for the Semantic Web. The relationship between REST and RDF is perfect and builds around the concept of resources and their representations (REST) and the expression of meaningful links between those resources (RDF).

In Restlet 2.0, we have added Literal, Link, LinkReference, LinkSet and RdfRepresentation classes. That makes it easy to build a RDF graph, like you would use a DOM API to build and XML document..

This extension contains a full RDF API, leveraging the Restlet API, and capable of processing RDF documents either in a DOM-like way or in a SAX-like way. It is also capable of writing large RDF documents is a SAX-like way. We currently support two serialization formats: RDF/XML, RDF/n3, Turtle and N-Triples. In the next version we will extend those formats to Turtle and NTriples.

A RdfClientResource class facilitates the navigation in the Web of Data.

ROME extension
A new ROME extension was added to support several versions of RSS and Atom syndication feeds formats. This extension is complementary with the existing Atom extension which is fully based on Restlet API.

Servlet extension
Improved to support multiple declarations of the ServerServlet in the same Servlet application.

SLF4J extension
The Restlet logging, based on JULI (java.util.logging), now has an extension mechanism allowing an efficient redirection to alternate mechanisms like log4j as explained here. A new SLF4J extension has been added to facilitate the replacement of Restlet's default logger facade.

Spring extension
Updated to leverage Spring Framework version 3.0.

XML extension
New "org.restlet.ext.xml" extension including XML related classes previously in the core Restlet API. This ensures that the core Restlet API stays as consistent as possible across all editions. In this case, those features weren't available in Android.

XStream extension
Added an "org.restlet.ext.xstream" extension providing transparent serialization between Java objects and XML or JSON. Connectors
==========

Content negotiation
Content negotiation was rewritten to support all possible dimensions such as media type, language, character set or encoding.

CLAP client
Enhanced the CLAP connector to support a default authority ("class") for shorter URIs (ex: "clap:///org/restlet/Uniform.class"). Added LocalReference#createClapReference(int, Package) and createClapReference(Package) methods to help building shorter CLAP URIs.

Internal HTTP connectors
The internal HTTP connectors were replaced with new ones based supporting the new asynchronous processing features in Restlet API. They are actually the only connectors for now, beside the GWT edition, supporting those new asynchronous capabilities which should be

considered as a preview feature at this point. In version 2.1 we attempt to support them in alternative connectors such as Jetty, Grizzly and Netty.

In addition, the new design of the internal connector is asynchronous in nature and will provide you production ready performance when we leverage non-blocking NIO. This is working in the Restlet Incubator but is only planned for the next 2.1 version. For now, you should mainly use these connectors for development purpose and configure connectors such as Jetty and Apache HTTP Client when deploying to production.

Internal JAR and ZIP clients

Client connectors for the ZIP and JAR pseudo-protocols were added.

RIAP connectors

Added client and server RIAP connectors that use a protected singleton unique in the JVM.

Misc

Enhanced Maven support

A long time ago, we offered a Maven distribution via our own Maven repositories and regularly we try to improve its quality, for example working with Buckminster users to adjust our Maven metadata. However, we use a custom forge based on Ant as our official build system and this has been causing some pains to Maven developers and putting some barriers for potential contributors.

Thanks to ideas and contributions from the community, we are now providing Maven POM files in our SVN repository as an alternative way to build Restlet. Of course, those POM files are the same that are distributed in our Maven repository and are consistently synchronized with our main Ant script to ensure that they don't diverge in term of dependency versions for example.

For details on building Restlet with Maven, please read this short page on our developers wiki. Note that we have also adjusted our Maven GroupId (only "org.restlet" is used now) are redistributed third-party libraries are now packaged with a "org.restlet.lib." ArtifactId prefix.

Licensing changes

Eclipse Public License 1.0 is an additional licensing option offered

OAuth extension

The OAuth extension available in Restlet 1.1 has been moved to the Restlet Incubator as it would require too much work to get aligned with the new Restlet security and resource APIs in version 2.0. Note that this

is temporary and we definitely want to reintroduce this feature in Restlet 2.1.

What's new in version 2.0

Introduction

As a large amount of new features were added since version 1.1 of the Restlet Framework, major packages reorganization and API refactoring was done on the Restlet API (including the core Resource API) and a growing number of special Restlet editions (Java SE/EE, Google Web Toolkit, Google App Engine and Android), we decided to rename the 1.2 version into a major 2.0.

In the next sections, you will get a synthesis of the major changes done to the Restlet Framework.

References

For more details, you can read the 2.0 announce on our blog as well as the full list of changes.

Migration guide from version 1.1 to 2.0

Introduction

This section intends to explain the main differences between the Restlet 1.1 and 2.0 releases and to help you migrate your existing applications.

Adjust your imports

The Restlet API and several extensions have been deeply restructured and enhanced as explained earlier, but all 1.1 artifacts were either moved from one package to another or deprecated, but are still available.

When you upgrade a Restlet 1.1 project with 2.0 dependencies, your existing code will look broken as many imports won't be resolved by your favorite IDE. However, simply adjusting the package imports (using the dedicated feature of your IDE, like the "Organize Imports" feature in Eclipse) will fix those issues. Indeed, the classes themselves have either not changed their API at all or have been properly deprecated.

Verify your routers

In version 1.1, the default router configuration was trying to match the start of the URI of incoming requests (using Template.MODE_STARTS_WITH constant for the Router#defaultRoutingMode property) and was including the query string when matching the URI against the template (setting Router#defaultMatchingQuery to "true").

In version 2.0, we decided to change those defaults as we would tend to match URIs that could end with anything, without control. Now the default matching mode is Template.MODE_EQUALS and the default query matching property is set to "false".

If you still want to include the query string in your URI templates, then you do need to restore the old values. Otherwise, nothing needs to be changed. Note that another issue with this approach is that query variables must be provided by the user in the exact same order as the URI template, even though people tend to consider that this order shouldn't matter.

Replace usage of deprecated features

The next step is to look at each deprecated feature and look in the Javadocs at the preferred alternative in Restlet 2.0. The most significant change is related to the resource API which are been greatly enhanced and simplified at the same time. Basically, instead of extending the Resource class for your REST server resources, you should now extend ServerResource.

In addition, you can now separate the resource contract in an annotated Java interface, implemented by your ServerResource subclass. The advantage of doing this is that your contract is well isolated and can be written first. Most importantly, it can be used on the client-side by the ClientResource class to remotely call your server resource. See an example in this first application page.

What's new in version 2.2

Introduction

In the next sections, you will get a synthesis of the major changes done to the Restlet Framework in version 2.2.

For more details, you can read the 2.2 announce on our blog as well as the full list of changes.

Main changes

- Java 6 requirement
- Apache License 2.0 option
- Jackson extension now supports JSON, JSON binary (Smile), XML, YAML and CSV formats
- Internal HTTP client and server now based on stable JDK Net classes
- reduced size of org.restlet.jar by about 45Kb

- best default HTTP client on Android
- moved previous internal connector to new NIO extension (preview)
- New Swagger extension (only JAX-RS API support for now)
- New Thymeleaf templating extension
- New GSON extension, supporting Google’s JSON serialization library
- New Guice extension, supporting Google's dependency injection library
- OAuth 2.0 final RFC (preview)
- added client support for HTTP OAuth MAC authentication
- HTTP PATCH method support
- Javadocs artefacts added to Maven repository
- Annotation based JAX-RS client (not compliant with JAX-RS 2.0)
- JSONP filter to workaround single origin policies in browsers
- Converter exceptions are now properly transmitted
- OSGi extension support RESTful inter-bundle communication
- Updated over 25 dependencies (Jackson, Jetty, Apache, MongoDB, etc.)
- Forge migration to GitHub and Travis CI
- User questions migration to StackOverflow
- Easier contribution as modules are now regular Eclipse projects (not PDE plugins)
- Many bug fixes

Migration guide from version 2.1 to 2.2

This section intends to explain the main differences between the Restlet 2.1 and 2.2 releases and to help you migrate your existing applications. Both releases are meant to be compatible at the API level, so you should at most observe deprecate features while upgrading.

Note that if you intend to migrate directly from 1.1 to 2.2, you should really consider migrating first from 1.1 to 2.0. For migration instructions between 1.1 and 2.0, you can check this page.

Replace all JAR files

Restlet JARs and dependencies

Deprecated API features

The next step is to look at each deprecated feature and look in the Javadocs at the preferred alternative in version 2.2. # What's new

- In version 2.3
- In version 2.2
- In version 2.1
- In version 2.0
- In version 1.1 # Extensions in version 1.1
- Full support for WADL, a popular description language for RESTful application. It can be used to configure components, applications and resources. In addition, existing Restlet applications can be enhanced to dynamically expose a REST API documentation as either a raw WADL XML document or as a converted HTML document. This magic documentation feature, fully customizable, works by introspecting application resources and using an HTML template provided by Yahoo! Several features were sponsored by NetDev Ltd.
- New JAXB extension for easy XML to POJO mappings. JAXB is a standard annotation-based API.
- New JiBX extension providing an efficient and flexible alternative to JAXB for XML to object serialization.
- New Spring extension to provide even more integration possibilities with Spring, Servlet and Restlet at the same time. The existing Spring API extension has also been improved based on user feed-back and contributions.
- Atom extension has been updated to conform to the latest Atom Publishing Protocol specifications. The extension now allows both the retrieval and the writing of APP service documents and Atom feeds.
- Extensive implementation of the JAX-RS 1.0 API (developed by JSR-311) was contributed by Stephan Koops. We are now waiting for access to the TCK for verification of completeness.

- New OAuth extension was contributed by Adam Rosien, leveraging a new pluggable authentication scheme. OAuth is a standard related to OpenID for securing API authorization. It is typically used as secure way for people to give an application access to their data.
- New XDB extension providing integration with Oracle embedded JVM contributed by Marcelo Ochoa.
- New Restlet-GWT module provided as a port of the Restlet client API to the Google Web Toolkit 1.5 AJAX platform. This module also supports HTTP authentication.

What's new in version 1.1

Introduction

After one year and half of development, the Restlet project has made tremendous progress. We will try to summarize here the main benefits that you can expect by migrating from Restlet 1.0 to the latest 1.1 version.

Uniform Development Environment

Truly boost the productivity of your team by leveraging our uniform Restlet API:

- Develop Client-side, Server-side or Unified Applications using the exact same RESTful concepts and classes
- Develop Web Services, Static or Dynamic Web Sites in the same way, blending them into RESTful Web Applications
- Support multiple protocols (HTTP, File, CLAP, WAR, POP3, SMTP, ...) using the exact same API and a pluggable connector mechanism

In order to deal with your largest development needs, support for application modularization has been added. It is now easy to split a large application into several ones, and still be able to efficiently communicate. There is a new internal protocol (RIAP) to do private RESTful calls between applications hosted in the same JVM.

Improved Deployment Flexibility

The Restlet project has always been open to other technologies and tries to give its developers the maximum freedom in term of deployment. We don't try to lock you down to a specific technology. Here is the current list of deployment environments that you can target:

- Standalone JAR

- Servlet containers (Tomcat, Jetty, Resin, etc.)
- JEE application servers (WebLogic, WebSphere, Glassfish, JBoss AS, etc.)
- Spring container (several integration options)
- OSGi environment (Eclipse Equinox, Apache Felix, etc.)
- Oracle database (XDB technology)
- Google Web Toolkit (Restlet-GWT module for Rich Web Clients)
- JAIN/SLEE

Most of the time, your Restlet Applications code will be fully portable, requiring only simple deployment configuration adjustments.

Regarding configuration, we now support our own compact XML syntax. This can be very convenient, in addition to the usual programmatic way, to let administrators tweak the configuration of Restlet components, connectors and virtual hosts, without recompiling the source code. This is also possible to leverage Spring XML configuration mechanism to similar results.

Automatic REST API Documentation

Don't you think that having a comprehensive, fully customizable and always up-to-date documentation for your REST API is essential? We do and made some major enhancements on this front in collaboration with our customers and users.

In this new version, we think we have the most complete and useful support for WADL (Web Application Description Language), the equivalent of WSDL for RESTful applications. Thanks to our WADL extension, you can now have your whole application, or just each single resource self-described. The WADL documentation can be dynamically generated from your source code and exposed either as machine processable XML (WADL) or as human readable HTML (WADL) documents.

Significantly Improved Performance

Top notch performance has always been a core concern for us. This led to the implementation of a pluggable connector mechanism in Restlet 1.0, and the shipping of several HTTP connector options such as Jetty and Simple.

In Restlet 1.1, we continued on this path and added a brand new Grizzly HTTP server based on the very responsive and scalable NIO framework developped by the Sun Glassfish project. This connector fits

perfectly with the NIO provisions in the Restlet API, resulting in direct disk-to-socket sending of static files and reduced memory and CPU usage!

We have also added very convenient internal HTTP client and server connectors that ensure that you can get started right away with your Restlet development needs. This is also a perfectly suitable option for compact or embedded deployment scenarios.

More Licensing Options

Led by the founder of the Restlet project, the Noelios Technologies company is now the main copyright holder of the Restlet source code. It funds most development efforts and incorporate contributions from the active Restlet community.

As a result, it can offers flexible licensing options to suit the needs of all users of the technology:

- LGPL 2.1
- LGPL 3.0
- CDDL 1.0
- Commercial license (optionally transferable)

More Extensions

In order to keep our core library light and focused, we have proposed an extension mechanism in Restlet 1.0. This makes sure that we don't force technology choice, beside the RESTful design at the core of the project, onto our users. Instead, we prefer to give them open integration options with their favorite technologies such as JavaMail, FreeMarker, JSON or Velocity.

In Restlet 1.1, we have continued those efforts and added the following extensions:

- JAX-RS to support the new annotation-based RESTful API
- WADL for "Automatic REST API Documentation" (see details above)
- JiBX and JAXB as two strong alternatives for XML serialization
- OAuth for REST API access delegation
- XDB for deployment in Oracle databases
- Atom for feed reading or writing
- SSL for more security options

Migration guide from Restlet 1.0 to 1.1

Introduction

This guide intends to explain the main differences between the Restlet 1.0 and 1.1 releases and to help you migrate your existing applications.

It presents a set of important modifications related to:

- Handling of context
- Handling of resources
- Usage of the Servlet Adapter
- Access to "current" objects
- Access to the original resource reference of the current request
- Handling of statuses and exceptions
- Miscellaneous
- List of deprecations (mainly renamed methods)

Important breaking changes

Handling of context

The handling of Context has been significantly refactored in the 1.1 release due to security reasons. In release 1.0, a Component shared its context between all attached applications. It appeared that this design could lead to conflicts where an application would update the component's context and thus impact the behavior of the other applications.

Here are the main changes implemented in the 1.1 release:

- Each connector now has its own instance of Context to allow setting of different parameter values on different connectors.
- A new Context#createChildContext() method has been added to create a new isolated child context from a parent component's context.
- The semantics of the Application(Context) constructor has changed in that the given context is no more the parent context, but the application's context.
- The "attach" methods on virtual hosts and internal component router now automatically set the child context on the target application with a null context is detected.

Thus, we encourage you to override the default Application() constructor instead of the Application(Context) one which is only useful in a few cases where you need to immediately have access to the application's context.

Another consequence is that parameters or attributes are not copied from the component to the application by the default implementation of the Context#createChildContext() method which is typically used to prepare the Application's context based on the parent Component's context.

Sample code

When a component instantiate an Application, or any kind of Restlet, it is no more mandatory to specify the context. Taking a simple component that attaches an application, here is the ancient way to achieve this:

```
Component component = new Component();


// Add a new HTTP server listening on port 8182.
component.getServers().add(Protocol.HTTP, 8182);


// Attach the sample application.
component.getDefaultHost().attach(new
FirstStepsApplication(component.getContext()));
```

And now, here is the new way:

```
Component component = new Component();


// Add a new HTTP server listening on port 8182.
component.getServers().add(Protocol.HTTP, 8182);


// Attach the sample application.
component.getDefaultHost().attach(new FirstStepsApplication());
```

If you still need to handle the context in the Application constructor, you must do as follow:

```
Component component = new Component();
```

```
// Add a new HTTP server listening on port 8182.
component.getServers().add(Protocol.HTTP, 8182);


// Attach the sample application.
component.getDefaultHost().attach(new
FirstStepsApplication(component.getContext().createChildContext()));
```

The application is instantiated with a child context, not the component context. Otherwise, your application might not start properly and a log trace will warn you.

Handling of resources

Some new boolean attributes have been added on the resource class. They help to specify the state of a Resource instance.

- The "available" attribute says if the current resource exists and can present a representation. If a resource is not available, then a 404 status is returned.
- The "modifiable" attribute says if the current resource supports the methods that update its state, that is to say, for the HTTP protocol: POST, PUT and DELETE.
- The "readable" attribute says if the current resource is able to generate a representation. If a resource is not readable, a resource will answer to GET and HEAD HTTP methods with a "method not allowed" response status.

Usage of the Servlet Adapter

- The ServletConverter now also copy the Servlet's request attributes into the Restlet attributes map.
- A static ServletCall.getRequest(Request) method has been added to the Servlet extension and gives access to the HttpServletRequest object.
- The underlying component can now be customized which allows to define several applications either with a "/WEB-INF/restlet.xml file", or a "org.restlet.component" parameter in the "web.xml". see ServerServlet javadocs for more details.

Accessing current objects

Some recurrent need is to access current objects such as the current application, the current context, etc. Although, the API of some objects

gives direct access to such properties (e.g. "Resource#getApplication"), it was generally not the case with Restlet 1.0, thus some static methods have been introduced in Restlet 1.1:

Method name

Response.getCurrent()

Application.getCurrent()

Context.getCurrent()

Context.getCurrentLogger()

These methods may help you to reduce the number of lines of code but, for proper object-oriented design, we recommend using them only under duress. Typical case is when you need to integrate Restlet code with a third-party library that doesn't let you pass in your Restlet context or objects. For example, you should by default prefer obtaining the current context using methods such as Restlet.getContext() or Handler.getContext().

Accessing the original resource's reference

New features have been added to the tunnel filter. Some of them allow you to automatically update the request according to some parameters specified in the query part of the target resource's reference. In this case, it happens that the resource's reference is updated. If you still want to access the original reference, a new attribute has been added to the Request object called "originalRef'".

Handling of statuses and exceptions

A new "ResourceException" class has been introduced. Basically, it encapsulates a status and the optional cause of a checked exception. This exception may be thrown by a Resource instance when handling GET, POST, PUT, etc requests. This exception can be handled by the status service when rendering error statuses.

Miscellaneous

This section lists several updates that may have an impact on your existing code

- The Resource class now accepts POST requests without any entity, or with an empty entity.
- The list of known media-types defined on the MetadataService has been completed with Tomcat's entries

- Added a Representation.release() to have an uniform way to release a representation without forcing a read for example or manually closing the socket, the channel, the file, etc.
- Added a Representation.exhaust() method that reads and discards content optimally (better than getText())
- Application is now concrete and has a setRoot() method.
- Filter.beforeHandle() and doHandle() methods can now indicate if the processing should continue normal, go to the afterHandle() method directly or stop immediately. **IMPORTANT NOTE**: as it isn't possible to add a return parameter to an existing method, the change can break existing filters. In this case, you just need to return the "CONTINUE" constant from your method and use "int" as a return parameter.
- Added Handler.getQuery() method to easily return the request's target resource reference query as a parsed form (series of parameters).
- The Message's getEntityAsDom(), getEntityAsSax() and getEntityAsForm() are now caching the result representation for easier reuse in a Restlet filters chain.

Deprecations

Resource class

List of renamed methods to prevent confusion with lower-level methods handleGet(), handletPost(), handlePut() and handleDelete() now part of the parent class of Resource, the Handler class.

Method

getPreferredRepresentation()

getRepresentation(Variant)

post(Representation)

put(Representation)

delete()

Variant class

Some properties and methods have been moved to the Representation subclass

- UNKNOWN_SIZE

- getExpirationDate, setExpirationDate
- getModificationDate, setModificationDate
- getSize, setSize
- getTag, setTag

TunnelService
List of renamed methods.

Method

getCharacterSetAttribute

setCharacterSetAttribute

getEncodingAttribute

setEncodingAttribute

getLanguageAttribute

setLanguageAttribute

getMediaTypeAttribute

setMediaTypeAttribute

MetadataService
This class allows to link a metadata with several extensions name. However, only the first one in the list will be returned by the getExtension(Metadata), it is considered as the preferred one. The "getMappings" and "setMappings" methods have been removed. Instead, use getExtension(Metadata), getMetadata(String) and the addExtension(String, Metadata), addExtension(String, Metadata, boolean preferred), clearExtension methods

ConverterService
Since 1.1 with no replacement as it doesn't fit well with content negotiation. Most users prefer to handle those conversion in Resource subclasses.

Request
As a consequence of the previous change, the following methods are not replaced: Request#getEntityAsObject, Request#setEntity(Object)

Response
List of renamed methods.

Method

getChallengeRequest

getRedirectRef

setRedirectRef

Finder
Method

createResource(Request, Response)

Template
Removed the Logger parameter to all constructors. which still can be set with the setLogger(Logger) method.

Restlet
The init(Request, Response) method is removed. Instead, make sure that you call the {@link #handle(Request, Response)} method from your Restlet superclass.

Form
Removed the Logger parameter to all constructors.

Status
Method

isInfo(int code)

isInfo()

ChallengeScheme
ChallengeScheme HTTP_AWS replaced by HTTP_AWS_S3

Protocol
The SMTP_STARTTLS protocol is removed. Use the "startTls" parameter on the JavaMail connector instead.

Response
Method

getChallengeRequest

getRedirectRef

setRedirectRef

Guard
List of renamed methods.

Method

challenge(Response)

checkSecret(String, char)

Context
List of renamed method.

Method

getDispatcher

TransformRepresentation
List of renamed methods

Method

getURIResolver

What's new in version 2.1

Introduction
In the next sections, you will get a synthesis of the major changes done to the Restlet Framework in version 2.1.

For more details, you can read the 2.1 announce on our blog as well as the full list of changes.

Better documentation

Restlet in Action book

Finished writing the Restlet in Action book, published by Manning. We made sure that the printed version was available at the same time as the 2.1.0 release.

Scalable internal connector

Non blocking NIO

In version 2.0, we have added support for asynchronous processing of calls as a preview feature, including provisional responses (1xx status code in HTTP). This feature was only usable with the internal HTTP connector that is part of the Restlet engine (org.restlet.jar file), relying on message queues to support asynchronous handling. However, its IO

processing was still done in a blocking manner, requiring two threads per connection which limits its scalability, even if persistent connections and pipelining and now supported.

There, we have started work on a new NIO version of this internal connector that leverages the non-blocking features of NIO to support a large number of concurrent connections and messages with only a single IO thread! The first results were very promising and we want to complete this connector in version 2.1 to replace the current internal connector with a lighter, faster and more scalable one.

SIP connector

SIP is a core protocol for Voice of IP (VoIP) to control multimedia session. It has been designed based on the HTTP protocol, using the same syntax for request and messages and with a similar processing flow, leveraging a lot provisional responses. We explored over the past months the possibility to provide a SIP connector based on the same internal connector that we use for HTTP and already we have a prototype working in the Restlet Incubator.

For version 2.1, we want to complete this initial work and make sure its works and scales properly on top of the new NIO based connector mentioned above. This will ship as an org.restlet.ext.sip extension. We also plan to explore a higher-level SIP application API that would provide a REST-minded alternative to the SIP Servlets.

Security enhancements

Google SDC connector

This protocol allows tunnelling HTTP calls from a public cloud such as GAE, AWS or any other IaaS, to an intranet protected by a firewall, without requiring changes to this firewall. See details in this blog post.

Google App Engine extension

This extension integrates GAE's authentication service with Restlet's security API.

OAuth 2.0 and OpenID 2.0 extensions

See user guide for details.

CookieAuthenticator

Added to the Crypto extension.

Multipart HTML forms

The support for composite representations such as multi-part forms, on both the client and the server side is a recurrent need expressed by users, we should address it directly in the API or via an extension.

Currently, users rely on the FileUpload extension on the server-side and on alternative HTTP clients such as Apache HTTP Client 3.1 on the client-side. It would be much better to have a built-in consistent solution.

There will also be a focus on facilitating the validation of form submissions.

Conneg service

Content negotiation has always been a strong feature of the Restlet Framework. In version 2.0, its implementation and negotiation algorithm is fixed in the Restlet Engine. In this new version, we want to make it customizable via a new ConnegService.

Eclipse integration

Eclipse is more than an IDE and now provides a comprehensive runtime platform via the Eclipse RT project. In addition, OSGi and model-driven technologies developed by the Eclipse foundation nicely fit with the Restlet Framework. In version 2.1, we will help developers to bridge the Restlet and the Eclipse worlds.

Model-driven REST

Based on our experience with customers, we believe that the combination of Restlet and the pragmatic model-driven technologies developed by the Eclipse Modeling project adds a lot of value. In version 2.1, we will add those new related extensions:

- EMF extension: to convert EMF representation beans into XML, XMI or HTML representations

Integration with Equinox/OSGi

We continued to mentor/support the ongoing project for integrating Restlet with Eclipse Equinox (OSGi runtime) which is now part of Restlet Incubator.

Eclipse update site

In addition to our Maven repository, Zip archives and Windows installers, we want to add a possibility to install and update Restlet modules and dependent libraries via the Eclipse IDE directly. For this we will provide an update site. Restlet modules are already OSGi bundles so this should be straightforward.

Migration guide from version 2.0 to 2.1

This section intends to explain the main differences between the Restlet 2.0 and 2.1 releases and to help you migrate your existing applications. Both releases are meant to be compatible at the API level, so you should at most observe deprecate features while upgrading.

Note that if you intend to migrate directly from 1.1 to 2.1, you should really consider migrating first from 1.1 to 2.0 and then from 2.0 to 2.1. For migration instructions between 1.1 and 2.0, you can check this page.

Replace all JAR files

Restlet JARs and dependencies

Deprecated API features

The next step is to look at each deprecated feature and look in the Javadocs at the preferred alternative in version 2.1. # Presentation layer

When compared to the Servlet API, the Restlet API doesn't have a sister API like Java Server Pages (JSP). Instead we made the design choice to be equally open to all presentation technologies. This openess is materialized in the Representation class which is used for response entities.

More concretely, we provide integrations with three popular template technologies : XSLT, FreeMarker and Apache Velocity. In addition, integration with Facelets has been achieved by a third party and it should be very easy to support any other reusable template technology.

The design idea of those extensions is to use a TemplateRepresentation that combines at generation time a data model with a template document.

Persistence layer

The Restlet framework is completely agnostic regarding the persistence technology that you want to use. Many alternatives having been used successfully and we are confident that you won't hit any special limitation in this area.

The basic idea is that from a Restlet point of view, your application with be composed of resources, extending the org.restlet.resource.Resource class. Those subclassed will be in charge of handling the incoming requests. One instance of your resource subclass will be created for each request to handle, making sure that you don't have to care about concurrent access at this point of your application.

When you resource is instantiated, it will need to expose its representations (via HEAD, GET methods), to store (PUT method), accept (POST method) or remove (DELETE method) representations. During construction, based on the actual identity of your resource and other parameters or attributes of the request, you will be able to contact your persistence backend in order to support your processing logic or the representations of your resources returned.

Architecture

Introduction

The Restlet Framework is composed of two main parts. First, there is the Restlet API, a neutral API supporting the concepts of REST and HTTP, facilitating the handling of calls for both client-side and server-side applications. This API is backed by the Restlet Engine and both are now shipped in a single JAR ("org.restlet.jar").

This separation between the API and the implementation is similar to the one between the Servlet API and Web containers like Jetty or Tomcat, or between the JDBC API and concrete JDBC drivers.

Overview of a REST architecture

Let's step back a little and consider typical web architectures from a REST point of view. In the diagram below, ports represent the connector that enables the communication between components which are represented by the larger boxes. The links represents the particular protocol (HTTP, SMTP, etc.) used for the actual communication.

Note that the same component can have any number of client and server connectors attached to it. Web Server B, for example, has both a server connector to respond to requests from the User Agent component, and client connectors to send requests to Web Server A and the Mail Server.

Overview of a Restlet architecture

In addition to supporting the standard REST software architecture elements as presented before, the Restlet Framework also provides a set of classes that greatly simplify the hosting of multiple applications within a single JVM. The goal is to provide a RESTful, portable and more flexible alternative to the existing Servlet API. In the diagram below, we can see the three types of Restlets that are provided in order to

manage these complex cases. Components can manage several Virtual Hosts and Applications.

Virtual Hosts support flexible configuration where, for example, the same IP address is shared by several domain names, or where the same domain name is load-balanced across several IP addresses. Finally, we use Applications to manage a set of related Restlets, Resources and Representations. In addition, Applications are ensured to be portable and reconfigurable over different Restlet implementations and with different virtual hosts. In addition, they provide important services like access logging, automatic decoding of request entities, configurable status page setting and more!

In order to illustrate these classes, let's examine a simple example. Here we create a Component, then add an HTTP server connector to it, listening on port 8182. Then we create a simple trace Restlet and attach it to the defaut VirtualHost of the Component. This default host is catching any request that wasn't already routed to a declared VirtualHost (see the Component.hosts property for details). In a later example, we will also introduce the usage of the Application class. Note that for now you don't see any access log displayed in the console.

```
public static void main(String[] args) throws Exception {
    // Create a new Restlet component and add a HTTP server connector to it
    Component component = new Component();
    component.getServers().add(Protocol.HTTP, 8182);
    // Then attach it to the local host
    component.getDefaultHost().attach("/trace", Part05.class);
    // Now, let's start the component!
    // Note that the HTTP server connector is also automatically started.
    component.start();
}
@Get
public String toString() {
    // Print the requested URI path
```

```
    return "Resource URI  : " + getReference() + '\n' + "Root URI      : "
            + getRootRef() + '\n' + "Routed part   : "
            + getReference().getBaseRef() + '\n' + "Remaining part: "
            + getReference().getRemainingPart();
}
```

Now let's test it by entering http://localhost:8182/trace/abc/def?param=123 in a Web browser. Here is the result that you will get:

```
Resource URI  : http://localhost:8182/trace/abc/def?param=123
Root URI      : http://localhost:8182/trace
Routed part   : http://localhost:8182/trace
Remaining part: /abc/def?param=123
```

Getting started with Maven

Introduction

Maven is a comprehensive project management system built around the concept of POM (Project Object Model). One of the main advantages is the automated handling of project dependencies, including their download. For more information on Maven, check the project home page.

The Maven support appeared to be important for many Restlet users. The initial response was to automatically generate the POM files for each module JAR shipped within the Restlet distribution. This enabled users to upload those JAR files to a local Maven repository, using a script like the one available in the Wiki.

But, this was clearly not easy enough and forced users to download the full distribution for each new version released, instead of just updating a couple of JARs. There also was issues with some third-party dependencies which aren't available in public Maven repositories, like the db4o, AsyncWeb or Simple.

That's why it has been decided to launch our dedicated Maven repository. It is freely accessible from http://maven.restlet.org and contains all Restlet JARs and third party dependencies that aren't available in the main public Maven repository.

Public repository configuration

Here are some instructions about how to configure Maven client to work with the online Maven repository.

You should have Maven installed.

- Go to Maven download page
- Download the latest version of Maven and install it on your local computer
- Add Maven bin folder to your PATH

Declare the repository for your project or for a parent project by updating the *pom.xml* file and adding the following code to the <repositories> section:

```
<repository>
  <id>maven-restlet</id>
  <name>Public online Restlet repository</name>
  <url>http://maven.restlet.org</url>
</repository>
```

As an alternative, you can also declare the repository for all of your projects. Go to the directory on the local computer where you just install Maven. Open and edit *conf/settings.xml* file. Add to the <profiles> section the following code:

```
<profile>
 <id>restlet</id>
 <repositories>
  <repository>
   <id>maven-restlet</id>
   <name>Public online Restlet repository</name>
   <url>http://maven.restlet.org</url>
  </repository>
 </repositories>
</profile>
```

Just after the </profiles> add the following:

```
<activeProfiles>
 <activeProfile>restlet</activeProfile>
```

</activeProfiles>

Available artifacts

The following table lists the available artifacts and their group and artifact ids. With the introduction of the editions for the Restlet framework, it is necessary to make a distinction between an extension for a given edition and the same extension for another extension simply because the code of the extension may change between each edition. This distinction is reflected in the group id of each artifacts which contains a reference to an edition.They are all set on the same pattern: "org.restlet.<edition>" where "<edition>" is three-letters code of an edition among:

- jse (Java SE edition)
- jee (Java EE edition),
- gae (Google App Engine edition),
- android (Android edition)
- gwt (Google Web Toolkit edition),
- osgi (OSGi Environments edition).

You can find here a full view of the list of extensions and the editions that ship them.

artifactId

org.restlet

org.restlet.ext.atom

org.restlet.ext.crypto

org.restlet.ext.e4

org.restlet.ext.emf

org.restlet.ext.fileupload

org.restlet.ext.freemarker

org.restlet.ext.gae

org.restlet.ext.gson

org.restlet.ext.gwt

org.restlet.ext.html

org.restlet.ext.httpclient

org.restlet.ext.jaas

org.restlet.ext.jackson

org.restlet.ext.javamail

org.restlet.ext.jaxb

org.restlet.ext.jaxrs

org.restlet.ext.jdbc

org.restlet.ext.jetty

org.restlet.ext.jibx

org.restlet.ext.json

org.restlet.ext.jsslutils

org.restlet.ext.lucene

org.restlet.ext.nio

org.restlet.ext.oauth

org.restlet.ext.odata

org.restlet.ext.openid

org.restlet.ext.osgi

org.restlet.ext.rdf

org.restlet.ext.rome

org.restlet.ext.sdc

org.restlet.ext.servlet

org.restlet.ext.simple

org.restlet.ext.sip

org.restlet.ext.slf4j

org.restlet.ext.spring

org.restlet.ext.swagger

org.restlet.ext.thymeleaf

org.restlet.ext.velocity

org.restlet.ext.wadl

org.restlet.ext.xdb

org.restlet.ext.xml

org.restlet.ext.xstream

org.restlet.test

Sample dependencies declaration

Each project based on the Restlet Framework needs to declare at least one dependency: the Restlet core module. According to your needs, you should complete the list of dependencies with the required extensions and connectors. For example, assuming your project is a Web server delivering static files, you need one HTTP server connector such as Simple. Since your Maven client correctly references the Restlet online repository, just open and edit the *pom.xml* file for your project and add the following lines of text into the <dependencies> section.

```
<dependency>
  <groupId>org.restlet.jse</groupId>
  <artifactId>org.restlet</artifactId>
  <version>2.2-RC4</version>
</dependency>
<dependency>
  <groupId>org.restlet.jse</groupId>
  <artifactId>org.restlet.ext.simple</artifactId>
  <version>2.2-RC4</version>
</dependency>
```

Getting started with Eclipse IDE

Introduction

There are two ways to use Restlet within Eclipse IDE. The first one is to create a Java project and use the Restlet JARs as external dependencies. This is very simple and works well in most cases.

The second way is to install Restlet JARs as Eclipse/OSGi bundles. All Restlet JARs including dependencies are valid OSGi bundles, so this is very convenient if you work in an Eclipse plug-in environment, such as an Eclipse RCP application. You then just need to create a plug-in project.

Getting Started with Maven and Spring

Overview

This document outlines how to integrate the Restlet Framework with Maven and Spring. It is not a tutorial on using the Restlet Framework.

Prerequisites

The reader should be familiar with Maven, Spring and the Restlet Framework since it deals exclusively with integration issues. To play along you will need to have a version of Maven installed on your environment. The code in this document has been tested against Maven 2.2.1.

The Steps

Step 1: Create a Maven Project

We can use the Maven '**archetype**' goal to quickly create a basic java project structure. We define the name of the artifact and the group (i.e. namespace) and the plugin will create a set of directories and some skeleton java source files. We will delete the App and AppTest source files since we will be creating our own classes.

```
 mvn archetype:create -DgroupId=com.mycompany.basecamp
-DartifactId=restlet-basecamp
```

```
restlet-basecamp/

restlet-basecamp/pom.xml

restlet-basecamp/src

restlet-basecamp/src/main

restlet-basecamp/src/main/java

restlet-basecamp/src/main/java/com

restlet-basecamp/src/main/java/com/mycompany

restlet-basecamp/src/main/java/com/mycompany/restlet

restlet-basecamp/src/main/java/com/mycompany/restlet/basecamp
```

restlet-basecamp/src/main/java/com/mycompany/restlet/basecamp/App.java

restlet-basecamp/src/test

restlet-basecamp/src/test/java

restlet-basecamp/src/test/java/com

restlet-basecamp/src/test/java/com/mycompany

restlet-basecamp/src/test/java/com/mycompany/restlet

restlet-basecamp/src/test/java/com/mycompany/restlet/basecamp

restlet-basecamp/src/test/java/com/mycompany/restlet/basecamp/AppTest.java

Step 2: Configure the POM

The pom.xml file generate from the previous step was for a 'jar' project. We are creating a 'war' project so we will make significant changes. The pom defines its dependency on some Restlet components, in particular the Spring and Servlet extension packages. It also defines the Jetty plugin since we will be running this web service in an embedded Jetty server.

```
<?xml version="1.0" encoding="UTF-8"?>

<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/maven-v4_0_0.xsd">


  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.restlet.basecamp</groupId>

  <artifactId>restlet-basecamp</artifactId>

  <packaging>war</packaging>

  <version>1.5</version>

  <name>Bootstrapping Restlet Project</name>


  <repositories>

    <repository>
```

```
    <id>restlet</id>
    <url>http://maven.restlet.org/</url>
  </repository>
</repositories>

<dependencies>
  <dependency>
    <groupId>org.restlet.jee</groupId>
    <artifactId>org.restlet</artifactId>
    <version>2.0.1</version>
  </dependency>
  <dependency>
    <groupId>org.restlet.jee</groupId>
    <artifactId>org.restlet.ext.servlet</artifactId>
    <version>2.0.1</version>
  </dependency>
  <dependency>
    <groupId>org.restlet.jee</groupId>
    <artifactId>org.restlet.ext.spring</artifactId>
    <version>2.0.1</version>
  </dependency>
</dependencies>

<build>
  <finalName>basecamp</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
```

```
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.25</version>
      <configuration>
        <contextPath>${basecamp.server.contextpath}</contextPath>
        <scanIntervalSeconds>10</scanIntervalSeconds>
        <webXml>${project.build.directory}/$
{project.build.finalName}/WEB-INF/web.xml</webXml>
      </configuration>
    </plugin>
  </plugins>
 </build>

 <properties>

<basecamp.server.contextpath>basecamp</basecamp.server.contextpath>
 </properties>
</project>
```

Step 3: Create the BaseCampResource

We will create the simplest of resources, called BaseCampResource, which extends ServerResource and responds to the HTTP GET method. Note the use of annotations, which is part of the Restlet Framework.

For the purpose of this document we will only define this simple resource.

```
package com.mycompany.restlet.basecamp.resource.demo;


import org.restlet.resource.Get;

import org.restlet.resource.ServerResource;


public class BaseCampResource extends ServerResource {


  @Get

  public String getResource()  {

    return "Hello World!";

  }

}
```

Step 4: Create the BaseCampApplication

In this step we define our Restlet Application, namely BaseCampApplication, which extends the core framework class. It's not really required for this example but if you need to override base class behaviour this is how you would go about it.

```
package com.mycompany.restlet.basecamp.application;


import org.restlet.Application;


public class BaseCampApplication extends Application {

}
```

Step 5: Sprinkle Some Spring

The application context, is used by Spring, to create and start the various components.

- **basecampComponent** is the Spring wrapper, which is a core framework class. This class references our application

- **basecampApplication** points to our application, which was developed in Step 4. It also reference the router, which determines how URLs are mapped to resource classes.
- **router** is the default router, which is also part of the core framework
- **'/hello'** is a reference to our resource class. Any bean name that starts with a forward slash ('/') is assumed to be a route to a resource and as such is registered with the router. In this example we only define one resource and one route.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

  <bean id="basecampComponent"
class="org.restlet.ext.spring.SpringComponent">

    <property name="defaultTarget" ref="basecampAppliction" />

  </bean>



  <bean id="basecampAppliction"
class="com.mycompany.restlet.basecamp.application.BaseCampApplication"
>

    <property name="root" ref="router" />

  </bean>



  <!--  Define the router -->

  <bean name="router" class="org.restlet.ext.spring.SpringBeanRouter" />



  <!-- Define all the routes -->

  <bean name="/hello"
class="com.mycompany.restlet.basecamp.resource.demo.BaseCampResourc
e" scope="prototype" autowire="byName" />

</beans>
```

Step 6: Set up the web.xml

Finally, we need to configure the web.xml, which is packaged in the war. The important parts are the **context-param** entries. One points to the the SpringComponent class instance in our application context and the other points to the location of the application context file.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app id="restlet-basecamp" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Restlet Basecamp</display-name>



  <listener>


<listener-class>org.springframework.web.context.ContextLoaderListener</lis
tener-class>

  </listener>



  <context-param>

    <param-name>org.restlet.component</param-name>

    <param-value>basecampComponent</param-value>

  </context-param>

  <context-param>

    <param-name>contextConfigLocation</param-name>

    <param-value>WEB-INF/applicationContext.xml</param-value>

  </context-param>



  <servlet>

    <servlet-name>basecamp</servlet-name>

    <servlet-class>org.restlet.ext.spring.SpringServerServlet</servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>basecamp</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Step 7: Build It

To create the war file execute '**mvn package**' in your shell. The war file will in the **target** subdirectory.

```
jima:restlet-basecamp jima$ mvn package

[INFO] ------------------------------------------------------------------------
[INFO] Building Bootstrapping Restlet Project
[INFO]    task-segment: [package]
[INFO] ------------------------------------------------------------------------

[INFO] [resources:resources {execution: default-resources}]
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test {execution: default-test}]
```

[INFO] Surefire report directory:
/Users/jima/projects/restlet-basecamp/restlet-basecamp/target/surefire-report
s

 T E S T S

There are no tests to run.

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO] [war:war {execution: default-war}]

[INFO] Packaging webapp

[INFO] Assembling webapp[restlet-basecamp] in
[/Users/jima/projects/restlet-basecamp/restlet-basecamp/target/summer]

[INFO] Dependency[Dependency {groupId=org.restlet.jee,
artifactId=org.restlet, version=2.0.1, type=jar}] has changed (was
Dependency {groupId=org.restlet.jee, artifactId=org.restlet, version=2.0.1,
type=jar}).

[INFO] Dependency[Dependency {groupId=org.restlet.jee,
artifactId=org.restlet.ext.servlet, version=2.0.1, type=jar}] has changed
(was Dependency {groupId=org.restlet.jee, artifactId=org.restlet.ext.servlet,
version=2.0.1, type=jar}).

[INFO] Dependency[Dependency {groupId=org.restlet.jee,
artifactId=org.restlet.ext.spring, version=2.0.1, type=jar}] has changed (was
Dependency {groupId=org.restlet.jee, artifactId=org.restlet.ext.spring,
version=2.0.1, type=jar}).

[INFO] Processing war project

[INFO] Copying webapp
resources[/Users/jima/projects/restlet-basecamp/restlet-basecamp/src/main/w
ebapp]

[INFO] Webapp assembled in[112 msecs]

[INFO] Building war: /Users/jima/projects/restlet-basecamp/restlet-basecamp/target/summer.war

[INFO] --

[INFO] BUILD SUCCESSFUL

[INFO] --

[INFO] Total time: 4 seconds

[INFO] Finished at: Tue Nov 09 21:30:38 EST 2010

[INFO] Final Memory: 18M/81M

[INFO] --

Step 8: Run It

To run the web service in an embedded jetty server simple execute "**mvn jetty:run-war**" this will compile and package the war file and then run it up inside a jetty servlet container. The web service will be available on **localhost:8080** with a root context of '**/basecamp**'

jima:restlet-basecamp jima$ mvn jetty:run-war

[INFO] [jetty:run-war {execution: default-cli}]

[INFO] Configuring Jetty for project: Bootstrapping Restlet Project

[INFO] Context path = /basecamp

[INFO] Tmp directory = determined at runtime

[INFO] Web defaults = org/mortbay/jetty/webapp/webdefault.xml

[INFO] Web overrides = none

[INFO] Starting jetty 6.1.25 ...

[INFO] Started Jetty Server

[INFO] Starting scanner at interval of 10 seconds.

If you now go to your web browser and enter the following URL the service, which you have just created, should respond with 'Hello World!'

Web browser screenshot

Web browser screenshot

Accessing the resource via a browser

Finally, if you go back to the console, where you started the server you should see some logging similar to what is displayed below

Nov 9, 2010 9:34:31 PM org.restlet.engine.log.LogFilter afterHandle

INFO: 2010-11-09 21:34:31 0:0:0:0:0:0:0:1%0 - 0:0:0:0:0:0:0:1%0 8080 GET /basecamp/hello - 200 12 0 57 http://localhost:8080 Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_4; en-us) AppleWebKit/533.18.1 (KHTML, like Gecko) Version/5.0.2 Safari/533.18.5 -

Step 9: Deploy It

You can also deploy the web service to a standalone web server such as Apache Tomcat using the Tomcat Maven Plugin. Basically, you need to define the plugin in your pom.xml and call '**mvn tomcat:deploy**' and '**mvn tomcat:undeploy**' to deploy and undeploy the web service respectively.

An minimal configured tomcat maven plugin is shown below but you should consult the documentation for the finer details.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>tomcat-maven-plugin</artifactId>
  <configuration>
    <server>${tomcat.server}</server>
    <url>${tomcat.server.manager}</url>
    <warFile>${project.build.directory}/${project.build.finalName}.war</warFile>
    <path>${basecamp.server.contextpath}</path>
  </configuration>
</plugin>
```

Resources

This archive contains all the source code described in this document.

restlet-basecamp (application/zip, 8.4 kB)

Debugging tools

Introduction

As a set of pure Java library, the Restlet framework can easily be debugged in your favorite IDE. All the source code is available, making

debugging session even easier by going inside the Restlet code if necessary.

A good way to start a debugging session is to put a breakpoint inside the handle() method or inside the constructor of your Resource subclass. Think also about turning one the access and application loggings.

Tools

Regarding protocol debugging, we recommend the you install tools such as:

- cURL : command line HTTP client for Unix
- WireShark : advanced network analyzer working at the IP or TCP or HTTP levels
- RESTClient : Java/Swing graphical HTTP client
- Poster : FireFox extension to test RESTful Web applications
- Netcat : swiss-army knife for TCP/IP.
- tcpmon : Java utility that monitors a TCP connection.
- etc.

Getting started

This section explains how to get started with the Restlet Framework, including set up and debugging tools in common development environments such as:

- Eclipse IDE
- Maven
- Maven and Spring
- Debugging tools

Finally, there is the Restlet in Action book written by Restlet creators and published by Manning that is a highly recommended reading for a smoother Restlet learning curve. Groovy integration
==========

As a Java library

As Groovy works with any Java library, it can naturally leverage the Restlet framework. For a detailled article explaining this usage, I recommand this post from Arc90:

"Building RESTful Web Apps with Groovy and Restlet"

- Part 1: Up and Running
- Part 2: Resources

As a Domain Specific Language

Another strength of Groovy is its capacity to define new languages in a very dynamic and flexible way. Qi Keke has develop a specific DSL for Restlet in Groovy. It is now available as an official Groovy Module. For more information, check the announce on Restlet's blog.

The Kauri project, based on Restlet, is also using its own Groovy DSL to configure the routing.

Restlet.org example

Table of contents

Introduction

The Restlet web site that you are currently navigating is powered by the Noelios Restlet Engine, the reference implementation of the Restlet API. In order to serve HTTP calls, we rely on a production-ready Simple 3.1 HTTP connector. As you will see below, running basic web sites with Restlets is very simple.

This page needs to be updated to use the new Restlet 2.3 API

Imported classes

First, let's declare the imported classes required to support our web component.

```
import org.restlet.Component;

import org.restlet.VirtualHost;

import org.restlet.data.Protocol;
```

Declaring the Main class

Now, we declare the main class, called the WebComponent, extending the org.restlet.Component. This component contains several virtual hosts and associated applications.

```
/**
 * The web component managing the Restlet web servers.
 *
 * @author Jerome Louvel (contact@restlet.com)
 */
public class WebComponent extends Component {
      ...
```

Main method

Below, you have the main method that is invoked by our startup scripts. Note that we require a few arguments in order to parameterize several aspects like IP address and port to listen on, or the location of static files.

```
/**
* Main method.
*
* @param args
*            Program arguments.
*/
public static void main(String[] args) {
  try {
    if ((args == null) || (args.length != 7)) {
       // Display program arguments
       System.err
             .println("Can't launch the web server. List of "
               + "required arguments:\n"
               + " 1) IP address to listen on\n"
```

```
                + " 2) Port to listen on\n"
                + " 3) File URI to the \"www\" directory location.\n"
                + " 4) File URI to the \"data\" directory location.\n"
                + " 5) Search redirect URI template."
                + " 6) Login for protected pages."
                + " 7) Password for protected pages.");
        } else {
            // Create and start the server
            new WebComponent(args[0], Integer.parseInt(args[1]), args[2],
                    args[3], args[4], args[5], args[6]).start();
        }
    } catch (Exception e) {
        System.err
                .println("Can't launch the web server.\nAn unexpected "
                + "exception occured:");
        e.printStackTrace(System.err);
    }
}
```

Build the component

Now we need to build the component containing our web applications. As we are handling several domain names (www.noelios.com, restlet.org, search.restlet.org, etc.) via the same HTTP server connector (with a single IP address and port open), we also need to declare several virtual hosts.

```
/**
* Constructor.
*
* @param ipAddress
*            IP address to listen on.
```

```
* @param port
*            Port to listen on.
* @param wwwUri
*            File URI to the "www" directory location.
* @param dataUri
*            File URI to the "data" directory location.
* @param redirectUri
*            The search redirect URI template.
* @param login
*            Login for protected pages.
* @param password
*            Password for protected pages.
*/
public WebComponent(String ipAddress, int port, String wwwUri,
      String dataUri, String redirectUri, String login, String password)
      throws Exception {
   getLogService().setLoggerName("com.noelios.web.WebComponent.www");

   // ------------------
   // Add the connectors
   // ------------------
   getServers().add(Protocol.HTTP, ipAddress, port);
   getClients().add(Protocol.FILE);

   // ---------------
   // restlet.org
   // ---------------
   VirtualHost host = new VirtualHost(getContext());
```

```
host.setHostDomain("restlet.org|81.67.81.67");
host.setHostPort("80|" + Integer.toString(port));
host.attach(new WwwRestletOrg(getContext(), dataUri, wwwUri
        + "/www-restlet-org"));
getHosts().add(host);

// --------------------------
// Redirect to restlet.org
// --------------------------
host = new VirtualHost(getContext());
host.setHostDomain("restlet.org|restlet.net|restlet.com|"
        + "www.restlet.net|www.restlet.com");
host.setHostPort("80|" + Integer.toString(port));
host.attach(new RedirectApplication(getContext(),
        "http://restlet.org{rr}", true));
getHosts().add(host);

// ----------------
// wiki.restlet.org
// ----------------
host = new VirtualHost(getContext());
host.setHostDomain("wiki.restlet.org");
host.setHostPort("80|" + Integer.toString(port));
host.attach("/", new RedirectApplication(getContext(),
        "http://wiki.java.net/bin/view/Javawsxml/Restlet{rr}", false));
getHosts().add(host);

// ------------------
```

```
// search.restlet.org
// ------------------
host = new VirtualHost(getContext());
host.setHostDomain("search.restlet.org|localhost");
host.setHostPort("80|" + Integer.toString(port));
host.attach(new SearchRestletOrg(getContext(), wwwUri
        + "/search-restlet-org", redirectUri));
getHosts().add(host);

// ---------------
// www.restlet.net
// ---------------
host = new VirtualHost(getContext());
host.setHostDomain("www.restlet.net");
host.setHostPort("80|" + Integer.toString(port));
host.attach(new RedirectApplication(getContext(),
        "http://restlet.tigris.org{rr}", false));
host.attach("/fisheye/", new RedirectApplication(getContext(),
        "http://fisheye3.cenqua.com/browse/restlet/{rr}", false));
getHosts().add(host);

// ---------------
// www.noelios.com
// ---------------
host = new VirtualHost(getContext());
host.setHostDomain("www.noelios.com");
host.setHostPort("80|" + Integer.toString(port));
host.attach(new WwwNoeliosCom(getContext(), dataUri, wwwUri
```

```
        + "/www-noelios-com", login, password));
    getHosts().add(host);

    // --------------------------
    // Redirect to www.noelios.com
    // --------------------------
    host = new VirtualHost(getContext());
    host.setHostDomain("noelios.com|noelios.net|noelios.org|"
        + "www.noelios.net|www.noelios.org");
    host.setHostPort("80|" + Integer.toString(port));
    host.attach(new RedirectApplication(getContext(),
        "http://www.noelios.com{rr}", true));
    getHosts().add(host);
}
```

Redirection Application

In addition to the main WebComponent class, we also rely on four application classes. Let's have a look at the RedirectApplication which is generic and reused several times.

```
/**
 * Application redirecting to a target URI.
 *
 * @author Jerome Louvel (contact@restlet.com)
 */
public class RedirectApplication extends Application {
    /** The target URI template. */
    private String targetUri;

    /** Indicates if the redirection is permanent or temporary. */
```

```
private boolean permanent;

/**
 * Constructor.
 *
 * @param parentContext
 *            The parent context. Typically the component's context.
 * @param targetUri
 *            The target URI template.
 * @param permanent
 *            Indicates if the redirection is permanent or temporary.
 */
public RedirectApplication(Context parentContext, String targetUri,
        boolean permanent) {
    super(parentContext);
    this.targetUri = targetUri;
    this.permanent = permanent;
}

@Override
public String getName() {
    return "Redirection application";
}

@Override
public Restlet createRoot() {
    int mode = (this.permanent) ? Redirector.MODE_CLIENT_PERMANENT
            : Redirector.MODE_CLIENT_TEMPORARY;
```

```
        return new Redirector(getContext(), this.targetUri, mode);
    }
}
```

Conclusion

In term of coding, that's about all that we use. In addition, we configure standard JDK logging properties in order to write a web log file based on the applications' log services that write to the "com.noelios.web.WebComponent.www" logger. Finally, we also rely on the Java Service Wrapper tool to execute our component as a Linux daemon.

Thanks to Michael Mayer for the idea of providing the source code of this web site as a sample application.

Prototype.js integration

Introduction

I will start this tutorial with a sample "microblog", that's a text based blog demonstrating the usage of AJAX in Restlet. Before we step in,we should review some knowledge if you never know or forget it:

- What's RESTful?
- What's Restlet?
- How to use Restlet?
- How to use db4o to simplify persistence?
- How to use JSON in Prototype.js?

This example needs to be updated for Restlet Framework 2.1. Help welcome

Demo construction

- Web client: call background service via JSON protocol in RESTful way (GET/PUT/POST/DELETE).
- Server side: uses db4o to work as store service provider, and expose data in RESTful way.
- Server handle process: Application dispatches request to Router, Router finds corresponding reource, Resource handles request and returns representation.

DB4OSimpler.Class

It's very clean from its name that it works as db4o function simpler.Its generalOperate method handles general operation with db4o:

Note:This class works as only non-concurrent model,because it doesn't work as client/server model.If you have requirement,you should modify it in concurrent(client/server) model by using Db4o.openServer method.

```
package com.bjinfotech.util;


import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.query.Query;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.apache.commons.beanutils.*;
/**
 * It's very clean from its name that it works as db4o function simpler.
 * Its generalOperate method handles general operation with db4o.
 * @author cleverpig
 *
 */
public class DB4OSimpler {
    //operation constants that will be used as generalOperate method's
param
    public static final int OPERATION_SAVE=0;
    public static final int OPERATION_LOAD=1;
    public static final int OPERATION_UPDATE=2;
    public static final int OPERATION_DELETE=3;
    public static final int OPERATION_QUERY=4;
    public static final int OPERATION_LIST=5;
    public static final int OPERATION_CLEAR=6;
    /**
     * perform general Operation
     * @param fileName
     * @param op_code corresponding integer type constant
     * @param example object which is needed in operation
```

```
     * @param keyFieldName object's key field name
     * @return
     */
    public static Object generalOperate(
            String fileName,
            int op_code,
            Object example,
            String keyFieldName){
        Object ret=null;
        //open db4o file to get ObjectContainer
        ObjectContainer db=Db4o.openFile(fileName);
        Iterator iter=null;
        List list=null;
        Query query=null;
        try{
            //perform operation according to op_code param value
            switch(op_code){
            case OPERATION_SAVE:
                //just set!It's very simple!
                db.set(example);
                ret=example;
                break;
            case OPERATION_LOAD:
                //just get!
                list=db.get(example);
                if (list!=null && list.size()>0){
                    ret=list.get(0);
                }
                break;
            case OPERATION_UPDATE:
                //at first,I find objects which will be updated with its key field
value
                query=db.query();
                query.constrain(example.getClass());
                query.descend(keyFieldName)
                    .constrain(BeanUtils.getProperty(example, keyFieldName));
```

```
    iter=query.execute().listIterator();
    //and then delete all of them
    while(iter.hasNext()){
        db.delete(iter.next());
    }
    //set new one,now!
    db.set(example);
    ret=example;
    break;
case OPERATION_DELETE:
    //just like update process:find firstly,and then delete them
    query=db.query();
    query.constrain(example.getClass());
    query.descend(keyFieldName)
        .constrain(BeanUtils.getProperty(example, keyFieldName));
    iter=query.execute().listIterator();
    if (iter.hasNext()){
        while(iter.hasNext()){
            db.delete(iter.next());
        }
        ret=true;
    }
    else{
        ret=false;
    }
    break;
case OPERATION_QUERY:
    //just like update process:find firstly,and then return them
    query=db.query();
    query.constrain(example.getClass());
    query.descend(keyFieldName)
        .constrain(BeanUtils.getProperty(example, keyFieldName));
    iter=query.execute().listIterator();
    list=new ArrayList();
    while(iter.hasNext()){
            list.add(iter.next());
```

```
        }
        if (list.size()>0)
            ret=list;
        break;
    case OPERATION_LIST:
        //return list of object which class is example.class.
        list=new ArrayList();
        iter=db.query(example.getClass()).listIterator();
        while(iter.hasNext()){
            list.add(iter.next());
        }
        if (list.size()>0)
            ret=list;
        break;
    case OPERATION_CLEAR:
        //delete anything which class is example.class.
        iter=db.query(example.getClass()).listIterator();
        int deleteCount=0;
        while(iter.hasNext()){
            db.delete(iter.next());
            deleteCount++;
        };
        ret=true;
        break;
    }
    //commit,finally
    db.commit();
}
catch(Exception ex){
    db.rollback();
    ex.printStackTrace();
}
finally{
    db.close();
}
return ret;
```

```
    }
}
```

MicroblogApplication.Class

It's a restful Micoblog Server which serves static files and resource(MicroblogResource) and exposes some services(static html and Microblog):

Note:***I used TunnelService replacing custom finder,'cause I think TunnelService is easy for using.But I'd discuss how to using custom finder to implement same function.***

```
package com.bjinfotech.restlet.practice.demo.microblog;

import org.restlet.Application;
import org.restlet.Component;
import org.restlet.Directory;
import org.restlet.Restlet;
import org.restlet.Router;
import org.restlet.data.Protocol;

/**
 * restful server
 * it serves static files and resource
 * @author cleverpig
 *
 */
public class MicroblogApplication {
    public static void main(String[] argv) throws Exception{
        Component component=new Component();
        //add http protocol
        component.getServers().add(Protocol.HTTP,8182);
        //add file protocol for accessing static web files in some directories
        component.getClients().add(Protocol.FILE);

        Application application=new Application(component.getContext()){
            @Override
            public Restlet createRoot(){
                //directory where static web files live
```

```
                final String
DIR_ROOT_URI="file:///E:/eclipse3.1RC3/workspace/RestletPractice/static_files
/";
                //create router
                Router router=new Router(getContext());
                //attach static web files to "www" folder
                Directory dir=new Directory(getContext(),DIR_ROOT_URI);
                dir.setListingAllowed(true);
                dir.setDeeplyAccessible(true);
                dir.setNegotiateContent(true);
                router.attach("/www/",dir);
                //attach resource class:MicroblogResource to "/restful/blog" as
web service URI
                router.attach("/restful/blog",MicroblogResource.class);
                return router;
            }
        };
        //use TunnelService to simplify request's dispatching
        application.getTunnelService().setEnabled(true);
        application.getTunnelService().setMethodTunnel(true);
        application.getTunnelService().setMethodParameter("method");
        //attach application
        component.getDefaultHost().attach(application);
        component.start();
    }
}
```

microblogAppInterface.js

This is a JavaScript file used in microblog.html file,it call functions which was exposed in server side:

```
var SAVE_MODEL=1;
var UPDATE_MODEL=2;

function switchEditorModel(model){
    switch(model){
        case SAVE_MODEL:
            Element.show('save_button');
            Element.hide('update_button');
```

```
            Element.hide('remove_button');
            Element.hide('new_button');
            break;
        case UPDATE_MODEL:
            Element.hide('save_button');
            Element.show('update_button');
            Element.show('remove_button');
            Element.show('new_button');
            break;
    }
}

...

Event.observe(
    'save_button',
    'click',
    function(){
        var formObj=Form.serialize('edit_form',true);
        var xmlHttp = new Ajax.Request(
            "/restful/blog?method=PUT",
            {
                method: 'post',
                parameters:
'json='+encodeURIComponent(Object.toJSON(formObj)),
                onComplete: function(transport){
                    var retObj=transport.responseText.evalJSON();
                    if (retObj.subject){
                        alert('ok,"'+retObj.subject+'" was saved!');
                        refreshBloglist();
                        switchEditorModel(UPDATE_MODEL);
                    }
                }
            }
        );
    },
    false
```

```
    );

    ...

    function refreshBloglist(){
        var xmlHttp = new Ajax.Request(
            "/restful/blog",
            {
                method: 'get',
                parameters: '',
                onComplete: function(transport){
                    var retObjs=transport.responseText.evalJSON();
                    if (retObjs.length && retObjs.length>0){
                        var listRepr='<ul>\n';
                        retObjs.each(function(obj,index){
                            if (index<retobjs.length-1 ) listrepr+="
    <li>"><a
href="javascript:load(\''+obj.subject+'\');">'+obj.subject+'</a>\n';
                        });
                        listRepr+="<\ul>\n";
                        $('blogList').innerHTML=listRepr;
                    }
                    else{
                        $('blogList').innerHTML='Here is empty';
                    }
                }
            }
        );
    }
    function load(subject){
        var xmlHttp = new Ajax.Request(
                "/restful/blog",
                {
                    method: 'get',
                    parameters: 'subject='+encodeURIComponent(subject),
                    onComplete: function(transport){
                        var retObj=transport.responseText.evalJSON();
```

```
                if (retObj.subject){
                    $('subject').value=retObj.subject;
                    $('content').value=retObj.content;
                    $('tags').value=retObj.tags;
                    alert('ok,"'+retObj.subject+'" was loaded!');
                    switchEditorModel(UPDATE_MODEL);
                }
            }
        }
    );
}
```

MicroblogResource.Class

```
package com.bjinfotech.restlet.practice.demo.microblog;

import java.util.List;
import java.util.logging.Logger;

import org.restlet.Context;
import org.restlet.data.CharacterSet;
import org.restlet.data.Form;
import org.restlet.data.Language;
import org.restlet.data.MediaType;
import org.restlet.data.Request;
import org.restlet.data.Response;
import org.restlet.resource.Representation;
import org.restlet.resource.Resource;
import org.restlet.resource.ResourceException;
import org.restlet.resource.StringRepresentation;
import org.restlet.resource.Variant;
import org.restlet.data.Status;
import com.bjinfotech.util.JSONSimpler;
import com.bjinfotech.util.Utils;

public class MicroblogResource extends Resource {
    Logger
log=Logger.getLogger(MicroblogResource.class.getSimpleName());
```

```
        //MicroblogPersistenceManager
        MicroblogPersistenceManager micoblogPM=new
MicroblogPersistenceManager();
        //StringRepresentation constant
        final StringRepresentation NO_FOUND_REPR=new
StringRepresentation("No found!");
        final StringRepresentation ERR_REPR=new
StringRepresentation("something wrong!");
        //json param name in request
        final String JSON_PARAM="json";

        public MicroblogResource(
                Context context,
                Request request,
                Response response) {
            super(context, request, response);
            this.getVariants().add(new Variant(MediaType.TEXT_PLAIN));
            //it's important,please don't forget it.
            this.setAvailable(true);
            this.setModifiable(true);
            this.setNegotiateContent(true);
        }
        @Override
        /**
         * representing after calling default get handle
         * @param variant
         */
        public Representation represent(Variant variant) throws
ResourceException{
            log.info("representing after calling default get handle...");
            Representation result = null;
            if (variant.getMediaType().equals(MediaType.TEXT_PLAIN)) {
                //find "subject" param in request,"subject" param is tranformed
from web client.it means load one blog with special subject
                String
subject=getRequest().getResourceRef().getQueryAsForm().getFirstValue("subj
ect");
                log.info("subject:"+subject);
```

```
            //handle query
            if (subject!=null && subject.length()>0){
               //find blog with special subject
               Microblog example=new Microblog();
               example.setSubject(subject);
               List queryRet=micoblogPM.query(example);
               //return result in JSON format StringRepresentation
               if (queryRet!=null && queryRet.size()>0){
                  result=new StringRepresentation(
                        JSONSimpler.serializeFromBean(queryRet.get(0)),
                        MediaType.APPLICATION_JSON,
                        Language.ALL,
                        CharacterSet.UTF_8
                     );
               }
               else{
                  result=NO_FOUND_REPR;
               }
            }
            else{
               //return blog list in JSON format StringRepresentation
               result=new StringRepresentation(
                     JSONSimpler.serializeFromBeanList(micoblogPM.list()),
                     MediaType.APPLICATION_JSON,
                     Language.ALL,
                     CharacterSet.UTF_8
                  );
            }
         }
         return result;
      }
      /**
       * call MicroblogPersistenceManager's method excluding query and
list,just save/update/delete.
       * @param method method name
       * @param jsonParamVal json param value coming from request
```

```
     * @return
     */
    protected StringRepresentation callMethod(String method,String
jsonParamVal){
        //transform json format string to Microblog object
        Microblog
microblog=(Microblog)JSONSimpler.deserializeToBean(jsonParamVal,Microblog
.class);
        if (microblog!=null){
            //call method and gain json format string as responseText
            String responseText=Utils.callMethodAndGainResponseJSONStr(
                    micoblogPM,
                    method,
                    jsonParamVal,
                    Microblog.class);
            log.info("response:"+responseText);
            //return json StringRepresentation
            return new StringRepresentation(
                    responseText,
                    MediaType.APPLICATION_JSON,
                    Language.ALL,
                    CharacterSet.UTF_8
            );
        }
        else{
            return ERR_REPR;
        }
    }
    @Override
    /**
     * handling post in high level
     * @param entity
     */
    public void acceptRepresentation(Representation entity) throws
ResourceException{
        log.info("handling post in high level...");
        super.acceptRepresentation(entity);
```

```
            getResponse().setStatus(Status.SUCCESS_OK);
            Form f = new Form(entity);
            String jsonParamVal=f.getValues(JSON_PARAM);
            log.info("json param:"+jsonParamVal);
            //call update and set response
            getResponse().setEntity(callMethod("update",jsonParamVal));
        }
        @Override
        /**
         * handling put in high level
         * @param entity
         */
        public void storeRepresentation(Representation entity) throws
ResourceException{
            log.info("handling put in high level...");
            super.storeRepresentation(entity);
            getResponse().setStatus(Status.SUCCESS_CREATED);
            Form f = new Form(entity);
            String jsonParamVal=f.getValues(JSON_PARAM);
            log.info("json param:"+jsonParamVal);
            //call save and set response
            getResponse().setEntity(callMethod("save",jsonParamVal));
        }
        @Override
        /**
         * handling delete in high level
         * @param entity
         */
        public void removeRepresentations() throws ResourceException{
            log.info("handling delete in high level...");
            super.removeRepresentations();
            getResponse().setStatus(Status.SUCCESS_OK);
            Form f = getRequest().getEntityAsForm();
            String jsonParamVal=f.getValues(JSON_PARAM);
            log.info("json param:"+jsonParamVal);
            //call delete and set response
```

```
            getResponse().setEntity(callMethod("delete",jsonParamVal));
        }
    }
```

Microblog.Class

```
package com.bjinfotech.restlet.practice.demo.microblog;
public class Microblog {
    private String subject;
    private String content;
    private String tags;
   public String getSubject() {
        return subject;
    }
    ...
}
```

Running Application

Running MicroblogApplication,and visit http://localhost:8182/www/microblog.html.

running_applicaion

Click to enlarge

Checkout Full Code

microblog_sourcecode (application/x-zip, 2.3 MB)

How to custom Finder to replace TunnelService

Sure, you can custom a finder to do what tunnelService do.

You can visit http://dobrzanski.net/2007/04/22/using-put-and-delete-methods-in-ajax-requesta-with-prototypejs/ to get detail about how to using restful way in prototype.

A simpler way to do this is to customize the TunnelService. getApplication().getTunnelService().setMethodName("_method"). That's all!

In Application:

```
    ...
    Router router = new Router(getContext());
```

```
    //It's very easy!
    router.setFinderClass(PrototypeFinder.class);
    ...
```

Custom Finder:

```
    public class PrototypeFinder extends Finder {
        public  PrototypeFinder(Context context, Class
    targetClass) {
            super(context, targetClass);
        }

        public void handle(Request request, Response response) {
            //get  "_method" param value
            Parameter p = request.getEntityAsForm().getFirst("_method");
            //reset requst method accoring "_method" param value
            request.setMethod(null != p ? Method.valueOf(p.getValue()) :
    request.getMethod());
            super.handle(request, response);
        }
    }
```

javascript snippet in web page:

```
    ...


    function callJSON() {
        new Ajax.Request('/ajax', {
            parameters: 'name=PUT', method: 'put', putBody: "PUT BODY",
            onComplete: function (transport) {
                alert(transport.responseText);
            }
        });
        new Ajax.Request('/ajax', {
            parameters: 'name=POST', method: 'post',
            onComplete: function (transport) {
                alert(transport.responseText);
            }
        });
```

```
        new Ajax.Request('/ajax', {
                parameters: 'name=DELETE', method: 'delete',
                onComplete: function (transport) {
                        alert(transport.responseText);
                }
        });
}


...
```

Thanks

Evgeny Shepelyuk:this guy gave me a lot of good advice!

Links

- Router
- Application
- Resource
- db4o

Glossary

Table of contents

13 Representation

14 Request

15 Resource

16 Response

17 Restlet

18 Route

19 Router

20 Server

21 Transformer

22 Uniform

23 Virtual Host

Application
Restlet that can be attached to one or more VirtualHosts. Applications are guaranteed to receive calls with their base reference set relatively to the VirtualHost that served them. This class is both a descriptor able to create the root Restlet and the actual Restlet that can be attached to one or more VirtualHost instances.

Authenticator
Filter authenticating the client sending the request based on mechanisms such as challenge request and response or SSL certificates.

Authorizer
Filter authorizing requests based on flexible criterias such as method name, client role.

Client
Connector acting as a generic client. It internally uses one of the available connectors registered with the current Restlet implementation.

Component
Restlet managing a set of Clients, Servers and other Restlets.

"A component is an abstract unit of software instructions and internal state that provides a transformation of data via its interface." Roy T. Fielding

Component managing a set of VirtualHosts and Applications. Applications are expected to be directly attached to VirtualHosts. Components are also exposing a number of services in order to control several operational features in a portable way, like access log and status setting.

Connector

Restlet enabling communication between Components.

"A connector is an abstract mechanism that mediates communication, coordination, or cooperation among components. Connectors enable communication between components by transferring data elements from one interface to another without changing the data." Roy T. Fielding

"Encapsulate the activities of accessing resources and transferring resource representations. The connectors present an abstract interface for component communication, enhancing simplicity by providing a clean separation of concerns and hiding the underlying implementation of resources and communication mechanisms." Roy T. Fielding

Context

Contextual data and services provided to a Restlet. The context is the means by which a Restlet may access the software environment within the framework. It is typically provided by the immediate parent Restlet (Component and Application are the most common cases). The services provided are access to a logger, access to configuration parameters and to a request dispatcher.

Directory

Handler mapping a directory of local resources. Those resources have representations accessed by the file system, the WAR context or the class loaders. An automatic content negotiation mechanism (similar to the one in Apache HTTP server) is used to select the best representation of a resource based on the available variants and on the client capabilities and preferences.

Engine

A Restlet Engine is an implementation of the Restlet API. The reference implementation, provided by Noelios Technologies, is therefore called the Noelios Restlet Engine (NRE).

Filter

Restlet filtering calls before passing them to an attached Restlet. The purpose is to do some pre-processing or post-processing on the calls going through it before or after they are actually handled by an attached Restlet. Also note that you can attach and detach targets while handling incoming calls as the filter is ensured to be thread-safe.

Finder
Restlet that can find the target resource that will concretely handle the request. Based on a given resource class, it is also able to instantiate the resource with the call's context, request and response without requiring the usage of a subclass. Once the target resource has been found, the call is automatically dispatched to the appropriate handle*() method (where the '*' character corresponds to the method name) if the corresponding allow*() method returns true.

For example, if you want to support a MOVE method for a WebDAV server, you just have to add a handleMove() method in your subclass of Resource and it will be automatically be used by the Finder instance at runtime.

If no matching handle*() method is found, then a Status.CLIENT_ERROR_METHOD_NOT_ALLOWED is returned.

Redirector
Rewrites URIs then redirects the call or the client to a new destination.

Representation
Current or intended state of a resource. For performance purpose, it is essential that a minimal overhead occurs upon initialization. The main overhead must only occur during invocation of content processing methods (write, getStream, getChannel and toString).Current or intended state of a resource.

"REST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. A representation is a sequence of bytes, plus representation metadata to describe those bytes. Other commonly used but less precise names for a representation include: document, file, and HTTP message entity, instance, or variant." Roy T. Fielding

Request
Generic request sent by client connectors. It is then received by server connectors and processed by Restlets. This request can also be processed by a chain of Restlets, on the client or server sides. Requests are uniform across all types of connectors, protocols and components.

Resource
Intended conceptual target of a hypertext reference. "Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on. In other

words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource."

"The only thing that is required to be static for a resource is the semantics of the mapping, since the semantics is what distinguishes one resource from another." Roy T. Fielding

Another definition adapted from the URI standard (RFC 3986): a resource is the conceptual mapping to a representation (also known as entity) or set of representations, not necessarily the representation which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content (the representations to which it currently corresponds) changes over time, provided that the conceptual mapping is not changed in the process. In addition, a resource is always identified by a URI.

Response

Generic response sent by server connectors. It is then received by client connectors. Responses are uniform across all types of connectors, protocols and components.

Restlet

Dispatcher that provides a context and life cycle support.

Route

Filter scoring the affinity of calls with the attached Restlet. The score is used by an associated Router in order to determine the most appropriate Restlet for a given call.

Router

Restlet routing calls to one of the attached routes. Each route can compute an affinity score for each call depending on various criteria. The attach() method allow the creation of routes based on URI patterns matching the beginning of a the resource reference's remaining part.

In addition, several routing modes are supported, implementing various algorithms:

- Best match (default)
- First match
- Last match
- Random match
- Round robin
- Custom

Note that for routes using URI patterns will update the resource reference's base reference during the routing if they are selected. If you are using hierarchical paths, remember to directly attach the child routers to their parent router instead of the top level Restlet component. Also, remember to manually handle the path separator characters in your path patterns otherwise the delegation will not work as expected.

Finally, you can modify the routes list while handling incoming calls as the delegation code is ensured to be thread-safe.

Server
Connector acting as a generic server. It internally uses one of the available connectors registered with the current Restlet implementation.

Transformer
Filter that can transform XML representations by applying an XSLT transform sheet.

Uniform
Base class exposing a uniform REST interface.

"The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components. By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. Implementations are decoupled from the services they provide, which encourages independent evolvability." Roy T. Fielding

It has many subclasses that focus on a specific ways to handle calls like filtering, routing or finding a target resource. The context property is typically provided by a parent component as a way to give access to features such as logging and client connectors.

Virtual Host
Router of calls from Server connectors to Restlets. The attached Restlets are typically Applications.

FAQ

How to use Restlet in an Applet ?
In version 2.0, the Restlet engine creates its own classloader, instance of the EngineClassLoader class. This is fine most of the time, except when a security manager is used, such as for Applets running inside a sandbox.

The solution is to use a custom Restlet engine as below, that won't create this new classloader:

```
public class AppletEngine extends Engine {


    @Override
    protected ClassLoader createClassLoader() {
        return getClass().getClassLoader();
    }


}
```

Now you just need to call this line before using the Restlet API:

```
Engine.setInstance(new AppletEngine());
```

Solve 405 status code responses

Having set up your server resource with annotated methods, you're ready to send it requests and eager to get JSON, XML representations of its state. But unfortunately, it fails. In some cases, you're returned an error response with a HTTP 405 status code.By definition, it means that the sent method is not supported which keep you quite disappointed as the resource does declare a "@Get" method!

The solution is all about content negotiation. Let's say your client requires a JSON representation. Before the resource is sollicitated to generate the desired representation, the content negotiation algorithm will detect if the resource is really able to generate a JSON representation, that is to say there is a Java method in the code of the server resource which is (A/) properly annotated, and (B/) having a compatible return type. the case (A/) is easy to understand, let's focus on case (B/). This may happen when you server resource use annotation with media type parameters:

```
public class MyResource extends ServerResource {

  @Get("html")
  public String toHtml() {
    return "<html><body>hello, world</body></html>";
  }
}
```

In this case, the client requires a JSON representation but the server resource is not able to generate it.

This may also happen when you rely on the converter service, and that you don't have properly configured the classpath of your project. Did you reference the archive of the right Restlet extension (such as the Jackson extension) and the archives of its library dependencies?

How to trace the internal client and server connectors?

These connectors are configured via the parameter called "tracing" of their context (see the javadocs).

Here is a sample code that illustrates how to configure the HTTP server connector of a Component:

```
Component c = new Component();

Server s = new Server(Protocol.HTTP, 8182);

c.getServers().add(s);

s.getContext().getParameters().add("tracing", "true");
```

Here is a sample code that illustrates how to configure the HTTP client connector of a resource:

```
Client client = new Client(new Context(), Protocol.HTTP);

client.getContext().getParameters().add("tracing", "true");

ClientResource resource = new
ClientResource("http://localhost:8182/<resource>");

resource.setNext(client);
```

How do I implement the traditional MVC pattern?

There is only a rough correspondence between the MVC pattern and the Restlet framework; some debate exists as to whether it should be employed at all. For those who wish to follow the MVC pattern with Restlet, here is the basic proposition:

- Controller == Restlets (mainly Filters, Routers, Finders). You can visualize the controller as a sort of processing chain, where the last node should be a Finder with all the information necessary to locate the target Resource for the call. Note that Finders are generally implicitely created when attaching Resource classes to a Router.
- Model == Resource + Domain Objects. Just start from the org.restlet.resource.Resource class and load the related Domain Objects in the constructor based on the request attributes (ex:

identifier extracted from the URI). Then you can declare the available variants with getVariants() and override methods like represent(Variant) for GET, acceptRepresentation(Representation) for POST, removeRepresentations() for DELETE or storeRepresentation(Representation) for PUT.

- View == Representation. To expose views of your model, you create new Representations for your Resources. You can leverage on one of the numerous Representation subclasses (InputRepresentation for example) available in the org.restlet.resource package or in extension packages like for JSON documents, FreeMarker and Velocity templates. OSGi deployment ==========

Introduction

This page will explain you how to run your Restlet application in an OSGi environment such as Eclipse Equinox or Apache Felix. For additional coverage of OSGi and Restlet, we recommend you to also read this developer's page.

Since Restlet 2.1 RC2, a new edition of Restlet Framework for OSGi environements is available as well as an Eclipse update site.

Simple example

Since Restlet 1.1, the integration of Restlet and OSGi has become much easier. Each Restlet module and library is an OSGi bundle, and the automatic detection of pluggable connectors and authenticators works in the same way as for the standalone Restlet mode. Here are some instructions to get a simple Restlet project working with OSGi:

1. Copy the content of the Restlet distribution under "lib" folder into the Eclipse "dropins" folder, including all JARs and subdirectories
2. Launch Eclipse 3.4 which should be longer than usual as new plug-ins are automatically installed
3. Open "Help / About ... / Plug-in Details" dialog and check that Restlet plug-ins are there ("Restlet" is the provider name)
4. Create a new "Plug-in Project", name it "OsgiUsageTest1" and press "Next >"
5. Select the checkbox to generate an Activator and press "Finish"
6. In the Dependencies tab of the manifest editor, import the following packages: org.restlet, org.restlet.data, org.restlet.representation, org.restlet.resource, org.restlet.security, org.restlet.service, org.restlet.util

7 Open the "Activator" class generated and type the code below:

```
public class Activator implements BundleActivator {

    private Server server;

    public void start(BundleContext context) throws Exception {
        server = new Server(Protocol.HTTP, 8554, new Restlet() {
            @Override
            public void handle(Request request, Response response) {
                response.setEntity("Hello world!", MediaType.TEXT_PLAIN);
            }
        });

        server.start();
    }

    public void stop(BundleContext context) throws Exception {
        server.stop();
    }

}
```

1 Open the Run Configurations dialog

2 Create a new configuration under the "OSGi framework" tree node

3 Deselect all bundles and select the "OsgiUsageTest1" one

4 Click on the "Add Required Bundles" button to add all dependencies

5 Click on the "Run" button

6 Open your browser at the "http://localhost:8554/" URI

7 "Hello world!" should be displayed!

Complete example

Now, let's look at a more complete example, leveraging the Jetty connector and attaching Resources to a Router. For this we will reuse the Part12 example of the Restlet tutorial.

1 Create a new "Plug-in Project", name it "OsgiUsageTest2 (application/zip, 6.5 kB)" and press "Next >"

2 Select the checkbox to generate an Activator and press "Finish"

3 In the Dependencies tab of the manifest editor, import the following packages: org.restlet, org.restlet.data, org.restlet.representation, org.restlet.resource, org.restlet.security, org.restlet.service, org.restlet.util

4 Open the "Activator" class generated and type the code below:

```
import org.osgi.framework.BundleActivator;

import org.osgi.framework.BundleContext;

import org.restlet.Application;

import org.restlet.Component;

import org.restlet.data.Protocol;


public class Activator implements BundleActivator {


    private Component component;


    public void start(BundleContext context) throws Exception {

        // Create a component

        component = new Component();

        component.getServers().add(Protocol.HTTP, 8182);


        // Create an application

        final Application application = new Test12();
```

```
        // Attach the application to the component and start it
        component.getDefaultHost().attachDefault(application);
        component.start();
    }

    public void stop(BundleContext context) throws Exception {
        component.stop();
    }

}
```

For the rest of the source code, copy and paste from the regular Restlet tutorial available in the "org.restlet.example.tutorial" package the following classes:

- Test12
- OrderResource
- OrdersResource
- UserResource

Now, let's launch our component, using Jetty as our HTTP server:

1. Open the Run Configurations dialog
2. Create a new configuration under the "OSGi framework" tree node
3. Deselect all bundles and select the "OsgiUsageTest2" and the "org.restlet.ext.jetty" bundles
4. Click on the "Add Required Bundles" button to add all dependencies
5. Configure the start order of the bundles: "org.restlet" must start first, give it a start level of "1" and leave the rest to "default"
6. Click on the "Run" button
7. Open your browser at the "http://localhost:8182/users/scott" URI

8 'Account of user "scott"' should be displayed! and the log messages should reflect that Jetty was started.

If you prefer not to rely on start level to select the actual connector to be used, you can specify the connector helper class to use in the Server/Client constructors:

- Client(Context context, List<Protocol> protocols, String helperClass)
- Server(Context context, List<Protocol> protocols, String address, int port, Restlet target, String helperClass)

Otherwise, it is possible to manually register connectors with the engine, for example:

- Engine.getInstance().getRegisteredClients().add(new HttpServerHelper(null));

An archive of this test project is available here (application/zip, 6.5 kB).

Standalone Equinox deployment

Here is a straightforward, light-weight way of setting up an Equinox OSGi container proposed by Dave Fogel:

1) You will need the eclipse equinox osgi jar file (from some recent version of eclipse, in the plugins folder, in this case 3.5M5):

org.eclipse.osgi_3.5.0.v20090127-1630.jar

2) Download the FileInstall bundle by Peter Kriens, which will monitor a directory and automatically install bundles it finds there. (for a longer description you can see http://felix.apache.org/site/apache-felix-file-install.html)

http://www.aqute.biz/repo/biz/aQute/fileinstall/1.3.4/fileinstall-1.3.4.jar

3) you then create the following directory structure (Note substitute your actual equinox bundle version for "3.X.X"):

my_equinox/

    org.eclipse.osgi_3.X.X.jar

    fileinstall-1.3.4.jar

    load/

        configuration/

            config.ini

"load/" is an empty dir, where you will later put the bundles you wish to test.

"config.ini" should be a text file with the following lines in it (but not indented):

osgi.bundles=fileinstall-1.3.4.jar@start

eclipse.ignoreApp=true

4) run equinox from the command line (make sure to "cd" to the "my_equinox" directory first):

java -jar org.eclipse.osgi_3.X.X.jar -console

(or)

java -jar org.eclipse.osgi_3.X.X.jar -console 7777 &

This will start up equinox with a command-line console. if you run the first version above, you will enter the console directly. This can be inconvenient if you have other things to do on the command line. Using the 2nd version will launch equinox in a new process and tell it to listen on port 7777 for telnet connections. To connect to the running osgi console, you then type:

telnet locahost 7777

5) type "help" in the osgi console for a list of commands. To quickly check the status of all installed bundles, type (where "osgi>" is the osgi command prompt):

osgi> ss

"ss" stands for "short status", and in this case you should see something like:

id State Bundle

0 ACTIVE org.eclipse.osgi_3.X.X

1 ACTIVE biz.aQute.fileinstall_1.3.4

6) copy any bundles you want to install to the "load/" directory. The FileInstall bundle will automatically attempt to load and start these bundles. If you add in the bundles in random order, you may see some temporary error messages complaining about missing dependencies, but these should resolve themselves as the rest of your bundles load. you may again type "ss" in the osgi console to see the status of the bundles.

In general, you can use the Run Configurations dialog in eclipse to make sure you have a compatible set of bundles satisfying all their mandatory dependencies, and then copy that set into the /load directory. Let me know if you have any problems with this setup.

Issues when using Restlet within OSGi

Using client connectors

You need to be very careful when using client connectors within an OSGi container. You must be sure that the bundle providing the connector is already loaded when trying to add the client connector. Otherwise I'll see something like that in the trace:

Internal Connector Error (1002) - No available client connector supports the requiredprotocol: 'HTTPS'. Please add the JAR of a matching connector to your classpath.

In this case, before adding your client connector, you need to check the loaded bundles and before executing your REST request, the registered client connector.

Here is the code to see all registered client connectors:

```
List<ConnectorHelper<Client>> clients =
Engine.getInstance().getRegisteredClients();

System.out.println("Connectors - "+clients.size());

for(ConnectorHelper<Client> connectorHelper : clients) {

    System.out.println("connector = "+connectorHelper.getClass());

}
```

You can use OSGi bundle listeners to see if necessary bundles are loaded. Here is a sample of code:

```
// Checking the bundle loading in the future

bundleContext.addBundleListener(new BundleListener() {

    public void bundleChanged(BundleEvent event) {

        if (event.getBundle().getSymbolicName().equals("org.restlet.ext.ssl")


                & event.getBundle().getState()==BundleEvent.RESOLVED) {


            registerClientConnector();
```

```
        }
    }
});


// Checking if the bundle is already present


Bundle[] bundles = bundleContext.getBundles();
for (Bundle bundle : bundles) {
    if (bundle.getSymbolicName().equals("org.restlet.ext.ssl")
            && bundle.getState()==BundleEvent.RESOLVED) {
        registerClientConnector();
    }
}
```

The registerClientConnector method simply does something like that: component.getClients().add(Protocol.HTTPS);.

Other references

Standalone Equinox

- David Fogel - Detailed instruction in this mail
- Wolfgang Werner - Building web services on Equinox and Restlet #1
- Wolfgang Werner - Building web services on Equinox and Restlet #2
- Wolfgang Werner - Building web services on Equinox and Restlet #3

Appendices

In addition to the current documentation, here is a list of additional sources of information that we recommend:

1. Restlet.org (Javadocs, Mailing lists, Issue tracker, etc.)
2. Restlet.com (technical support, commercial licenses, services, etc.)
3. More REST related resources

ECCN

Introduction

This page gives some information to help you obtain an ECCN (Export Control Classification Number) for your Restlet based application.

What are aware of several organizations that attempted to obtain such a number for their Restlet-based application but we don't think that there is a single ECCN for Restlet. Depending on the actual Restlet extensions that you are using and redistributing, this classification could change.

In order to help you find your ECCN, we try to answer the typical questions related to the classification process.

FAQ

Does Restlet include any encryption technology?

The Restlet Framework has a cryptographic extension (org.restlet.ext.crypto.jar file) that includes all cryptographic related features. It is based on regular Java Cryptography APIs (javax.crypto) and used for authentication purpose only (so far):

In addition, we can take advantage of SSL/HTTPS features to encrypt communications.

1)**org.restlet.security** package

This is the generic security API for Restlet dealing with authentication and authorization. It doesn't contain actual implementations for specific schemes.

However, pluggable authenticator helpers can be registered in the Restlet engine, such as these ones in the crypto extension (note the "internal" packages are hidden from public Javadocs).

2) **org.restlet.engine.http.security** package

This package contains one class for HTTP Basic support, not really encryption related. This was removed in version 2.1 and the HttpBasicHelper was moved into org.restlet.engine.security.

3) **org.restlet.engine.security** package

This package contains SSL related classes, but only in version 2.0. In version 2.1 those classes were moved to the org.restlet.ext.ssl extension:

What encryption algorithms and key lengths are used?

When we encrypt authentication data in a cookie, we give the option to change the algorithm and the secret key, see details here.

For the HTTPS support in connectors, the DefaultSslContextFactory uses :

- Algorithm used: “TLS” (see the JSSE reference for details and RFC 2246 for TLS 1.0 specs)
- Certificate algorithm: “SunX509”
- Key store type: via System.getProperty("javax.net.ssl.keyStoreType")
- Key length : depends on the certificate that you configure for your HTTPS server (if any)

What its encryption function is designed for?

Authentication

The purpose is authentication for the CookieAuthenticator class and data hiding for the HTTPS/SSL connectors. Note that those software components are optional, so you can still use Restlet without relying on any encryption technology.

Digital signature

No, beside the HTTP digest feature (see Content-MD5 header) supported via the org.restlet.data.Digest and org.restlet.representation.DigesterRepresentation classes.

Software copy protection

No.

Data hiding

The purpose is data hiding for the HTTPS/SSL connectors. Note that those software components are optional, so you can still use Restlet without relying on any encryption technology.

Are there any differences between Restlet version 2.0 and 2.1?

There are the three classes that were moved from the org.restlet.engine.security package related to SSL into the Crypto extension. See these two pages to compare:

- org.restlet.engine.security package in Restlet 2.0

- org.restlet.engine.security package in Restlet 2.1
- org.restlet.engine.security package in Restlet 2.3

Here are some precisions for version 2.0. Those two classes, do not contain or rely on any encryption algorithm:

- SslContextFactory is an empty abstract class
- SslUtils is only providing utility method to extract the length of the key used and parse parameters

So, only DefaultSslContextFactory is interesting here (see source code here).

Is there both object and source code for the encryption technology?

Yes, Restlet is an open source project, and all source code is publicly available. Note that the encryption features provided rely on Java or third-party software. You should verify the eligibility of any dependency that you are using via Restlet. RESTful Web Services book ==========

Introduction

This book is the first one dedicated to REST. It is authored by Leonard Richardson and Sam Ruby and published by O'Reilly.

For now, please find the source code of all the book examples that were reimplemented using Restlets. Those examples are also available and the Restlet distributions under the "src/org.restlet.example" directory.

Representation package

The **org.restlet.representation** package contains common representation data elements. Here is a hierarchy diagram with the core Representation classes:

representations Local connectors ==========

The "local" Reference instances can be easily created via the LocalReference class.

The list of supported parameters is available in the javadocs:

- Local client commons parameters

HTTP connector (internal)

- HTTP client : supporting chunked encoding, persistent connections and asynchronous processing, but not HTTPS

- HTTP server : supporting chunked encoding, persistent connections and asynchronous processing, but not HTTPS

The internal HTTP connectors can be configured with several sets of parameters:

- Base parameters
- Connection parameters
- Client parameters
- Server parameters Restlet Internal Access Protocol
  ==========

Introduction

RIAP is used to access to other resources hosted in the same Restlet Component or VirtualHost or Application, allowing powerful modularization of larger Restlet applications.

This pseudo-protocol came to life as a solution for the issue 374 "Add support for pseudo protocols". Which in turn started of based from discussion on issue 157 "Optimize Internal Calls"?

Description

The riap:// scheme identifies a so-called pseudo-protocol (terminology derived from Apache Cocoon pointing out the difference between 'real' or 'official'/ public protocols and this scheme which only relates to internal processing of the Restlet system architecture.

RIAP is short for 'Restlet Internal Access Protocol'. In short, it is just a mechanism that allows various applications to call upon each other to retrieve resources for further processing.

"Application" here refers to the strict Restlet API notion. Same remark should be made on the usage of "Component" further down.

The riap:// scheme brings a URI-notation for these inter-application calls. With that URI-notation those calls fall naturally under appliance of the uniform interface. There are three blends of this protocol, that use distinct 'authorities' for different use cases.

riap://component/**

- resolves the remainder URI with respect to the current context's "component"

riap://host/**

- resolves the remainder URI with respect to the current context's "virtual host"

riap://application/**

- resolves the remainder URI with respect to the current context's "application"
- (so applications could use this scheme to call resources within themselves!)

Use case

Think about some application X that needs a resource Y that is available on a configurable base URI. Whatever that base URI is, the application-implementation should just use the context.getClientDispatcher() to get() that needed resource.

So basically the dependency from application X to "whatever service that will offer me resource Y" can be expressed in this base-URI. Thinking about dependency injection solutions, the configuration of X is covered by calling some setResourceYBaseUri(...). That will cover any of these 'service-providers' of this resource Y:

file://whateverpath/Y

- a local static file

http://service.example.com/somepath/Y

- an external service provider

riap://component/internalpath/Y

- a URI resolved versus the internal router of the current 'Component'

riap://application/fallback/Y

- this application itself providing some basic version of the required resource.

Next to this expression flexibility of dependencies, this riap:// approach ensures some extra

efficiency:

1 It avoids going through the actual network layers versus the alternative of calling http://localhost/publicpath/Y

2 It offers direct access to the Response and Representation objects allowing to bypass possibly useless serialization-deserialization (e.g. for sax and or object representations)

shielding:

1 It offers the possibility to keep the internal component offering resource Y to be available only internally (not publicly over http://hostname/publicpath/Y)

Bottom line: the basic idea behind this riap:// is one of flexibly decomposing your 'component' in smaller reusable/configurable/interchange-able 'applications' while assuring optimal efficiency when calling upon each other.

How to use

Calling the riap:

To call a resource via the riap:// scheme one can just use:

```
Context context;


context.getClientDispatcher().get("riap://component/some-path/Y");
```

There is no need to register a RIAP client, this is handled by built-in support.

Making resources available to the riap

Applications need just to be attached to the internalRouter property of the Component.

```
Component component;


component.getInternalRouter().attach("/path", someApplication);
```

Applications can be attached multiple times (at different paths) to both the internal router as to several virtual hosts.

A 'pure internal' application should only be attached to the internal router.

Limitations

Note that internal/dynamic resources that require protocol specific attributes of the URI where it is invoked (like hostname) might yield errors or unexpected results when called via the RIAP protocol.

Also beware that constructed URI's to secundary resources which are based on the 'own reference' can yield unexpected or at least different results when the resource is suddenly being called via RIAP.

In addition, if you attempt to nest applications, the sub-application can't refer to the parent application via RIAP at this point.

Give these cases some special attention during your design, specially when porting stuff from other environments (e.g. Servlet app)

Internal connectors

Here is the list of connectors built-in the Restlet engine:

- Local
- FILE
- CLAP
- HTTP
- RIAP
- Zip

Note that those connectors available in the engine JAR but still need to be declared in your Restlet component, otherwise your applications won't be able to use them!

CLAP connector

CLAP (ClassLoader Access Protocol) is a custom scheme to access to representations via classloaders (e.g.: clap://thread/org/restlet/Restlet.class). This protocol accepts three kinds of authority :

- class: the resources will be resolved from the classloader associated with the local class.
- system: the resources will be resolved from the system's classloader.
- thread: the resources will be resolved from the current thread's classloader.

CLAP client commons parameters

Restlet Engine

Introduction

The engine is the set of classes that supports or implements the Restlet API. Since version 2.0, the Restlet API and Engine have been merged in a single "org.restlet.jar" file.

This separation has however stayed conceptually between the classes that are expected to be used by Restlet API users while developing their applications and the one that power this API and provide extensibility those additional connectors, authenticators and so on.

Internal connectors

In addition to supporting the Restlet API, the Reslet Engine includes a set of internal connectors that can be further extended or replaced through extensions. # Base package

This **org.restlet** package contains the most important classes of the Restlet API, mapping key HTTP and REST concepts to Java including:

- **Client**: Connector acting as a generic client.
- **Component**: Restlet managing a set of Connectors, VirtualHosts, Services and Applications.
- **Connector**: Restlet enabling communication between Components.
- **Message**: Generic message exchanged between components.
- **Request**: Generic request sent by client connectors.
- **Response**: Generic response sent by server connectors.
- **Server**: Connector acting as a generic server.
- **Uniform**: Uniform REST interface.

It also contains key framework classes:

- **Application**: Restlet managing a coherent set of resources and services.
- **Context**: Contextual data and services provided to a set of Restlets.
- **Restlet**: Uniform class that provides a context and life cycle support. # Connectors

Introduction

A connector in the REST architecture style is a software element that manages network communication for a component, typically by implementing a network protocol (e.g. HTTP). A client connector initiates communication with a server (of any kind) by creating a request. A server connector listens for connections (from clients of any kind), transmits the request to the component that performs the request processing, creates the response and sends it to the client.

All connectors are provided as plugins for the Restlet Engine. This document will describe how to add a connector to your application, how to configure it and will give you the list of available server and client connectors.

Add a connector to your application

All connectors and their dependencies are shipped with the Restlet distribution by the way of jar files. Adding a connector to your application is as simple as adding the archives of the chosen connector and its dependencies to the classpath.

Configuration

Each connector looks for its configuration from its context. The latter provides a list of modifiable parameters, which is the right place to set up the connector's configuration. Some parameters are defined by the Restlet engine and thus are shared by all clients (in the ClientHelper hierarchy) and server connectors (in the ServerHelper hierarchy), and most of them by the connector's ClientHelper or ServerHelper subclasses.

The list of all parameters are available in the javadocs. Pleaser refer to the rest of this document for references to these documentation. Here are the commons parameters dedicated to internal connectors.

Server connectors

Here are the commons parameters dedicated to non-internal HTTP server connectors.

Here are the commons parameters dedicated to internal HTTP server connectors.

Here is a sample code showing how to set such a parameter on a component's server connector.

```
// Create the HTTP server and listen on port 8182

Component c = new Component();

Server server = c.getServers().add(Protocol.HTTP, 8182);
```

```
server.getContext().getParameters().add("useForwardedForHeader", "true");

c.start();
```

Client connectors

Here are the commons parameters dedicated to non-internal HTTP client connectors.

Here are the commons parameters dedicated to internal HTTP client connectors.

Here is a sample code showing how to set such a parameter.

```
Client client = new Client(new Context(), Protocol.HTTP);

client.getContext().getParameters().add("useForwardedForHeader","false");
```

Here is a sample code showing how to set such a parameter on a component's client connector.

```
// Create the HTTP server and listen on port 8182

Component c = new Component();

Client client = c.getClients().add(Protocol.HTTP);

client.getContext().getParameters().add("useForwardedForHeader", "false");
```

If you want to configure the client connector used by a ClientResource, there are several cases. When your ClientResource instances are created in the context of an application hosted by a Component, the client connector of the component is used for all requests. Thus, just configure the component's client connector as shown just above. If not, just set it:

```
// Instantiate the client connector, and configure it.

Client client = new Client(new Context(), Protocol.HTTP);

client.getContext().getParameters().add("useForwardedForHeader","false");



// Instantiate the ClientResource, and set it's client connector.

ClientResource cr = new ClientResource("http://www.example.com/");

cr.setNext(client);
```

List of available connectors

Server connectors

Extension

Internal

Jetty

NIO

Simple

Servlet

Client connectors

Extension | Version | Protocols | Asynchronous | Proxy | Comment --- | --- | --- | --- | --- | --- Internal | 2.2 | CLAP, FILE, FTP, HTTP, HTTPS, RIAP | No | Yes | Recommended for development and lightweight deployments Apache HTTP Client | 4.3 | HTTP, HTTPS | No | Yes | Recommended for robust and scalable deployments JavaMail | 1.4 | SMTP, SMTPS, POP, POPS | No | No | Stable JDBC | 3.0 | JDBC | No | No | Stable Lucene Solr | 4.6 | SOLR | No | No | Stable NIO | 2.2 | HTTP, HTTPS | Yes | Yes | Fully asynchronous, preview mode Authentication ==========

Introduction

There are two commons ways to authenticate your users with your Restlet application. The first is to use is to leverage the standard HTTP authentication mechanism, either the Basic or the Digest authentication. The Basic mechanism is sending the password in clear and should only be used over a secure HTTPS channel. The second mechanism is to use a custom authentication form and some cookie set by the server.

HTTP Basic authentication

Introduction

Here is a very simple code illustrating a component that guards its applications with the BASIC authentication scheme.

The whole code can be downloaded here (application/force-download, 1.2 kB).

Description of the server side

Component

The sample component is composed of two parts. The first one is a very simple Restlet that answers to request with a "hello, word" text representation. Of course this Restlet is a very simple representation of your own complex application.

```
// Restlet that simply replies to requests with an "hello, world" text message

Restlet restlet = new Restlet() {

   @Override

   public void handle(Request request, Response response) {

      response.setEntity(new StringRepresentation("hello, world",
MediaType.TEXT_PLAIN));

   }

};
```

Then, the component protects this Restlet with a ChallengeAuthenticator instance based on the BASIC authentication scheme. Once the instance is created, it is given the list of known (login/password) pairs via the "verifier" attibute.

```
// Guard the restlet with BASIC authentication.

ChallengeAuthenticator guard = new ChallengeAuthenticator(null,
ChallengeScheme.HTTP_BASIC, "testRealm");

// Instantiates a Verifier of identifier/secret couples based on a simple Map.

MapVerifier mapVerifier = new MapVerifier();

// Load a single static login/secret pair.

mapVerifier.getLocalSecrets().put("login", "secret".toCharArray());

guard.setVerifier(mapVerifier);


guard.setNext(restlet);


Component component = new Component();

component.getServers().add(Protocol.HTTP, 8182);
```

```
component.getDefaultHost().attachDefault(guard);

component.start();
```

Customization

As you may have noticed earlier, the list of known login/password pairs are loaded by the authenticator via the "verifier" attribute via a simple Verifier based on a Map. This is useful when the list of pairs are known by advance and is not susceptible to change. In a more realistic case, the credentials are hosted in a database or a LDAP directory, etc. In this case, the responsibility to resolve the password according to the login is deported to a class that implements the org.restlet.security.Verifier interface.

The contract of the Verifier is a simple method which aims at returning the given password according to the given login. The Restlet framework provides an abstract implementation that retrieves the pair of identifier/secret in the request's challenge response and allow to verify them:

```
import org.restlet.security.LocalVerifier;


public class TestVerifier extends LocalVerifier {


    @Override
    public char[] getLocalSecret(String identifier) {
        // Could have a look into a database, LDAP directory, etc.
        if ("login".equals(identifier)) {
            return "secret".toCharArray();
        }


        return null;
    }


}
```

Description of the client side

The credentials are transmitted to the request via a ChallengeResponse object as follow:

```
public static void main(String args[]) {

    ClientResource resource = new ClientResource("http://localhost:8182/");



    // Send an authenticated request using the Basic authentication scheme.

    resource.setChallengeResponse(ChallengeScheme.HTTP_BASIC, "login",
"secret");



    // Send the request

    resource.get();

    // Should be 200

    System.out.println(resource.getStatus());

}
```

If you try to access http://localhost:8182/ via a web browser, a window will appear to type in your credentials.

HTTP Digest authentication

Introduction

Here is a very simple code illustrating a component that guards its applications with the DIGEST authentication scheme.

The whole code can be downloaded here (application/force-download, 1.6 kB).

Description of the server side

Component

The sample component is composed of two parts. The first one is a very simple Restlet that answers to request with a "hello, word" text representation. Of course this Restlet is a very simple representation of your own complex application.

```
// Restlet that simply replies to requests with an "hello, world" text message

Restlet restlet = new Restlet() {

    @Override
```

```
    public void handle(Request request, Response response) {

        response.setEntity(new StringRepresentation("hello, world",
MediaType.TEXT_PLAIN));

    }

};
```

Then, the component protects this Restlet with an Authenticator instance based on the DIGEST authentication scheme. A dedicated constructor of the Guard class of the Restlet API allows you to create such instance. Once the instance is created, it is given the list of known (login/password) pairs via the "secrets" attibute.

```
DigestAuthenticator guard = new DigestAuthenticator(null, "TestRealm",
"mySecretServerKey");



// Instantiates a Verifier of identifier/secret couples based on a simple Map.

MapVerifier mapVerifier = new MapVerifier();

// Load a single static login/secret pair.

mapVerifier.getLocalSecrets().put("login", "secret".toCharArray());

guard.setWrappedVerifier(mapVerifier);



// Guard the restlet

guard.setNext(restlet);



component.getDefaultHost().attachDefault(guard);
```

Customization

As you may have noticed earlier, the list of known login/password pairs are loaded by the authenticator via the "verifier" attribute via a simple Verifier based on a Map. This is useful when the list of pairs are known by advance and is not susceptible to change. In a more realistic case, the credentials are hosted in a database or a LDAP directory, etc. In this case, the responsibility to resolve the password according to the login is deported to a class that implements the org.restlet.security.Verifier interface.

The contract of the Verifier is a simple method which aims at returning the given password according to the given login. The Restlet framework provides an abstract implementation that retrieves the pair of identifier/secret in the request's challenge response and allow to verify them:

```
import org.restlet.security.LocalVerifier;


public class TestVerifier extends LocalVerifier {


    @Override

    public char[] getLocalSecret(String identifier) {

        // Could have a look into a database, LDAP directory, etc.

        if ("login".equals(identifier)) {

            return "secret".toCharArray();

        }


        return null;

    }


}
```

Description of the client side

The authentication with the DIGEST scheme is bit more difficult than the one for the BASIC scheme. The credentials provided by the client is the result of computation of data given by the client on one side (login and password) and by the server on the other side ("nonce" value, "realm" value, etc.). Unless these pieces of data are known in advance by the client, it appears that in general a first request is required in order to collect them.

```
ClientResource resource = new ClientResource("http://localhost:8182/");


resource.setChallengeResponse(ChallengeScheme.HTTP_DIGEST, "login",
"secret");

// Send the first request with unsufficient authentication.
```

```
try {
    resource.get();
} catch (ResourceException re) {
}
// Should be 401, since the client needs some data sent by the server in
// order to complete the ChallengeResponse.
System.out.println(resource.getStatus());
```

Then, the second step allows to get the required data for the final computation of a correct ChallengeResponse object:

```
// Complete the challengeResponse object according to the server's data
// 1- Loop over the challengeRequest objects sent by the server.
ChallengeRequest c1 = null;
for (ChallengeRequest challengeRequest : resource.getChallengeRequests())
{
    if (ChallengeScheme.HTTP_DIGEST.equals(challengeRequest.getScheme()))
{
        c1 = challengeRequest;
        break;
    }
}

// 2- Create the Challenge response used by the client to authenticate its
requests.
ChallengeResponse challengeResponse = new ChallengeResponse(c1,
                                                resource.getResponse(),
                                                "login",
                                                "secret".toCharArray());
```

Finally, the request is completed with the computed ChallengeResponse instance:

```
resource.setChallengeResponse(challengeResponse);
```

```
// Try authenticated request

resource.get();

// Should be 200.

System.out.println(resource.getStatus());
```

Security package

Introduction

The **org.restlet.security** package contains classes related to security. As there are numerous protocols (like HTTP, SMTP, etc.) that we want to support and as each one has various ways to authenticate requests (HTTP Basic, HTTP Digest, SMTP Plain, etc.), the Restlet API provides a flexible mechanism to support them all in a unified way.

In this section, we will start off with an example explaing how to guard sensitive resources and then describe with more details the security API, the engine support and available extensions.

Tutorial

When you need to secure the access to some Restlets, several options are available. A common way is to rely on cookies to identify clients (or client sessions) and to check a given user ID or session ID against your application state to determine if access should be granted. Restlets natively support cookies via the Cookie and CookieSetting objects accessible from a Request or a Response.

There is another way based on the standard HTTP authentication mechanism. By default, the Restlet Engine accepts credentials sent and received in the Basic HTTP and Amazon Web Services schemes.

When receiving a call, developers can use the parsed credentials available in Request.challengeResponse.identifier/secret via the ChallengeAuthenticator filter. Filters are specialized Restlets that can pre-process a call before invoking and attached Restlet or post-process a call after the attached Restlet returns it. If you are familiar with the Servlet API, the concept is similar to the Filter interface. See below how we would modify the previous example to secure the access to the Directory:

```
@Override

public Restlet createInboundRoot() {

   // Create a simple password verifier
```

```
        MapVerifier verifier = new MapVerifier();
        verifier.getLocalSecrets().put("scott", "tiger".toCharArray());

        // Create a guard
        ChallengeAuthenticator guard = new ChallengeAuthenticator(
            getContext(), ChallengeScheme.HTTP_BASIC, "Tutorial");
        guard.setVerifier(verifier);

        // Create a Directory able to return a deep hierarchy of files
        Directory directory = new Directory(getContext(), ROOT_URI);
        directory.setListingAllowed(true);
        guard.setNext(directory);

        return guard;
    }
```

Note that the authentication and authorization decisions are clearly considered as distinct concerns and are fully customizable via dedicated filters that inherit from the Authenticator (such as ChallengeAuthenticator) and the Authorizer abstract classes. Here we simply hard-coded a single user and password couple. In order to test, let's use the client-side Restlet API:

```
    public static void main(String[] args) {
        // Prepare the request
        ClientResource resource = new ClientResource("http://localhost:8182/");

        // Add the client authentication to the call
        ChallengeScheme scheme = ChallengeScheme.HTTP_BASIC;
        ChallengeResponse authentication = new ChallengeResponse(scheme,
"scott", "tiger");
```

```
        resource.setChallengeResponse(authentication);


        try {
            // Send the HTTP GET request
            resource.get();
            // Output the response entity on the JVM console
            resource.getResponseEntity().write(System.out);
        } catch (Exception e) {
            if
(resource.getStatus().equals(Status.CLIENT_ERROR_UNAUTHORIZED)) {
                // Unauthorized access
                System.out
                        .println("Access unauthorized by the server, check your
credentials");
            } else {
                // Unexpected status
                System.out.println("An unexpected status was returned: "
                        + resource.getStatus());
            }
        }
    }
```

You can change the user ID or password sent by this test client in order to check the response returned by the server. Remember to launch the previous Restlet server before starting your client. Note that if you test your server from a different machine, you need to replace "localhost" by either the IP address of your server or its domain name when typing the URI in the browser. The server won't need any adjustment due to the usage of a VirtualHost which accepts all types of URIs by default.

Restlet API

The Restlet API in version 2.3 has fully refactored its security model (see specifications here). It is based on some properties of the ClientInfo class: user and roles. This model relies on Java security principals in a way similar to JAAS. But Restlet security API and JAAS

are distinct, and some bridges are required if you want to use JAAS (see the org.restlet.ext.jaas extension).

The security model distinguishes the authentication and authorization aspects.

The authentication steps relies on a dedicated filter which is a subclass of Authenticator (e.g. ChallengeAuthenticator, or your own) which verifies the credentials stored in the request (challengeResponse attribute) thanks to a Verifier (e.g. a SecretVerifier), Then, the Enroler (if any) is called in order to set the list of roles of the user.

Once the credentials have been authenticated and the User and Roles have been set by the Authenticator/Verifier/Enroler, then the request is ready to be routed to the right resource.

If the request is not authenticated, a "Unauthorized (401)" response is sent back, by default.

According to your policy, some parts of the hierarchy of resource are only authorized to some kinds of people. This is the role of the Authorizer filter. In the sample code attached to this page, we've implemented a RoleAuthorizer filter that checks a set of authorized roles. You will see that the URIs hierarchy is split in two parts, each of them is protected by its own instance of RoleAuthorizer.

If a request is not authorized, a "Forbidden (403)" response is sent back, by default.

Of course, you can customize each part: Authenticator, Verifier, Enroler, Authorizer.

Regarding the Authenticator helper, this piece of code is used to handle the conversion of header values (of the protocol, e.g. HTTP) with the Restlet API model (Request/Response) in one direction, or both for a certain type of Authentication challenge scheme. At this time, the following schemes are supported:

- HTTP_BASIC (client and server) with the core module
- HTTP_DIGEST (client and server) with the crypto extension
- Amazon S3, shared_key and shared_key_lite (client) with the crypto extension
- SMTP (client) with the core module

ChallengeRequest

Contains information about the authentication challenge that is sent by an origin server to a client.

For server-side Restlet applications, this object will typically be created and set on the current Response by a Guard added to the request processing chain. Then, when the response goes back to the Restlet engine, the server connector and the matching authentication helper will format it into the proper protocol artifact such as a "WWW-Authenticate" header in HTTP.

For client-side Restlet applications, this object will typically be received in response to a request to a target resource requiring authentication. In HTTP, this is signaled by a 401 (Unauthorized) status. This means that a new request must be sent to the server with a valid challenge response set for the required challenge scheme. The client connector and matching AuthenticationHelper are responsible for parsing the protocol artifact such as the "WWW-Authenticate" header in HTTP.

ChallengeResponse

Contains information about the authentication challenge response sent by a client to an origin server.

For server-side Restlet applications, this object will typically be created by the server connector with the help from the matching AuthenticationHelper for parsing.

For client-side Restlet applications, this object must be manually created before invoking the context's client dispatcher for example.

ChallengeScheme

Indicates the challenge scheme used to authenticate remote clients. This only identifies the scheme used or to be used but doesn't contain the actual logic needed to handle the scheme. This is the role of the AuthenticatorHelper subclasses.

Restlet Engine

Most of the logic related to authentication is located in the package "org.restlet.security".

AuthenticatorUtils

Static utilities methods to parse HTTP headers, find the matching authentication helper and misc methods.

AuthenticatorHelper

Base class for authentication helpers. There are also subclasses for the schemes internally supported by the engine or via extensions :

- HTTP Basic
- SMTP Plain

Extensions

In addition to the internal authentication helpers, additional schemes can be supported using pluggable extensions. Currently, there is an org.restlet.ext.crypto extension available for:

- HTTP Digest
- HTTP Amazon S3 (client)
- HTTP Azure SharedKey

Confidentiality

SSL is typically used to ensure that requests and responses are exchanged confidentially between clients and a server. Configuring SSL requires the setting of several parameters on the HTTPS server connector instance used. You need to provide a key store containing the certificate for your server instance. See connectors' documentation for more configuration details.

Note that SSL configuration is closely associated with an IP address and a listening port. This may cause issue with the usage of virtual hosting where several domains and applications share the same IP adress and listening port. It is not possible to associate a SSL certificate per virtual host because in order to determine a virtual host, we need to have read and parsed the HTTP request headers to get the "Host" header. You can only do that if you already use the certificate to read the incoming SSL stream.

In this case, the only solution is to have two listening server sockets (hence two IP addresses if you want to use the default HTTPS 443 port) and then two Restlet server connectors configured, each one pointing to a specific certificate. With some additional work, it is possible to use the same key store to provide both certificate. This requires usage of alias names and custom SSL context factories.

In addition to the parameters that are similar but specific to each type of HTTPS server connector, it is possible to configure the SSL connectors using an SslContextFactory, in a way that is common to all three types of HTTPS server connectors (Simple, Jetty and Grizzly). Configuring SSL is done in this order:

1. An instance of SslContextFactory can be passed in the sslContextFactory *attribute* of the connector's context. This can be useful for cases that requires such an instance to be customized.

2. When no sslContextFactory *attribute* is set, the full name of a concrete class extending SslContextFactory can be passed in the sslContextFactory *parameter* of the connector's context. Such a class *must*have a default constructor. The context's parameters are passed to its init method, so as to initialize the SslContextFactory instance via text parameters.

3. The org.restlet.engine.security.DefaultSslContextFactory is an SslContextFactory that supports a basic set of parameters, and will default to the values specified in the javax.net.ssl.* system properties (see JSSE Reference guide).

4. There can in fact be several values of sslContextFactory (since there can be several values for parameters), in which case the first one constructed and initialized successfully will be used.

5. If no sslContextFactory *attribute* or *parameter* is set, the configuration will fall back to the parameters that are specific to each type of connector.

SSL client authentication is not configured as part of the SSL context, although the trust store and which peer certificates to trust are.

Authentication

In Restlet, authentication and authorization are handled very differently than in the Servlet world. Here you have full control of the process, no external XML descriptor is necessary.

The default approach is to use the org.restlet.Guard class (a Restlet filter) or a subclass of it. By default, this class uses a map to store the login/password couples but this can be customized by overriding the "authenticate(Request)" method.

From an org.restlet.data.Request, the login/password can be retrieved using these methods:

- Request.getChallengeResponse().getIdentifier() : String // LOGIN

- Request.getChallengeResponse().getSecret() : String // PASSWORD

The HTTP server connectors currently only support HTTP BASIC authentication (the most widely used).

SSL client authentication

To enable client-side SSL authentication on an HTTPS server connector, set the wantClientAuthentication or needClientAuthentication parameters to true (in the first case, presenting a client certificate will be optional). The chain of client certificate is then accessible as List of X509Certificates in the org.restlet.https.clientCertificates Request attribute.

Coarsed grained authorization

For the authorizations that are common to a set of resources, a Guard subclass can also be used by overriding the "authorize(Request)" method. Note that this method accepts all authenticated requests by default. You can plugin in your own mechanism here, like an access to a LDAP repository.

Fine grained authorization

If the permissions are very fine grained, the authorizations should probably handled at the resource level directly instead of the Guard filter level (unless you want to put a filter in front of each resource). With the approach, you can make your permission depend on the value of the target resource on the resource and specific method invoked, etc.

Potential vulnerabilities

There are many ways of securing REST style applications; most of the common and easily usable ones involve a set of credentials that the user agent automatically presents on behalf of the user each time it makes any request. HTTP Basic and Digest Authentication, cookies, and even SSL certificate authentication all work this way; once the user agent (browser) has obtained the credentials, it sends the credentials every time it asks the server to perform an operation.

However, this convenience comes at a price: a class of vulnerabilities known as "XSRF" -- cross site request forgery. If a malicious site (http://badsite) embeds a GET reference to a URI on a target site, for example, by simply adding an , when user agents visit http://badsite, *if they are already authenticated* to http://targetsite, the GET request will be performed with authentication, and will succeed.

With a little more work, and the availability of Javascript on the malicious site, this attack can be easily extended to perform POST operations as well. Other HTTP verbs (e.g. PUT and DELETE) are difficult to exploit via XSRF, as this requires access to XmlHttpRequest or analogous components that are generally written to adhere to same-source rules. Still, it would be unwise to utterly discount the possibility.

Some good practices for defending against XSRF include:

Do not ever allow GET operations to have side effects (like transferring money to an attacker's bank account...) GET should expose the state of a resource, not send a command.

POST is for sending commands that have side effects. To trivially protect these, require some additional form of authentication inside the entity of a POST operation. For example, if the client is authenticated by a cookie, require that the client restate the cookie inside the entity of the POST. This will be difficult for an XSRF attacker to know how to forge.

For the most resistance, the server can present the client with an application level authorization token, an opaque value that the server can verify belongs to the right authenticated user.

- Such a token should be be difficult for a third party to calculate, e.g. a server-salted MD5 or SHA1 hash of the user's identification credential.
- To defeat XSRF, this application-level token needs to be transmitted by means that the user-agent does *not*automatically return with each request. For example, it can be sent in the HTML of a form as a hidden field, and returned via POST in the encoded form entity.

Because Restlet supports flexible URI patterns, you can embed an application-level token in your URIs in many ways (e.g. /admin/{token}/delete/youraccount). This provides an effective, if circuitous, means of protecting PUT, DELETE and other verbs from unlikely XSRF attacks. It also may help to avoid certain brute force or denial of service attacks against well-known URI targets.

If cookies are being used as a primary or supplemental credential, protect them against XSS by setting the accessRestricted property in CookieSetting; this stops the cookies from being used by script in all modern browsers.

Sample code

Security sample (application/zip, 2.9 kB)

Pluggable authenticators

Introduction

The aim of the document is to explain how to support its own challenge scheme. Let's say you have defined your own challenge scheme called "MySCHEME". Basically, the server's response to unauthenticated request will contain a WWW-Authenticate header as follow:

WWW-Authentication: MySCHEME realm=”<realm>”

Definition of the custom challenge scheme

You will first need to declare your own ChallengeScheme, certainly as a static member:

```
public static ChallengeScheme MySCHEME = new ChallengeScheme("This is my own challenge scheme", "MySCHEME");
```

Definition of the custom authentication helper

This helper is responsible to cope with server and/or client sides requirements:

- generate the right response ("WWW-Authenticate" header) and parse the client request ("Authorization" header) on server side,
- generate the right request ("Authorization" header) and parse of the server's response ("WWW-Authenticate" header) on client side.

This helper need to declare that it supports a given challenge scheme, and need to be registered by the engine.

The registration can be done manually:

```
Engine.getInstance().getRegisteredAuthenticators().add(new MyCustomAuthenticationHelper());
```

or "magically" by creating a service file located in the "META-INF/services" and called "org.restlet.engine.security.AuthenticatorHelper". It contains only one line of text which is the full path of the the helper class.

Then, the helper must declare its support of a ChallengeScheme. This is done in the constructor:

```
public MyCustomAuthenticationHelper() {
    super(ChallengeScheme.CUSTOM, false, true);
}
```

The two boolean values correspond to the support of the client and server sides of the authentication: in the sample code above, this helper supports only the server side of the challenge scheme.

The support of server side authentication is denoted by the implementation of the following methods:

- formatRawRequest(ChallengeWriter, ChallengeRequest, Response, Series<Parameter>) which is responsible to generate a correct "WWW-Authenticate" header by writing to the given instance of

ChallengeWriter. By default, nothing is done which results in the following header:

```
WWW-Authentication: MySCHEME
```

In order to add the realm, proceed as follow:

```
   public void formatRawRequest(ChallengeWriter cw,
                                ChallengeRequest challenge,
                                Response response,
                                Series<Parameter> httpHeaders) throws IOException {
       if (challenge.getRealm() != null) {
           cw.appendQuotedChallengeParameter("realm",
challenge.getRealm());
       }
   }
```

- parseResponse(ChallengeResponse, Request, Series<Parameter>) which let the helper complete the given instance of ChallengeResponse (it typically parses the ChallengeResponse#.getRawValue

The support of client side authentication is denoted by the implementation of the following methods:

- formatRawResponse(ChallengeWriter, ChallengeResponse, Request, Series<Parameter>) which generates the right "Authorization" header by using the given instance of ChallengeWriter
- parseRequest(ChallengeRequest, Response, Series<Parameter>) which parses the response sent by the server. It especially parses the ChallengeRequest#getRawValue.

Authorization

Introduction

Authorization should happen after authentication to define what an authenticated user is effectively authorized to do in your application. We identified four approaches to authorization in a Restlet application. When we mentionned LDAP as a permission store, it could be replaced by a static file or by any sort of database.

You can find the sources of the code shown here.

Coarse-grained authorization

You configure some Authorizer instances (or subclasses like RoleAuthorizer, MethodAuthorizer, etc.) and attach them at key points in the call routing graph (before a Router, for a single route, etc.).

This works nicely if you can assign roles to your authenticated users and if your authorization rules don't vary too much between resources. Otherwise, you might end-up with an Authorizer instance in front of each resource class...

Here is a simple example, you will get a String response containing the resource's name when accessing resources with authorized profile and a "403 Forbidden" error otherwise.

This code is self-sufficient so you just need to copy-paste it in a MyApiWithRoleAuthorization class, create the resources as shown below and run it. Don't forget to add the JSE edition org.restlet.jar (download here) in your build path.

Main class for role authorization example:

```
public class MyApiWithRoleAuthorization extends Application {


    //Define role names

    public static final String ROLE_USER = "user";

    public static final String ROLE_OWNER = "owner";


    @Override

    public Restlet createInboundRoot() {

        //Create the authenticator, the authorizer and the router that will be
protected

        ChallengeAuthenticator authenticator = createAuthenticator();

        RoleAuthorizer authorizer = createRoleAuthorizer();

        Router router = createRouter();


        Router baseRouter = new Router(getContext());


        //Protect the resource by enforcing authentication then authorization
```

```
    authorizer.setNext(Resource0.class);
    authenticator.setNext(baseRouter);

    //Protect only the private resources with authorizer
    //You could use several different authorizers to authorize different roles
    baseRouter.attach("/resourceTypePrivate", authorizer);
    baseRouter.attach("/resourceTypePublic", router);
    return authenticator;
}

private ChallengeAuthenticator createAuthenticator() {
    ChallengeAuthenticator guard = new ChallengeAuthenticator(
            getContext(), ChallengeScheme.HTTP_BASIC, "realm");

    //Create in-memory users with roles
    MemoryRealm realm = new MemoryRealm();
    User user = new User("user", "user");
    realm.getUsers().add(user);
    realm.map(user, Role.get(this, ROLE_USER));
    User owner = new User("owner", "owner");
    realm.getUsers().add(owner);
    realm.map(owner, Role.get(this, ROLE_OWNER));

    //Attach verifier to check authentication and enroler to determine roles
    guard.setVerifier(realm.getVerifier());
    guard.setEnroler(realm.getEnroler());
    return guard;
}
```

```
    private RoleAuthorizer createRoleAuthorizer() {
        //Authorize owners and forbid users on roleAuth's children
        RoleAuthorizer roleAuth = new RoleAuthorizer();
        roleAuth.getAuthorizedRoles().add(Role.get(this, ROLE_OWNER));
        roleAuth.getForbiddenRoles().add(Role.get(this, ROLE_USER));
        return roleAuth;
    }

    private Router createRouter() {
        //Attach Server Resources to given URL
        Router router = new Router(getContext());
        router.attach("/resource1/", Resource1.class);
        router.attach("/resource2/", Resource2.class);
        return router;
    }

    public static void main(String[] args) throws Exception {
        //Attach application to http://localhost:9000/v1
        Component c = new Component();
        c.getServers().add(Protocol.HTTP, 9000);
        c.getDefaultHost().attach("/v1", new MyApiWithRoleAuthorization());
        c.start();
    }
}
```

Resources classes, call them Resource1, Resource2 etc... and copy-paste their content from here:

```
public class Resource0 extends ServerResource{
```

```
    @Get
    public String represent() throws Exception {
        return this.getClass().getSimpleName() + " found !";
    }

    @Post
    public String add() {
        return this.getClass().getSimpleName() + " posted !";
    }

    @Put
    public String change() {
        return this.getClass().getSimpleName() + " changed !";
    }

    @Patch
    public String partiallyChange() {
        return this.getClass().getSimpleName() + " partially changed !";
    }

    @Delete
    public String destroy() {
        return this.getClass().getSimpleName() + " deleted!";
    }
}
```

Main class for method authorization example, use the last class and replace its createInboundRoot and createRoleAuthorizer with the methods below:

```
@Override
public Restlet createInboundRoot() {
    //ChallengeAuthenticator
    ChallengeAuthenticator ca = createAuthenticator();
    ca.setOptional(true);

    //MethodAuthorizer
    MethodAuthorizer ma = createMethodAuthorizer();
    ca.setNext(ma);

    //Router
    ma.setNext(createRouter());
    return ca;
}

private MethodAuthorizer createMethodAuthorizer() {
    //Authorize GET for anonymous users and GET, POST, PUT, DELETE for
    //authenticated users
    MethodAuthorizer methodAuth = new MethodAuthorizer();
    methodAuth.getAnonymousMethods().add(Method.GET);
    methodAuth.getAuthenticatedMethods().add(Method.GET);
    methodAuth.getAuthenticatedMethods().add(Method.POST);
    methodAuth.getAuthenticatedMethods().add(Method.PUT);
    methodAuth.getAuthenticatedMethods().add(Method.DELETE);
    return methodAuth;
}
```

Fine-grained authorization

If your authorization rules tend to be very specific to each resource class, or more complexly specific to resource instances, then it is

better to handle them at the resource level manually. As an help, you can leverage the ServerResource#isInRole() method.

Create a simple server as below:

```
public class ApiWithFineGrainedAuthorization extends Application {


    @Override
    public org.restlet.Restlet createInboundRoot() {
        Router root = new Router(getContext());
        root.attach("/resource1", ResourceFineGrained.class);
        return root;
    }


    public static void main(String[] args) throws Exception {
        //Attach application to http://localhost:9000/v1
        Component c = new Component();
        c.getServers().add(Protocol.HTTP, 9000);
        c.getDefaultHost().attach("/v1", new MyApiWithMethodAuthorization());
        c.start();
    }
}
```

With a resource like this:

```
public class ResourceFineGrained extends ServerResource {


    @Get
    public String represent() throws ResourceException {
        if (!getRequest().getProtocol().equals(Protocol.HTTPS)) {
            throw new ResourceException(new Status(426, "Upgrade required",
"You should switch to HTTPS", ""));
        }
```

```
        return this.getClass().getSimpleName() + " found !";
    }

    @Post
    public String postMe() {
        if (!isInRole(MyApiWithRoleAuthorization.ROLE_OWNER)) {
            throw new ResourceException(Status.CLIENT_ERROR_FORBIDDEN);
        }
        return this.getClass().getSimpleName() + " posted !";
    }

    @Patch
    public String patchMe() {
        if (!getClientInfo().isAuthenticated()) {
            throw new
ResourceException(Status.CLIENT_ERROR_METHOD_NOT_ALLOWED);
        }
        return this.getClass().getSimpleName() + " patched !";
    }
}
```

For a call to http://localhost:9000/v1/resourceTypePublic/resource1/ you will need to use the owner profile to use POST and just authenticate to use PATCH. You will need to use an HTTPS endpoint to GET this resource.

Middle-grained authorization

The idea would be to define several URI subspaces in your overall application URI space. Each subspace would have a unique name, like a role name. Then in LDAP you could assign permissions to roles for each subspace, not duplicating URI information in LDAP.

Then, in your Restlet application, you could have a SubspaceAuthorizer class that would retrieve the rules from LDAP based on a given subspace name. An instance of SubspaceAuthorizer for each identified

subspace should be created and attached to the proper place in the routing graph.

Therefore you wouldn't need to duplicate URI matching code or have to maintain URI templates in several places. Configuring HTTPS
==========

Background

HTTPS is HTTP over SSL or TLS, that is, secure HTTP. SSL and TLS normally rely on Public Key Cryptography, which requires a pair of ***keys*** to encrypt and decrypt messages – one key is a ***private key*** and the other is a ***public key***. As the names imply, a private key is only known to the person/computer that created it, while the public key is made available to everyone. The public key and the private key complement each other. There are two main operations that can be performed: ***encryption*** and ***signing***. Encryption is used to make the content of a message secret and is done using the public key; decryption can then only be performed using the corresponding private key. Signing is used to assert the authenticity of a message and involves similar operations, but is done using the private key; then, validating the signature is done using the corresponding public key.

The distribution of the public keys and their binding to an identity is done by the ***Public Key Infrastructure (PKI)***, which uses certificates. A ***certificate* (usually X.509) is essentially a signed statement that includes a public key and other information such as date of validity, subject distinguished name (which associates an identity with this) or the issuer distinguished name (which gives the identity of the signer). The signer of such a certificate is usually a*certificate authority (CA)**, which may be a company such as Thawte or VeriSign, who may charge you for this service, or can be a local CA created within your institution or company. PKIs describe the relationships and trust models between the CAs and are associated with legal documents describing the intended use of various X.509 attributes (depending on CA policies).

However, certificates can also be ***self-signed***, in which case the trust model has to be established by some other means (for example, someone you trust gives you this certificate). Self-certification is fine for testing in a limited and controlled environment, but is not generally suitable for production environments.

In HTTPS, HTTP lies on top of SSL (or TLS). The client first establishes the SSL connection (which is not aware of HTTP): the client initiates a conversation with a server. As part of the SSL handshake, the server first sends its certificate to the client as proof that the server is who/what it claims to be. This certificate is checked by the client

against a list of certificates that it 'trusts'. How does a client know which certificates to trust? It depends on the client. A Java program uses a trust store (which is by default the 'cacerts' file within the JRE and which comes with a pre-populated list of trusted certificates/certification authorities); what the trust store is may be configured in Java. Similarly, browsers may maintain their own lists or – in the case of Windows or OSX – may use a feature of the operating system (e.g. Internet Options or Keychain). You can add your own certificates to these lists, or tell the Java VM, browser and/or your client operating system to trust the certificates you have created. If as part of this SSL handshake, the server certificate is not verified and trusted by the client, the SSL connection will be closed before any HTTP traffic.

When connecting to a server, not only the certificate should be verified, but the identity of the host name of the server should be checked against the certificate. Clients check that they are indeed communicating with the server they intend by checking that the ***common name (CN)*** field of the subject distinguished name in the server certificate is the host name intended. This host name can also be placed in the subject alternative name attribute in the certificate.

A Java container of keys and certificates is called a ***keystore***. There are two usages for keystores: as a ***keystore*** (the name may be confusing indeed) and as a ***truststore***. The keystore contains the material of the local entity, that is the private key and certificate that will be used to connect to the remote entity (necessary on a server, but can be used on a client for client-side authentication). Its counterpart, the truststore, contains the certificates that should be used to check the authenticity of the remote entity's certificates. There are various keystore types supported by the Sun JVM, mainly Java's own JKS format (default) or the PKCS#12 format which tends to be used by browsers and tools such as OpenSSL.

Step 1 : Creating Keys and a Self-Signed Certificate

Java comes with a command-line utility called ***keytool*** that can be used to handle keystores, in particular to create keys, certificate requests (CSR) and self-signed certificates (there are other utilities, such as ***OpenSSL*** and ***KeyMan*** from IBM). More precisely, keytool will create a ***keystore*** file that can contain one or more key pairs and certificates (we will only create one of each). By default, when you create a pair of keys, keytool will also create a self-signed certificate. The most important thing in this step is that you correctly specify the name of the machine where the certificate will be used as the common name, in the '-dname' option (see below). In the following example, the machine is called 'serverX' (the command-line options are put onto separate lines for readability only):

keytool -genkey

-v

-alias serverX

-dname "CN=**serverX**,OU=IT,O=JPC,C=GB"

-keypass password

-keystore serverX.jks

-storepass password

-keyalg "RSA"

-sigalg "MD5withRSA"

-keysize 2048

-validity 3650

The output should be:

Generating 2,048 bit RSA key pair and self-signed certificate (MD5withRSA) with

a validity of 3,650 days

for: CN=serverX, OU=IT, O=JPC, C=GB

[Storing serverX.jks]

To explain each option:

Option

-genkey

-v

-alias

-dname

-keypass

-keystore

-storepass

-keyalg

-sigalg

-keysize

-validity

Keystore files can have different formats. The example above, the default format of “JKS” (Java Key Store) was used. The type for PKCS#12 files (.p12) is "PKCS12", which can be specified by adding the following options to the keytool command line:

-storetype “PKCS12”

Step 2: Exporting the Self-Signed Certificate

The file 'serverX.jks' now contains the keys and a self-signed certificate. To be useful, the certificate needs to be exported so that it can be imported into other keystores such as those used by the Java VM or Windows. To export the certificate, use keytool with the following options:

keytool -export

-v

-alias serverX

-file serverX.cer

-keystore serverX.jks

-storepass password

The output should be:

Certificate stored in file <serverX.cer>

To explain each option:

Option

-export

-alias

-file

-keystore

-storepass

Note that the name of the alias and the keystore/certificate files are not significant, but they are named 'serverX' for consistency and clarity.

Step 3: Importing the Self-Signed Certificate

You should now have a file called 'serverX.cer' that contains your server's self-signed certificate. The server will present this certificate whenever an HTTPS client sends a request. There are different ways of installing the certificate on the server; in the Restlet example server code below, the certificate is loaded from the keystore when the Restlet server is started.

There are different ways to get a HTTPS client to trust your certificate. If you are using a browser, there may be an option to add it to a list of trusted certificates. In Windows XP, the certificate can be added to the 'Trusted Root Certification Authorities' via Internet Options (in IE7 or Control Panel - Internet Options). On the 'Content' tab, click 'Certificates', then go to 'Trusted Root Certification Authorities' tab, click 'Import...' and follow the steps to import your certificate file ('serverX.cer'). It will give warnings about not being verified, which is ok for testing, but it must be properly signed by proper Certification Authority for production. Firefox 3 also has the ability to add exceptions to trust individual certificates (self-signed or issued by an unknown CA).

If you are using another Java program instead of a browser, then you need to let the Java VM know about the certificate. There are several ways to do this, but here are two:

1. Import the certificate to the Java VM trusted certificates file, which is called 'cacerts' by default and located in the *lib/security* directory of the Java home directory, for example *C:\Program Files\Java\jre6\lib\security\cacerts*

The keytool command to do this is:

keytool -import

-alias serverX

-file serverX.cer

-keystore "C:\Program Files\Java\jre6\lib\security\cacerts"

-storepass "changeit"

Note that the default password for the cacerts keystore file is 'changeit'.

2. Add the following Java VM arguments to your Java client command line:

-Djavax.net.ssl.trustStore=C:\\somedir\\serverX.jks

-Djavax.net.ssl.trustStoreType=JKS

-Djavax.net.ssl.trustStorePassword=password

These arguments tell the Java VM where to find your certificate. **Please note that this approach should only be used in a test environment, not in production, as the password is shown in plain text**.****

Step 4: Sample Restlet Server Code

In addition to the standard Restlet jar files, you also need to reference jar files for HTTPS. The 'Simple' HTTPS connector uses these jar files:

lib/org.restlet.ext.simple_3.1.jar

lib/org.simpleframework_3.1/org.simpleframework.jar
lib/org.restlet.ext.ssl.jar

lib/org.jsslutils_0.5/org.jsslutils.jar

The server code in this example will explicitly load the certificate from the keystore file (serverX.jks):

```
package com.jpc.samples;


import org.restlet.Component;

import org.restlet.Server;

import org.restlet.data.Parameter;

import org.restlet.data.Protocol;

import org.restlet.util.Series;


public class SampleServer {

    public static void main(String[] args) throws Exception {


        // Create a new Component.

        Component component = new Component();
```

```
        // Add a new HTTPS server listening on port 8183
        Server server = component.getServers().add(Protocol.HTTPS, 8183);
        Series\<Parameter\> parameters =
server.getContext().getParameters();
        parameters.add("sslContextFactory",
        "org.restlet.ext.ssl.PkixSslContextFactory");
        parameters.add("keystorePath", "\<path\>serverX.jks");
        parameters.add("keystorePassword", "password");
        parameters.add("keyPassword", "password");
        parameters.add("keystoreType", "JKS");

        // Attach the sample application.
        component.getDefaultHost().attach("", new SampleApplication());

        // Start the component.
        component.start();
    }
}
```

Step 5: Sample Restlet Client Code

```
package com.jpc.samples;

import java.io.IOException;
import org.restlet.Client;
import org.restlet.data.Form;
import org.restlet.data.Protocol;
import org.restlet.data.Reference;
import org.restlet.data.Response;
import org.restlet.resource.Representation;
```

```
public class SampleClient {

    public static void main(String[] args) throws IOException {
        // Define our Restlet HTTPS client.
        Client client = new Client(Protocol.HTTPS);

        // The URI of the resource "list of items".
        Reference samplesUri = new Reference("https://serverX:8183/sample";);

        // Create 9 new items
        for (int i = 1; i \< 10; i++) {
            Sample sample = new Sample(Integer.toString(i), "sample " + i, "this
is
                sample " + i + ".");
            Reference sampleUri = createSample(sample, client, samplesUri);
            if (sampleUri != null) {
                // Prints the representation of the newly created resource
                get(client, sampleUri);
            }
        }

        // Prints the list of registered items
        get(client, samplesUri);
    }
}
```

...other code not shown (similar to original HTTP Restlet example)...

Conclusion

That's it! Your client and server should now be communicating via HTTPS. Note that the information contained in this article is meant only as rudimentary starting point for using HTTPS with Restlet and not as a comprehensive guide to secure web applications. Comments and feedback welcome!

Tunnel service

Introduction

This service tunnels request method or client preferences. The tunneling can use query parameters and file-like extensions. This is particularly useful for browser-based applications that can't fully control the HTTP requests sent.

Here is the list of the default parameter names supported:

Property

Default name

Value type

Description

methodParameter

method

See values in Method

For POST requests, let you specify the actual method to use (DELETE, PUT, MOVE, etc.).

For GET requests, let you specify OPTIONS as the actual method to use.

characterSetParameter

charset

Use extension names defined in MetadataService

For GET requests, replaces the accepted character set by the given value.

encodingParameter

encoding

Use extension names defined in MetadataService

For GET requests, replaces the accepted encoding by the given value.

languageParameter

language

Use extension names defined in MetadataService

For GET requests, replaces the accepted language by the given value.

mediaTypeParameter

media

Use extension names defined in MetadataService

For GET requests, replaces the accepted media type set by the given value.

The client preferences can also be updated based on the extensions available in the last path segment. The syntax is similar to file extensions by allows several extensions to be present, in any particular order: e.g. "/path/foo.fr.txt"). This mechanism relies on the mapping between an extension and a metadata (e.g. "txt" => "text/plain") declared by the MetadataService.

The client preferences can also be updated according to the user agent properties (its name, version, operating system, or other) available via the ClientInfo.getAgentAttributes() method. The feature is based on a property-like file called "accept.properties" and loaded from the classpath. Here is an excerpt of such file :

agentName: firefox accept: application/xhtml+xml,text/html,text/xml;q=0.9,application/xml;q=0.9

It allows to specify a complete "accept" header string for a set of (key:value) pairs. The header value is given with the "accept" key, and the set of (key:value) pairs is the simple list of key:value just above the "accept" line.

Task service

Introduction

This service is capable of running tasks asynchronously. The service instance returned will not invoke the runnable task in the current thread.

In addition to allowing pooling, this method will ensure that the threads executing the tasks will have the thread local variables copied from the

calling thread. This will ensure that call to static methods like Application.getCurrent() still work.

Also, note that this executor service will be shared among all Restlets and Resources that are part of your context. In general this context corresponds to a parent Application's context. If you want to have your own service instance, you can use the wrap(ExecutorService) method to ensure that thread local variables are correctly set.

Metadata service

Introduction

This service provides access to metadata and their associated extension names.

Status service

Introduction

This service handles error statuses. If an exception is thrown within your application or Restlet code, it will be intercepted by this service if it is enabled.

When an exception or an error is caught, the getStatus(Throwable, Request, Response) method is first invoked to obtain the status that you want to set on the response. If this method isn't overridden or returns null, the Status.SERVER_ERROR_INTERNAL constant will be set by default.

Also, when the status of a response returned is an error status (see Status.isError(), the getRepresentation(Status, Request, Response) method is then invoked to give your service a chance to override the default error page.

If you want to customize the default behavior, you need to create a subclass of StatusService that overrides some or all of the methods mentioned above. Then, just create a instance of your class and set it on your Component or Application via the setStatusService() methods.

Display error pages

Another common requirement is the ability to customize the status pages returned when something didn't go as expected during the call handling. Maybe a resource was not found or an acceptable representation isn't available? In this case, or when any unhandled exception is be intercepted, the Application or the Component will automatically provide a default status page for you. This service is associated to the org.restlet.util.StatusService class, which is accessible as an Application and Component property called "statusService".

In order to customize the default messages, you will simply need to create a subclass of StatusService and override the getRepresentation(Status, Request, Response) method. Then just set an instance of your custom service to the appropriate "statusService" property.

ConverterService

Introduction

The converter service of an application handles the conversion between representations (received by a resource for example) and beans or POJOs. It can be used either programmatically or transparently thanks to annotated Restlet resources.

Description

Server-side usage

Let's describe how this service is typically used with a ServerResource that supports GET and PUT methods as below:

```
@Get("json")

// Returns a json representation of a contact.

public Contact toJson(){

  [...]

}


@Put()

// Handles a Web form in order to set the full state of the resource.

public void store(Form form){

  [...]

}
```

In this case, the aim of the converter service is to:

- return a JSON representation of a contact
- convert a Web form into a Form object and pass it to the store method.

Client-side usage

On the client-side you can leverage the service either by creating a proxy of an annotated resource interface, via ClientResource.wrap(...)

or ClientResource.create(...) methods. Otherwise, it is possible to invoke any web API and specify the excepted return type and let the service convert between beans and representations. Here is the list of related methods on ClientResource:

- delete(Class<T> resultClass) : T
- get(Class<T> resultClass) : T
- options(Class<T> resultClass) : T
- post(Object entity, Class<T> resultClass) : T
- put(Object entity, Class<T> resultClass) : T

For example if you want to retrieve an XML representation as a DOM document, you can just do:

```
ClientResource cr = new ClientResource("http://myapi.com/path/resource");

Document doc = cr.get(Document.class);
```

And that's all you need to do, as long as you have the org.restlet.ext.xml.jar in your classpath!

Internals of the service

A converter service does not contain the conversion logic itself. It leverages a set of declared converters which are subclasses of ConverterHelper. A converter is a piece of code that handles:

- either the conversion of a Representation to an object
- or the conversion of an object to a Representation
- or both conversion

Conversion mainly relies on the media type of the given Representation (application/json, text/xml, etc.) and an instance of a specific class. For example, the default converter provided by the core module (org.restlet.jar) allows the conversion of Web forms (media type "application/x-www-form-urlencoded") to org.restlet.data.Form instances and vice versa. The converter provided the FreeMarker extension is only able to generate Representations from a FreeMarker Template (class freemarker.template.Template).

A converter is declared using a simple text file located in the "META-INF/services" source directory. Its name is "org.restlet.engine.converter.ConverterHelper" and it contains generally a single line of text which is the full path of the converter class. For example, the FreeMarker extension contains in the

"src/META-INF/services" directory such text file with the following line of text: "org.restlet.ext.freemarker.FreemarkerConverter".

Available converters

Conversion from representations to objects

Module

Core

Core

Core

Core

Atom

Atom

GWT

Jackson

JAXB

JiBX

JSON

RDF

WADL

XML

XStream

Conversion from objects to representations

Module | From Object | To Representations with media type ------ | --------- | ---------- Core | java.lang.String, java.io.File, java.io.InputStream, java.io.Reader, StringRepresentation, FileRepresentation, InputStreamRepresentation, ReaderRepresentation, org.restlet.representation.Representation | any Core | org.restlet.Form | APPLICATION_WWW_FORM Core | java.io.Serializable | APPLICATION_JAVA_OBJECT, APPLICATION_JAVA_OBJECT_XML Atom | org.restlet.ext.atom.Feed | APPLICATION_ATOM Atom | org.restlet.ext.atom.Service | APPLICATION_ATOM_PUB FreeMarker | freemarker.template.Template | any GWT | an Object, org.restlet.ext.gwt.ObjectRepresentation |

APPLICATION_JAVA_OBJECT_GWT Jackson | an Object | APPLICATION_JSON JavaMail | a javax.mail.Message | a org.restlet.ext.javamail.MessageRepresentation JAXB | object supporting JAXB annotations, org.restlet.ext.JaxbRepresentation | APPLICATION_ALL_XML, APPLICATION_XML, TEXT_XML JiBX | JiBX bound object, org.restlet.ext.JibxRepresentation | APPLICATION_ALL_XML, APPLICATION_XML, TEXT_XML JSON | org.json.JSONArray, org.json.JSONObject, org.json.JSONTokener | APPLICATION_JSON RDF | org.restlet.ext.rdf.Graph | TEXT_RDF_N3, TEXT_RDF_NTRIPLES, APPLICATION_RDF_TURTLE, APPLICATION_ALL_XML ROME | com.sun.syndication..fedd.synd.SyndFeed | org.restlet.ext.rome.SyndFeedRepresentation Velocity | org.apache.velocity.Template | any WADL | org.restlet.ext.wadl.ApplicationInfo | APPLICATION_WADL XML | org.w3c.dom.Document, org.restlet.ext.xml.DomRepresentation, org.restlet.ext.xml.SaxRepresentation | APPLICATION_ALL_XML, APPLICATION_XML, TEXT_XML XStream | an object | APPLICATION_ALL_XML, APPLICATION_XML, TEXT_XML, APPLICATION_JSON Range service ==========

Introduction

This service automatically exposes ranges of response entities. This allows resources to not care of requested ranges and return full representations that will then be transparently wrapped in partial representations by this service, allowing the client to benefit from partial downloads.

Service package

Introduction

The **org.restlet.service** package contains services used by applications and components. This chapter lists the services hosted by default by Component and Application instances.

Component services

Here is the list of services hosted by default by an instance of Component:

- Log service: provide access to logging service.
- Status service: provide common representations for exception status.

Application services

Here is the list of services hosted by default by an instance of Application :

- Connector service: declare necessary client and server connectors.

- Decoder service: automatically decode or decompress request entities.
- Metadata service: provide access to metadata and their associated extension names.
- Range service: automatically exposes ranges of response entities.
- Status service: provide common representations for exception status.
- Task service: run tasks asynchronously.
- Tunnel service: tunnel method names or client preferences via query parameters.

Connector service

Introduction

This service declares client and server connectors. This is useful at deployment time to know which connectors an application expects to be able to use. Implementation note: the parent component will ensure that client connectors won't automatically follow redirections. This will ensure a consistent behavior and portability of applications.

Log service

Introduction

The log service provides access to logging service. The implementation is fully based on the standard logging mechanism introduced in JDK 1.4.

Being able to properly log the activity of a Web application is a common requirement. Restlet Components know by default how to generate Apache-like logs or even custom ones. By taking advantage of the logging facility built in the JDK, the logger can be configured like any standard JDK log to filter messages, reformat them or specify where to send them. Rotation of logs is also supported; see the java.util.logging package for details.

Note that you can customize the logger name given to the java.util.logging framework by modifying the Component's "logService" property. In order to fully configure the logging, you need to declare a configuration file by setting a system property like:

```
System.setProperty("java.util.logging.config.file",
"/your/path/logging.config");
```

For details on the configuration file format, please check the JDK's LogManager class. You can also have a look at the Restlet 2.3 logging documentation.

Default access log format

The default format follows the W3C Extended Log File Format with the following fields used:

1 Date (YYYY-MM-DD)

2 Time (HH:MM:SS)

3 Client address (IP)

4 Remote user identifier (see RFC 1413)

5 Server address (IP)

6 Server port

7 Method (GET|POST|...)

8 Resource reference path (including the leading slash)

9 Resource reference query (excluding the leading question mark)

10 Response status code

11 Number of bytes sent

12 Number of bytes received

13 Time to serve the request (in milliseconds)

14 Host reference

15 Client agent name

16 Referrer reference

If you use Analog to generate your log reports, and if you use the default log format, then you can simply specify this string as a value of the LOGFORMAT command: (%Y-%m-%d\t%h:%n:%j\t%S\t%u\t%j\t%j\t %j\t%r\t%q\t%c\t%b\t%j\t%T\t%v\t%B\t%f)

For custom access log format, see the syntax to use and the list of available variable names in Template.

Decoder service

Introduction

This service decodes or decompresses automatically request entities.

Part II - Core Restlet

Introduction
This chapter presents the Restlet API. It is a small set of packages, with classes, interfaces and enumerations that together provide a framework. This framework will guide you on the path to RESTful design and development. But be aware that you still need to understand REST in order to fully take advantage of the Restlet features. For this purpose, we recommend the book "RESTful Web Services" from O'Reilly. We even wrote a part of it covering Restlet usage.

For a more detailed presentation of the API, we recommend that you have a close look at its Javadocs available on the Restlet Web site. Below is a hierarchy diagram showing the main concepts of the API and their relationships:

Here is a diagram illustrating how the API composes components, connectors, virtual host and applications. Applications are in turn composed of resources.

http://restlet.org:8444 Util package ==========

Introduction
The **org.restlet.util** package contains various utility classes.

Client resources

Introduction
For a short introduction on the usage of client resource, you should read the first client page.

Automatic conversion
You can get automatic conversion for retrieved representations like this:

```
String myString = myClientResource.get(String.class);
```

For sent representations:

```
myClientResource.put(myString);
```

There is even a more transparent way if you define an annotated Java interface (using the Restlet @Get, @Put, etc. annotations). Once it is defined, you can use it on the server side in your ServerResource subclasses or on the client side to consume it:

```
  MyAnnotatedInterface myClient =
myClientResource.wrap(MyAnnotatedInterface.class);
```

In this case, automatic conversion is handled for you. By default, the Restlet Engine support Java object serialization (binary or XML), but for more interoperable representations, we suggest to add our Jackson extension or XStream extension to your classpath in order to get clean XML or JSON representations.

Leveraging annotations

It also possible to share an annotated Java interface between client and server resources.

Annotated Java interface

First, we define the contract between the client and the server as a Java interface:

```
package annos;


import org.restlet.resource.Delete;

import org.restlet.resource.Get;

import org.restlet.resource.Post;

import org.restlet.resource.Put;


import client.Customer;


public interface TestResource {


    @Get

    public Customer retrieve();


    @Put

    public void store(Customer customer);
```

```
    @Post
    public void stop() throws Exception;

    @Delete
    public void remove() throws Exception;

}
```

Server side implementation

Then, we implement is in a subclass of ServerResource

```
package annos;

import org.restlet.Context;
import org.restlet.Server;
import org.restlet.data.Protocol;
import org.restlet.resource.ServerResource;

import client.Customer;

public class TestServerResource extends ServerResource implements
TestResource {

    private static volatile Customer myCustomer = Customer.createSample();

    private static final Server server = new Server(Protocol.HTTP, 8182,
            TestServerResource.class);

    public static void main(String[] args) throws Exception {
        Context ctx = new Context();
```

```
        server.setContext(ctx);
        server.getContext().getParameters().add("keystorePassword",
"password");
        server.start();
    }

    public Customer retrieve() {
        System.out.println("GET request received");
        return myCustomer;
    }

    public void store(Customer customer) {
        System.out.println("PUT request received");
        myCustomer = customer;
    }

    public void stop() throws Exception {
        System.out.println("POST request received");
        server.stop();
    }

    public void remove() throws Exception {
        System.out.println("DELETE request received");
        myCustomer = null;
    }

}
```

Client-side consumption

Finally, we can consume it via the ClientResource class:

```
package annos;


import org.restlet.resource.ClientResource;


import client.Customer;


public class TestClientResource {


    public static void main(String[] args) throws Exception {
        ClientResource clientResource = new ClientResource(
            "http://localhost:8182/rest/test");
        TestResource testResource = clientResource.wrap(TestResource.class);


        // Retrieve the JSON value
        Customer result = testResource.retrieve();


        if (result != null) {
            System.out.println(result);
        }
    }
}
```

Consuming response entities

Even though the ClientResource class gives you a high level client for HTTP transactions, you need to be aware that the underlying network elements such as connections and sockets must be handled and reused carefully.

The best practice is to always fully consume the content of response entities, in case of success as well as in case of error. If you are not interested in this content, you should at least call the Representation#exhaust() method to silently consume its content.

If you forget to do this, the associated HTTP connection will stay active for a while and won't be reused or collected, potentially leading to starvation issues (unable to create new connections) or consuming too much resources (unable to collect the socket and threads blocked on this connection). In addition, once the whole content has been consumed (read or exhausted) it is recommended to manually release the representation and its associated objects used to produce its content (such as database connections) by calling its Representation#release() method.

In addition, if you are not interested in the entity content or by further calls from this client, you can try calling the Representation#release() method immediately which will close the underlying stream but might also close the underlying socket connection. In case you are not interested in the request at all, it is however better to explicitly call the ClientResource#abort() method.

Server resources

See the first server and first application examples.

Resource package

Introduction

The **org.restlet.resource** package contains client and server resource classes.

Architecture

Below is an overview of the architecture, including all processing layers, from the lowest TCP/IP network layer to the highest annotated interface proxies.

Class diagram

Here is a tentative class design:

Annotations

We also defined a set of method level annotations:

Annotation

@Delete

@Get

@Options

@Post

@Put

Those annotations are specific to the Restlet API and shouldn't be confused with those of the JAX-RS API. For support of the JAX-RS API by the Restlet Framework, you should look at the provided extension.

Annotations parameter

All annotation have a single optional parameter. Its name is the default "value" name allowing a compact annotation syntax.

Here is the grammar for this parameter:

```
CHARACTER  = 'a-z' | 'A-Z' | '0-9'
TOKEN      = CHARACTER [CHARACTER]*
EXTENSION  = TOKEN
PARAMETER  = TOKEN ['=' TOKEN]
VARIANT    = EXTENSION ['+' EXTENSION]*
ENTITY     = VARIANT ['|' VARIANT]*
INPUT      = ENTITY
OUTPUT     = ENTITY
QUERY      = PARAMETER ['&' PARAMETER]
ANNOTATION = INPUT [',' INPUT]* [':' OUTPUT] ['?' QUERY]
```

Here are some valid values:

```
// Returns a representation in the "text/xml" media type
@Get("xml")
String toString();


// Stores representations in the "text/xml" media type
// after conversion to a DOM document
@Put("xml")
```

```
void store(Document doc)


// Stores representations in the "text/xml" media type after
// conversion to a DOM document and returns a plain text response
@Put("xml:txt")
String store(Document doc)


// Returns a representation in the "text/xml" media type with
// an inlining depth level of 2
@Get("xml?depth=2")


// Alternative variants
@Put("xml|json:json")


// Alternative variants
@Put("xml+ascii | json+utf8 : json")
```

Note the importance of registering the proper extension names via the MetadataService in order to use additional extension names.

Sample code

Here is how a sample resource would look like with the refactored API. Note that both extension names and full MIME type would be supported. Extensions can be updated via the MetadataService.

```
import java.io.InputStream;


import org.restlet.ext.atom.Feed;
import org.restlet.resource.Delete;
import org.restlet.resource.Get;
import org.restlet.resource.Post;
import org.restlet.resource.Put;
import org.restlet.resource.Representation;
```

```
import org.restlet.resource.ServerResource;
import org.w3c.dom.Document;

public class TestResource extends ServerResource {

    @Get
    public Feed toAtom() {
        // ...
        return null;
    }

    @Get("xml")
    public Representation toXml() {
        // ...
        return null;
    }

    @Post("xml")
    public Representation accept(Document entity) {
        // ...
        return null;
    }

    @Put("atom")
    public void storeAtom(Feed feed) {
        // ...
    }
```

```
    @Put("cust")
    public void storeXml(InputStream stream) {
        // ...
    }

    @Delete
    public void removeAll() {
        // ...
    }

}
```

Getting parameter values

This is a common need to retrieve data, especially "key=value" pairs from the query, the entity, or the cookies. Here are some sample lines of code which illustrate this feature.

Getting values from a web form

The web form is in fact the entity of the POST request sent to the server, thus you have access to it via request.getEntity(). There is a shortcut which allows to have a list of all input fields:

```
Form form = new Form(request.getEntity())

for (Parameter parameter : form) {
    System.out.print("parameter " + parameter.getName());
    System.out.println("/" + parameter.getValue());
}
```

Getting values from a query

The query is a part of the identifier (the URI) of the request resource. Thus, you have access to it via request.getResourceRef().getQuery(). There is a shortcut which allows to have a list of all "key=value" pairs:

```
Form form = request.getResourceRef().getQueryAsForm();

for (Parameter parameter : form) {
    System.out.print("parameter " + parameter.getName());
    System.out.println("/" + parameter.getValue());
```

```
}
```

Getting values from the cookies

Cookies are directly available from the request via the request.getCookies() method. It returns a collection of "Cookie" objects which extends the Parameter class and contains additional attributes:

```
for (Cookie cookie : request.getCookies()) {

    System.out.println("name = " + cookie.getName());

    System.out.println("value = " + cookie.getValue());

    System.out.println("domain = " + cookie.getDomain());

    System.out.println("path = " + cookie.getPath());

    System.out.println("version = " + cookie.getVersion());

}
```

Data package

The **org.restlet.data** package contains Information exchanged between components distributed over the internet. In particular, it contains Java classes mapping most HTTP headers. The semantics of each HTTP header is surfaces as a simple Java class or enumeration.

In addition, this document contains instructions to get parameters data out of: - web form sumbitted - URI query parameters - browser cookies URI rewriting and redirection
==========

Introduction

The first tool available is the Redirector, which allows the rewriting of a cool URI to another URI, followed by an automatic redirection. Several types of redirection are supported, the external redirection via the client/browser and the connector redirection for proxy-like behavior. In the example below, we will define a search service for our web site (named "mysite.org") based on Google. The "/search" relative URI identifies the search service, accepting some keywords via the "kwd" parameter:

```
// Create a root router

Router router = new Router(getContext());



// Create a Redirector to Google search service
```

```
String target = "http://www.google.com/search?q=site:mysite.org+
{keywords}";

Redirector redirector = new Redirector(getContext(), target,

Redirector.MODE_CLIENT_TEMPORARY);



// While routing requests to the redirector, extract the "kwd" query

// parameter. For instance :

// http://localhost:8182/search?kwd=myKeyword1+myKeyword2

// will be routed to

// http://www.google.com/search?
q=site:mysite.org+myKeyword1%20myKeyword2

Extractor extractor = new Extractor(getContext(), redirector);

extractor.extractFromQuery("keywords", "kwd", true);



// Attach the extractor to the router

router.attach("/search", extractor);
```

Note that the Redirector needs three parameters only. The first is the parent context, the second one defines how the URI rewriting should be done, based on a URI template. This template will be processed by the Template class. The third parameter defines the type of redirection; here we chose the client redirection, for simplicity purpose.

Also, we are relying on the Route class to extract the query parameter "kwd" from the initial request while the call is routed to the application. If the parameter is found, it is copied into the request attribute named "keywords", ready to be used by the Redirector when formatting its target URIs.

Routers and hierarchical URIs

Introduction

In addition to the Redirector, we have another tool to manage cool URIs: Routers. They are specialized Restlets that can have other Restlets (Finders and Filters for example) attached to them and that can automatically delegate calls based on a URI template. In general, you will set a Router as the root of your Application.

Here we want to explain how to handle the following URI patterns:

1 /docs/ to display static files

2 /users/{user} to display a user account

3 /users/{user}/orders to display the orders of a particular user

4 /users/{user}/orders/{order} to display a specific order

The fact that these URIs contain variable parts (between accolades) and that no file extension is used makes it harder to handle them in a typical Web container. Here, you just need to attach target Restlets to a Router using the URI template. At runtime, the route that best matches the request URI will received the call and be able to invoke its attached Restlet. At the same time, the request's attributes map will be automatically updated with the value of the URI template variables!

See the implementation code below. In a real application, you will probably want to create separate subclasses instead of the anonymous ones we use here:

```
    // Create a root router
    Router router = new Router(getContext());

    // Create a simple password verifier
    MapVerifier verifier = new MapVerifier();
    verifier.getLocalSecrets().put("scott", "tiger".toCharArray());

    // Create a Guard
    // Attach a guard to secure access to the directory
    ChallengeAuthenticator guard = new
ChallengeAuthenticator(getContext(),
            ChallengeScheme.HTTP_BASIC, "Tutorial");
    guard.setVerifier(verifier);
    router.attach("/docs/", guard).setMatchingMode(
            Template.MODE_STARTS_WITH);
```

```
// Create a directory able to expose a hierarchy of files
Directory directory = new Directory(getContext(), ROOT_URI);
guard.setNext(directory);

// Create the account handler
Restlet account = new Restlet() {
    @Override
    public void handle(Request request, Response response) {
        // Print the requested URI path
        String message = "Account of user \""
                + request.getAttributes().get("user") + "\"";
        response.setEntity(message, MediaType.TEXT_PLAIN);
    }
};

// Create the orders handler
Restlet orders = new Restlet(getContext()) {
    @Override
    public void handle(Request request, Response response) {
        // Print the user name of the requested orders
        String message = "Orders of user \""
                + request.getAttributes().get("user") + "\"";
        response.setEntity(message, MediaType.TEXT_PLAIN);
    }
};

// Create the order handler
```

```
Restlet order = new Restlet(getContext()) {
    @Override
    public void handle(Request request, Response response) {
        // Print the user name of the requested orders
        String message = "Order \""
            + request.getAttributes().get("order")
            + "\" for user \""
            + request.getAttributes().get("user") + "\"";
        response.setEntity(message, MediaType.TEXT_PLAIN);
    }
};


// Attach the handlers to the root router
router.attach("/users/{user}", account);
router.attach("/users/{user}/orders", orders);
router.attach("/users/{user}/orders/{order}", order);
```

Note that the routing assumes that your request contains an absolute target URI that identifies a target resource. During the request processing the resource's base URI is continuously updated, for each level in the hierarchy of routers. This explains why the default behavior of routers is to match only the beginning of the remaining URI part and not the totality of it. In some cases, you might want to change this default mode, and this is easy to do via the "defaultMatchingMode" property on Router, or by modifying the "matchingMode" property of the template associated with the route created by the Router.attach() methods. For the modes, you can use the Template.MODE_EQUALS or Template.MODE_STARTS_WITH constants.

Please note that the values of the variables are directly extracted from the URI and are therefore not percent-decoded. In order to achieve such a task, have a look to the Reference#decode(String) method.

Routing package

Introduction

The **org.restlet.routing** package contains classes related to call routing.

Another advantage of the Restlet Framework is the built-in support for cool URIs. A good description of the importance of proper URI design is given by Jacob Nielsen in his AlertBox. This document aims at showing the several configuration options of the routing process. From a high level point of view, routing is mainly based on the Router, Route, Template and Variable classes.

A Router is a Restlet that helps to associate a URI to the Restlet or the Resource that will handle all requests made to this URI.

The association of a URI and the target Restlet or Resource is personified by an instance of the Route class. In general, people wants to express generic associations such as: all URIs "/my/base/uri/{variable part}" target this Restlet or Resource. Hence, the Restlet framework allows to define routes based on the URI template specification.

URI Templates allows to defines pattern of URIs that contain embedded variables. These concepts have been formalised in the Restlet famework under the form of Template and Variable classes.

Here is a sample code illustrating the default usage of a Router:

```
// Create a router Restlet that defines routes.

Router router = new Router(getContext());



// Defines a route for the resource "list of items"

router.attach("/items", ItemsResource.class);

// Defines a route for the resource "item"

router.attach("/items/{itemName}", ItemResource.class);
```

It declares two routes, the first one associates the URI "/items" to the Resource "ItemsResource" and the second one "/items/itemName" to the Resource "ItemResource". The latter defines a variable called "itemName" which can be retrieved by the Resource.

Please note that a route attaches instances of a Restlet subclass whereas resources are attached via their class. The reason is that Resources have been designed to handle a single request, thus instances are generated at run-time via their class.

Default configuration

This chapter covers either the default behavior of instances of the Router, Route, Template and Variable classes or the behavior that applies by default during the process of routing a URI.

Matching mode

The default behavior of a Router is to match the whole request URI (actually the remaining part, left by previous Routers) with the attached URI patterns. If the request URI indeed is equal to one of the URI patterns, the target Resource or Restlet is invoked. The way the matching is done by default is based on the Template#MODE_EQUALS constant, meaning that the URI template must exactly match the remaining part. This is a strict mode that forbids subsequent routing.

This default routing mode can be changed by calling the Router#setDefaultMatchingMode() method with the Template#MODE_STARTS_WITH constant meaning that if the URI template associated to the route matches the start of the target resource URI, then it is accepted. This is a flexible mode that allows a hierarchy of routers to be used.

Here is a sample code:

```
router.setMatchingMode(Template.MODE_STARTS_WITH);
```

Routing and queries

With the default matching mode to Template#MODE_EQUALS, your Router won't accept URI with query strings going to this Resource. To solve this, you can either indicate at the router level, that the matching process does not take into account the query part of the reference:

```
router.setDefaultMatchQuery(false);
```

Or you can explicitly allow the query part in your URI template:

```
TemplateRoute route = router.attach("/users/{user}?{query}",
UserResource.class);

route.setMatchingQuery(false);
```

Routing mode

In addition to the matching mode, a Router supports several routing modes, implementing various algorithms:

- Best match
- First match (default since Restlet 2.0)
- Last match

- Random match
- Round robin
- Custom

By default, when considering a URI to route, a Router computes an affinity score for each declared route then choose the one that have the best affinity score. You can update this behavior by setting the routing mode property with one of the listed modes declared above.

```
router.setRoutingMode(Router.FIRST);
```

Matching of template variables

As said in the introduction, the routing feature relies on the declaration of templates of URI. In the sample code below, the URI template declares one variable called "user":

```
TemplateRoute route = router.attach("/users/{user}", UserResource.class);
```

At run-time, the value of this variable is put into the attributes of the request:

```
this.userId = (String) getRequest().getAttributes().get("user");
```

By default a variable is meant to match a URI segment as defined in the URI specification document. That is to say a sequence of characters between two "/".

A variable holds several other attributes:

- defaultValue: default value to use if the key couldn't be found in the model ("" by default)..
- required: indicates if the variable is required or optional (true by default).
- fixed: indicates if the value is fixed, in which case, the default property is always used (false by default).
- decodedOnParse: indicates if the parsed value must be decoded (false by default).

Tuning the routing

Sometimes the same matching mode can't be used for all routes. For example one delegates the routing to a child router while others routes are resources directly processing the request. In this case, it is possible to specify a different routing mode for each route using the Route#setMatchingMode() method. The Route instance is returned by the Router#attach() methods.

Matching mode

Regarding the matching mode, the default behavior of a Router can be customized for a single route by updating its underlying template instance as follow:

```
TemplateRoute route = router.attach("/users/{user}", UserResource.class);

route.getTemplate().setMatchingMode(Template.MODE_STARTS_WITH);
```

Routing and queries

If you want to route URIs according to some values located in the query part of the URI, and since those values have no determined place within the query, you cannot rely on the URI template based routing feature which is more or less based on regular expressions.

In this case you have to define your own router (with the "CUSTOM" routing mode). A sample implementation is available here (application/zip, 1.8 kB).

Matching of template variables

By default, as said above, a template variable is meant to match a URI segment. Here is the list of all available type of variables. For additional details, see the Javadocs of the Variable class:

Value

TYPE_ALL

TYPE_ALPHA

TYPE_ALPHA_DIGIT

TYPE_COMMENT

TYPE_COMMENT_ATTRIBUTE

TYPE_DIGIT

TYPE_TOKEN

TYPE_URI_ALL

TYPE_URI_FRAGMENT

TYPE_URI_PATH

TYPE_URI_QUERY

TYPE_URI_SCHEME

TYPE_URI_SEGMENT

TYPE_URI_UNRESERVED

TYPE_WORD

Here is the way to change the type of a variable:

```
TemplateRoute route = router.attach("/items/{itemName}", ItemResource);

Map<String, Variable> routeVariables = route.getTemplate().getVariables();

routeVariables.put("itemName", new Variable(Variable.TYPE_URI_WORD));
```

Tutorial

Introduction

Do you have a part of your web application that serves static pages like Javadocs? Well, no need to setup an Apache server just for that, use instead the dedicated org.restlet.resource.Directory class. See how simple it is to use it:

```
public class FileServer extends Restlet {

    // URI of the root directory.

    public static final String ROOT_URI = "file:///c:/path/to/root";


    public static void main(String[] args) throws Exception {


        // Create a component

        Component component = new Component();

        component.getServers().add(Protocol.HTTP, 8182);

        component.getClients().add(Protocol.FILE);


        // Create an application

        Application application = new Application() {

            @Override

            public Restlet createInboundRoot() {

                return new Directory(getContext(), ROOT_URI);

            }
```

```
        };


        // Attach the application to the component and start it
        component.getDefaultHost().attach(application);
        component.start();
    }
}
```

In order to run this example, you need to specify a valid value for ROOT_URI, In this case, it is set to "file:///c:/path/to/root/".Note that no additional configuration is needed. If you want to customize the mapping between file extensions and metadata (media type, language or encoding) or if you want to specify a different index name, you can use the Application's "metadataService" property.

Extracting attributes

Introduction

Extracting values from query, entity, cookies into the request's attributesis a common need that is supported by the Extractor class.

When routing URIs to Resource instances or Restlet, you can decide to transfer data from query, entity or cookies into the request's list of attributes.

This mechanism is declarative and has been implemented at the level of the "routes" (see the code below for more information). This mechanism transfers data from the query (method "extractQuery"), the entity (method "extractEntity") or the Cookies (method "extractCookies") to the request's list of parameters.

For example, when implementing your Application, assuming that the posted web form contains a select input field (called "selectField") and a text field (called "textField"). If you want to transfer them respectively to attributes named "selectAttribute" and "textAttribute", just proceed as follow.

```
@Override
public Restlet createInboundRoot() {
    Extractor extractor = new Extractor(getContext());
    extractor.extractFromEntity("selectAttribute", "selectField", true);
    extractor.extractFromEntity("textAttribute", "textField", false);
```

```
    extractor.setNext(...)


    return extractor;
}
```

You will get a String value in the "selectAttribute" (the selected option), and a Form object which is a collection of key/value pairs (key="textField" in this case) with the "textAttribute" attribute.

Here is sample code which helps to retrieve some attributes:

```
// Get the map of request attributes
Map<String, Object> map = request.getAttributes();


// Retrieve the "selectAttribute" value
String stringValue = (String) map.get("selectAttribute");
System.out.println(" value => " + stringValue);


// Retrieve the "textAttribute" collection of parameters
Object object = map.get("textAttribute");
if(object != null){
   if(object instanceof Form){
      Form form = (Form) object;
      for (Parameter parameter : form) {
         System.out.print("parameter " + parameter.getName());
         System.out.println("/ " + parameter.getValue());
      }
   }
}
```

Mapping HTTP headers

Introduction

The Restlet API offers a higher level view of the HTTP protocol. It tries to abstract and present in a clean object model, the application-level semantics of HTTP. As a side feature, it is possible to map other protocols to the same semantics, such as FILE, FTP, SMTP, etc.

However, developers often know lower-level details of HTTP or need to understand them for debugging purpose. This is the reason for this document, explaining the mapping between the HTTP semantics and the Restlet API.

Note that the headers that are not supported yet can still be overridden via the "org.restlet.http.headers" attribute of the request or the response. A warning message is logged : "Addition of the standard header "XXX" is discouraged. Future versions of the Restlet API will directly support it".

See details in the Javadocs of the Message.getAttributes() method.

From HTTP headers to Restlet API

HTTP header

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Accept-Ranges

Age

Allow

Authentication-Info

Authorization

Cache-Control

Connection

Content-Disposition

Content-Encoding

Content-Language

Content-Length

Content-Location

Content-MD5

Content-Range

Content-Type

Cookie

Date

ETag

Expect

Expires

From

Host

If-Match

If-Modified-Since

If-None-Match

If-Range

If-Unmodified-Since

Last-Modified

Location

Max-Forwards

Pragma

Proxy-Authenticate

Proxy-Authorization

Range

Referer

Retry-After

Server

Set-Cookie

Set-Cookie2

TE

Trailer

Transfer-Encoding

Upgrade

User-Agent

Vary

Via

Warning

WWW-Authenticate

X-Forwarded-For

X-HTTP-Method-Override

Appendix

Registry of headers maintained by IANA. Regsitry of HTTP status codes maintained by IANA.

- GSON
- Guice
- GWT
- HTML
- HTTP Client
- JAAS
- Jackson
- JavaMail
- JAXB
- JAX-RS
- JDBC
- Jetty
- JiBX
- JSON
- jSSLutils
- Lucene
- Nio
- OAuth
- OData
- OpenID
- OSGi
- RDF
- ROME
- SDC
- Servlet
- Simple
- SIP

Introduction

Google provides a Java version of his App Engine solution (GAE). It is a PaaS (Platform as a Service) solution that offers massive and flexible scalability for your Web applications by hosting them on the Google cloud (based on Google computing infrastructure). For more details, you can read our blog post with the official announce.

Due to the restrictions of the GAE, we need to provide an adaptation of Restlet for this environment. GAE is based on Java 6, with a restricted list of APIs. See GAE developers documentation for details.

Modules availables:

- Restlet core (API + Engine)
- Restlet extension - Javamail (SMTP client)
- Restlet extension - Net (HTTP and HTTPS client)
- Restlet extension - Servlet (HTTP and HTTPS server)
- Restlet extension - XML (DOM, SAX and XPath)

Usage example

Create a new GAE project with the Eclipse plugin provided, add the "org.restlet.jar" and the "org.restlet.ext.servlet.jar" files from the latest Restlet snapshots(make sure you download the edition for GAE) to your "/war/WEB-INF/lib/" directory and to your project build path.

Here is the Restlet resource to create:

```
package firstSteps;


import org.restlet.resource.Get;

import org.restlet.resource.ServerResource;


/**
 * Resource which has only one representation.
 *
 */
public class HelloWorldResource extends ServerResource {


    @Get
    public String represent() {
        return "hello, world (from the cloud!)";
    }


}
```

Now here is the parent application:

```
package firstSteps;

import org.restlet.Application;
import org.restlet.Restlet;
import org.restlet.routing.Router;

public class FirstStepsApplication extends Application {

    /**
     * Creates a root Restlet that will receive all incoming calls.
     */
    @Override
    public Restlet createInboundRoot() {
        // Create a router Restlet that routes each call to a
        // new instance of HelloWorldResource.
        Router router = new Router(getContext());

        // Defines only one route
        router.attachDefault(HelloWorldResource.class);

        return router;
    }
}
```

Finally, here is the Servlet configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <display-name>first steps servlet</display-name>


    <servlet>
        <servlet-name>RestletServlet</servlet-name>

<servlet-class>org.restlet.ext.servlet.ServerServlet</servlet-class>
        <init-param>
            <param-name>org.restlet.application</param-name>
            <param-value>firstSteps.FirstStepsApplication
</param-value>
        </init-param>
    </servlet>


  <!-- Catch all requests -->
  <servlet-mapping>
    <servlet-name>RestletServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

For more information on Restlet, please check our documentation pages.

Javadocs

The Javadocs of the Restlet edition for GAE are available online as well:

- Restlet API
- Restlet Extensions
- Restlet Engine

Restlet edition for OSGi Environments

Introduction

While OSGi has been for a long time an important deployment environment for Restlet developers, including an innovative project lead by NASA JPL, the rise of RESTful web APIs, the addition of an Eclipse Public License option to Restlet Framework and an increased effort from Restlet community led to the addition of this special edition.

This edition dedicated to OSGi environments such as Eclipse Equinox and Apache Felix was necessary to make space for more OSGi specific features in the framework. Using our automated Restlet Forge, we are able to automatically deliver this edition, including dedicated Javadocs and distributions as illustrated above.

Eclipse Update Site

In addition, we provide a new distribution channel specific to this OSGi edition: an Eclipse Update Site supporting easy installation and automated update right from the Eclipse IDE.

Note that this is only recommended if you intended to develop OSGi applications leveraging Restlet Framework, not applications for other target environments such as Java EE or GAE. In the later cases, you should either manually copy the JAR files in your Eclipse project or rely on our Maven repository using a tool such as m2eclipse.

In order to use the Eclipse update site, please refer to instructions for the "Eclipse" distribution on the download page.

Restlet integration with ECF

In addition, thanks to the work of Scott Lewis, the Eclipse Communications Framework lead, an integration of Restlet with ECF is available.

This integration supports the OSGi Remote Services specification by adding REST/HTTP bindings based on Restlet and leveraging the Restlet extension for OSGi introduced below.

For additional details, you can read his blog posts #1 and #2.

Restlet extension

Finally, based on work initiated via a Google Summer of Code project in 2010, a Restlet extension for OSGi is also available.

The extension facilitates the development of very dynamic Restlet applications, supporting for example the live addition of resources and virtual hosts.

This org.restlet.ext.osgi extension is lead by Bryan Hunt and has received significant contributions from Wolfgang Werner.

Conclusion

Step after step, Restlet support for OSGi and the Eclipse ecosystem is growing and we will continue to work in this direction, with plans to add visual tooling for Restlet in Eclipse and to provide more integration with OSGi standard services such as the log service. As always, stay tuned!

References

Blog post - Restlet improves OSGi support: new edition and update site
Invoking server side code ==========

Description

The Restlet API for GWT is a subset of the Restlet API. It is a subset because, as mentioned above, parts of the Restlet API have been deleted, where they depend on Java platform features not available in the Javascript environment.

It is also a subset because it is fully (and only) asynchronous. This is necessary to conform to the behavior of the XmlHttpRequest-based Client, as well as the structures of the unthreaded Javascript environment.

Here is a snippet of code (from the downloadable example below) showing briefly how to instantiate a ClientResource and make an asynchronous GET call:

```
// Add behaviour on the close button.
closeButton.addClickHandler(new ClickHandler() {
    public void onClick(ClickEvent event) {
        // Add an AJAX call to the server
        ClientResource r = new ClientResource("/ping");


        // Set the callback object invoked when the response is received.
        r.setOnResponse(new Uniform() {
            public void handle(Request request, Response response) {
                try {
                    button.setText(response.getEntity().getText());
                } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    });
    r.get();
    dialogBox.hide();
  }
});
```

Working with JSON

Description

The GWTedition contains a JSON extension that provides a org.restlet.client.ext.json.JsonRepresentation class that you can leverage to either parse a JSON representation received or to serialize a JSON value.

Here is a sample code taken from the example application. The JsonRepresentation gives access to the underlying JSONValue after the representation has been parsed.

```
ClientResource r = new ClientResource("/test");


// Set the callback object invoked when the response is received.
r.setOnResponse(new Uniform() {
   public void handle(Request request, Response response) {
      // Get the representation as an JsonRepresentation
      JsonRepresentation rep = new JsonRepresentation(response.getEntity());


      // Displays the properties and values.
      try {
         JSONObject object = rep.getValue().isObject();
         if (object != null) {
            for (String key : object.keySet()) {
```

```
                jsonRoot.addItem(key + ":" + object.get(key));
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
});


// Indicates the client preferences and let the server handle
// the best representation with content negotiation.
r.get(MediaType.APPLICATION_JSON);
```

Examples

Introduction

An example project download is provided that works both under Eclipse and via ant build. This sets up the basic framework for GWT compilation and debugging in Hosted Mode, provides a basic Restlet-powered server, and demonstrates how the compiled GWT application can be bundled into an executable server JAR.

Download Restlet GWT -- Simple Example (application/force-download, 3.0 MB)

This is a simple example demonstrating some basic patterns for using Restlet and GWT. It produces an executable JAR file which depends only on core Restlet libraries (included in "lib") to start a small Java Web server on port 8888, which you can visit to access a compiled GWT application that, in turn, talks to the Web server.

You can also run the application in GWT Hosted Mode under Eclipse by using the included SimpleExample.launch configuration; right click this and choose Run As ... SimpleExample.

It is structured as an Eclipse project; you should be able to import it into your Eclipse 3.3 or better workspace. You can also run the ant build script directly to produce the executable.

You must supply your own GWT binaries (it should work for 1.7 and upper releases of GWT) ; update the Eclipse build path and/or the

"gwt.sdk" property in the ant build script to point to the GWT binaries for your platform.

Zip content

Name

Description

src

Source files of both:

- client page developped with GWT and Restlet, in package "org.restlet.example.gwt.client".
- Web server application developped with Restlet that serves the compiled web page and hosts a resource requested from the Web page.

war

Directory where are located:

- the web.xml configuration file (in "WEB-INF" subdirectory)
- the client page and javascript files generated by the GWT compilation phase
- the librairies the server side of this project depends on (in "WEB-INF/lib" subdirectory) - they are taken from the "gae" edition, release 2.0rc3

lib

Directory that contains the libraries of the client side of the project - they are taken from the "gwt" edition, release 2.0rc3

Description

GWT page

You can find the source code of this page in directory "src/org/restlet/example/gwt/client".

Once the server is running, this page can be accessed at the following URL: "http://localhost:8080/TestGwtRestlet_2_0.html" .

This page is in charge to display several sample item such image, button, etc organized in panels. All of these objects are instances of GWT classes:

```
// Define an image
```

```
Image img = new
Image("http://code.google.com/webtoolkit/logo-185x175.png");

// Define a button

final Button button = new Button("Click me");

[...]

// Define a panel

VerticalPanel vPanel = new VerticalPanel();

// We can add style names.

vPanel.addStyleName("widePanel");

vPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);

// Add image, button, tree

vPanel.add(img);

vPanel.add(button);
```

These class illustrates also how to add an asynchronous call with AJAX inside the final Web page. It is as simple as to use a simple Restlet client in order to request the "ping" resource located at URL '"http://localhost:8080/ping" :

```
    // Add an AJAX call to the server

    ClientResource r = new ClientResource("/ping");


    // Set the callback object invoked when the response is received.

    r.setOnResponse(new Uniform() {

        public void handle(Request request, Response response) {

            try {

                button.setText(response.getEntity().getText());

            } catch (IOException e) {

                 e.printStackTrace();

            }

        }

    });
```

```
    r.get();
```

Server side

Basically, the server s responsible to serve the generated page, and to respond to asynchronous call described just above.

The generated page is served by a simple Directory Restlet from the "bin" directory when the server is run under Eclipse.

The asynchronous call is delegated to the PingResource class which inherits from the ServerRestlet Resource class. It simply answers to requests with a line of text that contains the HTTP method of the request and, if available, its challengeScheme if the user has provided credentials.

```
public class PingResource extends ServerResource {


    @Get("txt")
    public String toText() {
        StringBuilder sb = new StringBuilder("Restlet server alive. Method: ");
        sb.append(getRequest().getMethod());


        ChallengeResponse challengeResponse = getRequest()
                .getChallengeResponse();
        if (challengeResponse != null) {
            sb.append("/ Auth. scheme: ");
            sb.append(challengeResponse.getScheme());
        }


        return sb.toString();
    }
}
```

Architecture flexibility

Introduction

GWT's idiomatic RPC approach
Figure 1. (For comparison) GWT's idiomatic RPC approach requires a Servlet container and communicates opaquely via JSON-encoded data atoms. Alternatives involve writing low-level HTTP code.

The client-side Restlet/GWT module
Figure 2. The client-side Restlet/GWT module (org.restlet.gwt) plugs into your Google Web Toolkit code as a module, and allows you to use the high level Restlet API to talk to any server platform. XML and JSON representations are supported by default, and of course you can develop your own representational abstractions as well. Code written for Restlet should be simple to port to Restlet-GWT.

Restlet/GWT can work alongside GWT-RPC
Figure 3. Restlet/GWT can work alongside GWT-RPC and other mechanisms. REST APIs exposed by your custom Servlets can work alongside GWT-RPC in a Servlet container environment.

Restlet, an ideal server side companion for Restlet-GWT
Figure 4. . Not only does Restlet provide a simple means of defining and mapping REST resources, it also provides useful facilities like the Redirector, which can be used to gateway requests to other servers, easily working around the single-source limitations of the AJAX programming model. The Restlet server-side library even works in GWT Hosted Mode using the Restlet Framework GWT Extension, so your entire application can be readily debugged.

Integration with standalone Restlet connectors
Figure 5. If you don't need to use the Servlet-dependent GWT-RPC API, your compiled GWT application can be deployed very efficiently in a standalone Restlet environment using one of the embedded server connectors. Restlet 1.1's Net server connector, for example, has no dependencies other than Java; with this, it is possible to create and release an extremely small, but very full featured, standalone application.

Setting up a project

Client-side configuration
To use Restlet on the client side of your GWT application:

1) Create an application normally with the applicationCreator and/or projectCreator scripts supplied with GWT, or using your favorite GWT design or IDE plugins.

2) Add the Restlet JAR (org.restlet.jar) from the Restlet edition for GWT to the project classpath *[explain]*

3) Add the following to your application's module definition file (*yourapp*.gwt.xml):

```
<inherits name='org.restlet.Restlet'/>
```

This will make the Restlet API available to your GWT compiled code. The Restlet module in turn inherits the GWT standard HTTP. Two Restlet extensions are also provided based on the JSON, and XML modules. You can also check the full Javadocs of the API online.

Server-side configuration

If you would like to use Restlet on the server-side as well, you must also modify GWT's web.xml file in the war/WEB-INF directory.

1) Add the org.restlet.jar, org.restlet.ext.servlet.jar and org.restlet.ext.gwt.jar files from the Restlet edition for Java EE to the project classpath. **The last file is necessary for the automatic bean serialization to work**. Also, be sure to add any other Restlet extension JARs necessary for extensions you plan to use on the server side.

2) Modify the web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>

   <servlet>

      <servlet-name>restlet</servlet-name>

      <servlet-class>org.restlet.ext.servlet.ServerServlet</servlet-class>

      <init-param>

        <param-name>org.restlet.application</param-name>

        <param-value>application</param-value>

      </init-param>

   </servlet>


   <servlet-mapping>

      <servlet-name>restlet</servlet-name>

      <url-pattern>/rest/*</url-pattern>
```

```
    </servlet-mapping>

        ...
</web-app>
```

For *application*, supply the name of your Restlet Application, e.g. com.mycompany.server.TestApplication. You can also supply a *component* via an org.restlet.component parameter, or any other permitted ServerServlet configuration parameter.

Restlet edition for Google Web Toolkit

Introduction

This chapter presents the Restlet edition for GWT, which is a client-side port of the Restlet Framework to GWT 2.2 and above releases. See this blog post for the official announce.

Description

Google Web Toolkit is a powerful and widely used platform for rich internet application. It is based on a smart compiler taking Java source and producing optimized JavaScript (byte)code.

By default, GWT recommends using a custom GWT-RPC mechanism to communicate between the GWT front-end (Web browser) and the back-end (Web server). In addition, the back-end has to be built using a Servlet container in order to work properly and to invoke your custom classes and methods. This might remind you of the RMI (Remote Method Invocation) or CORBA mechanisms and it comes with the same issues: the tight coupling between client and server code. Also, this reduces the opportunities of integration with other back-end technologies, outside Java/Servlet.

As you know, REST is a much more flexible and interoperable way to communicate between a client and a server. On the server-side, you can use Restlet or alternative technologies. But on the GWT front-end, the support has been limited so far. The GWT API does contain classes to send HTTP requests but they are quite low-level (you need to understand the HTTP headers syntax for example) and slow down productivity.

With the GWT 1.5 release and its support for the Java 5 syntax, it became possible to automate the port of the Restlet Framework to GWT. Of course, we kept only the classes required for the client-side obviously and had to remove classes based on Java APIs not available in GWT (such as NIO channels or BIO streams).

Automatic bean serialization

Finally, Restlet 2.0 added support for annotated Restlet interfaces and automatic bean serialization into its GWT edition, in a way that is consistent with other editions. See this first application example. With this feature, you don't need to care about formatting or parsing your beans in JSON or XML as the Restlet edition for GWT with automatically generate serializer classes for you using GWT Deferred Binding mechanism and reusing the bean serialization format of GWT-RPC (not the protocol, just the format). Combined with easy content negotiation on the Restlet server-side, your web API stays unchanged, able to support JSON, XML and other formats in parallel with no code change!

Deploy GWT sample application in Servlet container

Introduction

Since the example project provides a basic Restlet-powered server, it should be easy to deploy it inside a servlet container. GWT does not provide standard way to achieve this task, however, you can still get more information at this URL : http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=FAQ_PackageAppInWARFile.

Our sample application is composed of:

- a client part: static page and files generated by the compilation of the org.rest.example.gwt.SimpleExample class,
- and a server part: that serves resources called from the static page and serves the static files (via a Directory Restlet).

The main task is to generate a correct WAR file. Hopefully this is not very difficult.

Set up the WAR file

Here is the content of the target WAR file:

+ META-INF

  - MANIFEST.MF

+ WEB-INF

  - web.xml

  + lib

    - org.restlet-2.0rc3.jar: Restlet core archives

    - org.restlet.ext.servlet-2.0rc3.jar: Servlet adapter for Restlet

    - org.restlet.ext.gwt-2.0rc3.jar: Server-side integration with GWT 1.7.

- org.restlet.ext.crypto-2.0rc3.jar: Extension that provide support of the HTTP_DIGEST challenge scheme

- org.restlet.ext.xml-2.0rc3.jar: XML extension

- org.restlet.ext.json-2.0rc3.jar, org.json.jar: JSON extension

- RestletGWTSimpleExample.jar: generated by the ant script, contains the static pages and the code of the server application.

Here the content of the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app

   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

   "http://java.sun.com/dtd/web-app_2_3.dtd">


<web-app>

   <context-param>

      <param-name>org.restlet.clients</param-name>

      <param-value>CLAP FILE WAR</param-value>

   </context-param>


   <servlet>

      <servlet-name>adapter</servlet-name>


<servlet-class>org.restlet.ext.gwt.GwtShellServletWrapper</servlet-class>

      <init-param>

         <param-name>org.restlet.application</param-name>


<param-value>org.restlet.example.gwt.server.TestServerApplication</param-value>

      </init-param>

   </servlet>
```

```
    <servlet-mapping>
        <servlet-name>adapter</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Tips and tricks

Responses with no entity

This tip applies in the case of applications that run in GWT hosted mode.

Imagine that a resource handles a POST request and tells a user to redirect to another URI. In this case, it seems useless to add en entity body since its content is generally discarded by the client. However, you will notice that your Internet browser displays a white page even if the POST request has been correctly issued by the server.

This is due to a limitation not fixed yet where very small entities are swallowed by the servlet container of the GWT hosted mode.

If you face this problem, a simple workaround is just to add a reasonably sized entity to the response. For example, you can add a simple filter at the top of your hierarchy of nodes which ensure that an entity is sent back to the client.

Working with XML

Description

The GWT edition contains a XML extension that provides a org.restlet.client.ext.xml.DomRepresentation class that you can leverage to either parse a XML representation received or to serialize a XML DOM.

Here is a sample code taken from the example application. The DomRepresentation gives access to the underlying DOM document via the "getDocument()" method.

```
ClientResource r = new ClientResource("/test");


// Set the callback object invoked when the response is received.
r.setOnResponse(new Uniform() {
    public void handle(Request request, Response response) {
```

```
        // Get the representation as an XmlRepresentation

        DomRepresentation rep = new
DomRepresentation(response.getEntity());



        // Loop on the nodes to retrieve the node names and text content.

        NodeList nodes;

        try {

            nodes = rep.getDocument().getDocumentElement().getChildNodes();

            for (int i = 0; i < nodes.getLength(); i++) {

                Node node = nodes.item(i);

                xmlRoot.addItem(node.getNodeName() + ":" +
node.getFirstChild().getNodeValue());

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

});



// Indicates the client preferences and let the server handle

// the best representation with content negotiation.

r.get(MediaType.TEXT_XML);
```

Restlet edition for Java EE

Introduction

This chapter presents the Restlet Framework edition for Java EE (Java Enterprise Edition).

This edition is aimed for development and deployment of Restlet applications inside Java EE application server, or more precisely inside Servlet containers such as Apache Tomcat.

Getting started

The rest of this page should get you started with the Restlet Framework, Java EE edition, in less than 10 minutes. It explains how to create a resource that says "hello, world" and run it.

What do I need?

We assume that you have a development environment set up and operational, and that you already have installed the Java 1.5 (or higher). In case you haven't downloaded the Restlet Framework yet, select one of the available distributions of the Restlet Framework 2.3.

The "hello, world" application

Let's start with the core of a REST application: the Resource. Here is the code of the single resource defined by the sample application. Copy/paste the code in your "HelloWorldResource" class.

```
package firstSteps;


import org.restlet.resource.Get;

import org.restlet.resource.ServerResource;


/**
 * Resource which has only one representation.
 */
public class HelloWorldResource extends ServerResource {


   @Get
   public String represent() {
      return "hello, world";
   }
```

```
}
```

Then, create the sample application. Let's call it "FirstStepsApplication" and copy/paste the following code:

```
package firstSteps;


import org.restlet.Application;

import org.restlet.Restlet;

import org.restlet.routing.Router;


public class FirstStepsApplication extends Application {


    /**

     * Creates a root Restlet that will receive all incoming calls.

     */

    @Override

    public synchronized Restlet createInboundRoot() {

        // Create a router Restlet that routes each call to a new instance of
HelloWorldResource.

        Router router = new Router(getContext());


        // Defines only one route

        router.attach("/hello", HelloWorldResource.class);


        return router;

    }


}
```

Run in a Servlet container

Let's now deploy this Restlet application inside your favorite Servlet container. Create a new Servlet Web application as usual, add a "firstStepsServlet" package and put the resource and application classes in. Add the archives listed below into the directory of librairies (/WEB-INF/lib):

- org.restlet.jar
- org.restlet.ext.servlet.jar

Then, update the "web.xml" configuration file as follow:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
        xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
            http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>first steps servlet</display-name>

  <!-- Restlet adapter -->
  <servlet>
    <servlet-name>RestletServlet</servlet-name>
    <servlet-class>org.restlet.ext.servlet.ServerServlet</servlet-class>
    <init-param>
        <!-- Application class name -->
        <param-name>org.restlet.application</param-name>
        <param-value>firstSteps.FirstStepsApplication</param-value>
    </init-param>
  </servlet>

  <!-- Catch all requests -->
  <servlet-mapping>
```

```
    <servlet-name>RestletServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Finally, package the whole as a WAR file called for example "firstStepsServlet.war" and deploy it inside your Servlet container. Once you have launched the Servlet container, open your favorite web browser, and enter the following URL:

http://<your server name>:<its port number>/firstStepsServlet/hello

The server will happily welcome you with the expected "hello, world" message. You can find the WAR file (packaged with archives taken from Restlet Framework 2.0 Milestone 5) in the "First steps application" files.

Run as a standalone Java application

A Restlet application cannot only run inside a Servlet container, but can also be run as a standalone Java application using a single "org.restlet.jar" JAR.

Create also a main class, copy/paste the following code wich aims at defining a new HTTP server listening on port 8182 and delegating all requests to the "FirstStepsApplication".

```
public static void main(String[] args) throws Exception {
    // Create a new Component.
    Component component = new Component();

    // Add a new HTTP server listening on port 8182.
    component.getServers().add(Protocol.HTTP, 8182);

    // Attach the sample application.
    component.getDefaultHost().attach("/firstSteps",
            new FirstStepsApplication());

    // Start the component.
    component.start();
```

```
}
```

Once you have launched the main class, if you can open your favorite web browser, and gently type the following URL: http://localhost:8182/firstSteps/hello, the server will happily welcome you with a nice "hello, world". Otherwise, make sure that the classpath is correct and that no other program is currently using the port 8182.

You can find the sources of this sample application in the "First steps application" files.

Conclusion

We hope you that enjoyed these first steps and encourage you to check the equivalent page in the Java SE edition for standalone deployments of the same application. This can also be a convenient way to develop and test your Restlet application before actually deploying it in a Java EE application server.

Notes

- Thanks to Didier Girard for suggesting this page.

Client connectors

Introduction

It is possible to declare client connectors when the application is hosted on a Servlet container. The Javadocs of the Servlet adapter (ServerServlet class) answers this question and others related to the configuration of Restlet based applications.

The web.xml file declares the client connectors in a dedicated "org.restlet.clients" parameter:

```
<servlet>
   <servlet-name>RestletAdapter</servlet-name>
   [...]
   <!-- List of supported client protocols (Optional - Only in mode 3) -->
   <init-param>
      <param-name>org.restlet.clients</param-name>
      <param-value>HTTP HTTPS FILE</param-value>
   </init-param>
   [...]
</servlet>
```

Restlet edition for Java SE

Introduction

This chapter presents the Restlet Framework edition for Java SE (Java Standard Edition).

This edition is aimed for development and deployment of Restlet applications inside a regular Java virtual machine using the internal HTTP server of the Restlet Engine, or a pluggable one such as Jetty. This page contains a detailed list of available HTTP server connectors.

Getting started

The rest of this page should get you started with the Restlet Framework, Java SE edition, in less than 10 minutes. It explains how to create a resource that says "hello, world" and run it.

What do I need?

We assume that you have a development environment set up and operational, and that you already have installed the Java 1.5 (or higher). In case you haven't downloaded the Restlet Framework yet, select one of the available distributions of the Restlet Framework 2.3. Make sure you add org.restlet.jar in your Build path.

The "hello, world" application

Let's start with the core of a REST application: the Resource. Here is the code of the single resource defined by the sample application. Copy/paste the code in your "HelloWorldResource" class.

```
package firstSteps;


import org.restlet.resource.Get;

import org.restlet.resource.ServerResource;


/**

 * Resource which has only one representation.
```

```
 */
public class HelloWorldResource extends ServerResource {


    @Get
    public String represent() {
        return "hello, world";
    }


}
```

Then, create the sample application. Let's call it "FirstStepsApplication" and copy/paste the following code:

```
package firstSteps;


import org.restlet.Application;
import org.restlet.Restlet;
import org.restlet.routing.Router;


public class FirstStepsApplication extends Application {


    /**
     * Creates a root Restlet that will receive all incoming calls.
     */
    @Override
    public synchronized Restlet createInboundRoot() {
        // Create a router Restlet that routes each call to a new instance of
HelloWorldResource.
        Router router = new Router(getContext());


        // Defines only one route
```

```
        router.attach("/hello", HelloWorldResource.class);


        return router;
    }


}
```

Run as a standalone Java application

A Restlet application can run inside a regular Java virtual machine or Java Runtime Environment (JRE), using a single "org.restlet.jar" JAR in the classpath. For this we only need to create a Restlet component and associate a HTTP server connector.

Create also a main class, copy/paste the following code which aims at defining a new HTTP server listening on port 8182 and delegating all requests to the "FirstStepsApplication".

```
public static void main(String[] args) throws Exception {
    // Create a new Component.
    Component component = new Component();

    // Add a new HTTP server listening on port 8182.
    component.getServers().add(Protocol.HTTP, 8182);

    // Attach the sample application.
    component.getDefaultHost().attach("/firstSteps",
            new FirstStepsApplication());

    // Start the component.
    component.start();
}
```

Once you have launched the main class, if you can open your favorite web browser, and gently type the following URL: http://localhost:8182/firstSteps/hello , the server will happily welcome

you with a nice "hello, world". Otherwise, make sure that the classpath is correct and that no other program is currently using the port 8182.

You can find the sources of this sample application in the "FirstStepsStandalone (application/zip, 1.4 kB)" file.

Conclusion

We hope you that enjoyed these first steps and encourage you to check the equivalent page in the Java EE edition for deployments of the same application in Servlet containers. This can also be a convenient way to deploy your Restlet application in an existing Java EE application server available in your organization.

Notes

- Thanks to Didier Girard for suggesting this page.

Logging

Introduction

By default, Restlet relies on JDK's logging API (JULI) to log records about its activity. For a complete documentation on this standard API, you can check the related JDK documentation. For additional configuration details, you should also read the Javadocs of the LogManager class. For example, to indicate the location of your logging properties file, you should add the following parameter to your JVM:

```
-Djava.util.logging.config.file="/home/myApp/config/myLogging.properties"
```

When developing your Restlet code, you should always use the current context to get a logger. This will ensure the proper naming and filtering of your log records. Most classes like Restlet subclasses or ServerResource subclasses offer a "getLogger()" method. Otherwise, you can rely on the getContext().getLogger() method or on the static "Context.getCurrentLogger()" method.

Note also that there are two main types of log record in Restlet:

- The code related log records, for example used for tracing, debugging or error logging
- The access related log records, used to log the requests received by a component (identical to IIS or Apache logs)

Programmatic configuration

Since version 2.1, it is now possible to programmatically change the log level using the Engine#setLogLevel(...) and setRestletLogLevel(...) static methods. It is also possible to enable selective call logging by

setting the Request#loggable property or by overriding the LogService#isLoggable(Request) method.

If you do provide a logging configuration file via the system properties, it will take over the programmatic configuration. Also, the new default log formatter will write each log entry in the console on a single compact line, reducing confusion while debugging.

It is also possible to define the calls to log based on a URI template by calling the LogService#setLoggableTemplate(...) method.

Logger names

As a single JVM can contain several Restlet components at the same time and each component can also contain several applications, we need a way to filter the log records for each application if needed. For this purpose, we leverage the hierarchical nature of JDK loggers:

- At the root of all Restlet loggers is the logger named "org.restlet"
- Below you have a logger per component, based on the simple class name. For example "org.restlet.MyComponent".
- Below you have a logger per application, based on the simple class name. For example "org.restlet.MyComponent.MyApplication".
- Each component also logs access using a name such as "org.restlet.MyComponent.LogService". This name can be customized using the "LogService.loggerName" property.

It can often be difficult to configure your logging properties file because you don't always know precisely the logger names. For Restlet code, everything is under the root "org.restlet" logger so it is relatively easy.

However, as the Restlet extensions rely on many third-party library, you need to understand how each one handles logging in order to consistently configure you logging. Many of them rely on Apache Commons Logging API as a neutral API that can plug implementations such as Log4J or JDK Logging. Other use the neutral SLF4J, but in most of the cases, it is possible to redirect those alternative logging mechanisms to the JDK logging one or the other way around.

Here is an attempt to list the logger names used by those libraries. Please help us to complete this table.

Library

db4o

Grizzly

Bloat

Java Activation

Java Mail

Java Servlet

JAX-RS

JAXB

StAX

JLine

JXTA

OAuth

AntLR

Commons Codec

Commons Collections

Commons DBCP

Commons FileUpload

Commons HTTP Client

Commons IO

Commons Lang

Commons Logging

Commons Pool

MINA

Velocity

Bouncy Castle

FreeMarker

JiBX

JSON

JUnit

Jetty

Simple

Spring

Tanuki Wrapper

If the logger name you are looking for isn't listed, there is an easy way to detect it. You just have to call the static "org.restlet.engine.log.TraceHandler.register()" method at the beginning of your program. It will replace the default console handler by a more compact one that will display the logger name for each log record received by the root logger (i.e. all the log records).

Sample configuration

As a starting point for your own logging properties file, here is the one we use on our Restlet base Web server. Feel free to copy and paste as needed.

```
# ====================================

# ==                    ==

# ==   Web Logging Properties   ==

# ==                    ==

# ====================================




# ------------------

# General properties

# ------------------




# This defines a whitespace separated list of class names for handler classes to load and register as handlers on

# the root Logger (the Logger named ""). Each class name must be for a Handler class which has a default constructor.

# Note that these Handlers may be created lazily, when they are first used.
```

```
handlers=java.util.logging.FileHandler
```

In this first section, we declare one default handler that will receive the log records. It is a file handler that will be configured below.

```
# ------------------

# Loggers properties

# ------------------


.level=WARNING

org.mortbay.level=WARNING

org.restlet.level=WARNING

com.noelios.level=WARNING


com.noelios.web.WebComponent.www.level=INFO

com.noelios.web.WebComponent.www.handlers=com.noelios.restlet.util.AccessLogFileHandler

com.noelios.web.WebComponent.www.useParentHandlers=false
```

In this second section, we indicate that by default we are only interested in log records with a WARNING level. We also configure the Mortbay's Jetty, Restlet API and NRE log levels to WARNING. This isn't necessary here but it can be useful when you need to lower the level of only one of them at some point, for debugging purpose for example.

We also configured a logger for the WWW access log of our Restlet component. For information, our Component subclass has this code in its constructor:

```
getLogService().setLoggerName("com.noelios.web.WebComponent.www");
```

Also note that we use a specific handler for this logger, the AccessLogFileHandler which is provided in the NRE. It can be easily configurer to produce Apache-style HTTP log files.

```
# ------------------------

# ConsoleHandler properties

# ------------------------
```

Specifies the default level for the Handler (defaults to Level.INFO).

java.util.logging.ConsoleHandler.level=WARNING

Specifies the name of a Filter class to use (defaults to no Filter).

java.util.logging.ConsoleHandler.filter=

Specifies the name of a Formatter class to use (defaults to java.util.logging.SimpleFormatter).

java.util.logging.ConsoleHandler.formatter=

The name of the character set encoding to use (defaults to the default platform encoding).

java.util.logging.ConsoleHandler.encoding=

In the section above we have disabled the default ConsoleHandler configuration as we don't use it on our server-side application.

General FileHandler properties

Specifies the default level for the Handler (defaults to Level.ALL).

java.util.logging.FileHandler.level=ALL

Specifies the name of a Filter class to use (defaults to no Filter).

java.util.logging.FileHandler.filter=

Specifies the name of a Formatter class to use (defaults to java.util.logging.XMLFormatter)

java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter

The name of the character set encoding to use (defaults to the default platform encoding).

java.util.logging.FileHandler.encoding=

Specifies an approximate maximum amount to write (in bytes) to any one file.

If this is zero, then there is no limit. (Defaults to no limit).

java.util.logging.FileHandler.limit=10000000

Specifies how many output files to cycle through (defaults to 1).

java.util.logging.FileHandler.count=100

Specifies a pattern for generating the output file name. (Defaults to "%h/java%u.log").

A pattern consists of a string that includes the following special components that will be replaced at runtime:

"/" the local pathname separator

"%t" the system temporary directory

"%h" the value of the "user.home" system property

"%g" the generation number to distinguish rotated logs

"%u" a unique number to resolve conflicts

"%%" translates to a single percent sign "%"

java.util.logging.FileHandler.pattern=/home/prod/data/log/WebComponent-app-%u-%g.log

Specifies whether the FileHandler should append onto any existing files (defaults to false).

java.util.logging.FileHandler.append=

Here we specify the file size limit, the number of rotation files (100) and the file name template.

```
# LogFileHandler properties

# ------------------------


# Specifies the default level for the Handler  (defaults to Level.ALL).

# org.restlet.engine.log.AccessLogFileHandler.level=ALL


# Specifies the name of a Filter class to use (defaults to no Filter).

# org.restlet.engine.log.AccessLogFileHandler.filter=


# Specifies the name of a Formatter class to use (defaults to
java.util.logging.XMLFormatter)

org.restlet.engine.log.AccessLogFileHandler.formatter=com.noelios.restlet.util
.AccessLogFormatter


# The name of the character set encoding to use (defaults to the default
platform encoding).

# org.restlet.engine.log.AccessLogFileHandler.encoding=


# Specifies an approximate maximum amount to write (in bytes) to any one
file.

# If this is zero, then there is no limit. (Defaults to no limit).

org.restlet.engine.log.AccessLogFileHandler.limit=10000000


# Specifies how many output files to cycle through (defaults to 1).

org.restlet.engine.log.AccessLogFileHandler.count=100


# Specifies a pattern for generating the output file name. (Defaults to
"%h/java%u.log").

# A pattern consists of a string that includes the following special
components that will be replaced at runtime:

#    "/" the local pathname separator
```

```
#    "%t" the system temporary directory

#    "%h" the value of the "user.home" system property

#    "%g" the generation number to distinguish rotated logs

#    "%u" a unique number to resolve conflicts

#    "%%" translates to a single percent sign "%"
.pattern=/home/prod/data/log/WebComponent-www-%u-%g.log


# Specifies whether the FileHandler should append onto any existing files
(defaults to false).

# org.restlet.util.AccessLogFileHandler.append=
```

This is similar to the previous section, but specific to our AccessLogFileHandler log handler. This let's us use a specific log formatter called AccessLogFormatter, also provided by the Engine.

Bridges to alternative logging mechanisms

Some users that prefer to use Log4J or LogBack instead of JULI, especially because they have more features and seem more flexible. Sometimes, Restlet applications are not isolated and have to integrate with existing logging strategies.

In these cases, we we propose two options.

SLF4J bridge from JULI

This bridge is provided by the SLF4J project. Once it is installed, you will just need to add these lines of code:

```
 import org.slf4j.bridge.SLF4JBridgeHandler;

 SLF4JBridgeHandler.install();
```

Then, you can drop the JAR from SLF4J corresponding to the target logging mechanism. For details, you can check this page. This is an excellent solution, especially if you are using Restlet extensions that rely on their own logging system, such as Jetty using Apache Commons Logging. In those case, you can use the bridge from Commons Logging to SLF4J and keep all log consistent.

In addition, if you want to disable the usual console outputs, you could add those lines:

```
java.util.logging.Logger rootLogger = LogManager.getLogMan
ager().getLogger("");

Handler[] handlers = rootLogger.getHandlers();
```

```
rootLogger.removeHandler(handlers[0]);

SLF4JBridgeHandler.install();
```

Restlet LoggerFacade for SLF4J

The previous method works, but has some performance penalties due to the systematic creation of JULI LogRecord instances. In order to workaround such issues, Restlet 2.0 has added a LogFacade class which is in charge of providing JULI Loggers to the Restlet Engine. The default class fully relies on JULI, but it is possible to replace it by a subclass by setting a system property.

A special subclass for SLF4J is even provided in the "org.restlet.ext.slf4j" extension. In order to install it, you need to add this system property

```
org.restlet.engine.loggerFacadeClass=org.restlet.ext.slf4j.Slf4jLoggerFacade
```

You can set this property on the command line with the "-D" prefix, or programmatically with the System#setProperty() method.

Additional resources

- Java Logging API and How To Use It

Part III - Restlet Editions

Introduction

The Restlet Framework is written in the Java language but can target very different execution environments. This part of the user guide will present the various editions supported and cover the differences between them.

- Android
- Google App Engine
- Google Web Toolkit
- Java EE
- Java SE
- OSGi Sample application ==========

Introduction

This sample application aims at illustrating the port of the Restlet API on the Google Android platform.

Architecture

The developed application uses the software architecture illustrated below. On the upper left side, you have an enhanced Android contacts

application that can recognize the friends of your contacts. For this, you need to enter the URI of the FOAF profile of your contacts in the "Note" field. In the future, a specific "FOAF" field will be added and this URI could also be guessed or provided via alternative means.

This FOAF URI entered can be either local (our case for testing reasons and to illustrate server-side Restlet support) or remote to use your mobile Web access if available. When run, the enhanced contacts applications lists the current contacts registered in the address book. For those who provide a FOAF URI, a click on their entry in the list invokes the FOAF service illustrated below. This service is built on top of Restlet and its recently added RDF extension. It retrieves the FOAF representation, parses it and displays the friends of your contact in the GUI. The user can then add some of those friends as new local contacts.

For testing purpose, we run a local HTTP server with Restlet that can server FOAF profiles (in the RDF/N-Triples media type) at those URIs:

- http://localhost/users/1
- http://localhost/users/2

Screenshots

List of currently registered contacts in the Android's address book

List of friends of the selected contact

A new contact has been added

Implementation

"FOAF" Service

Declaration of the service (AndroidManifest.xml):

```
<service android:name=".service.ContactService" android:exported="true"
android:enabled="true">

   <intent-filter>

      <action
android:name="org.restlet.example.android.service.IContactService" />

   </intent-filter>

</service>
```

Declaration of the IPC (inter process communication) interface (org/restlet/example/android/service/IContactService.aidl)

```
package org.restlet.example.android.service;


interface IContactService {


    List<FoafContact> getFriends(String foafUri);


}
```

Implementation of the FoafContact class

This class, referenced in the IPC interface description file, must implement the Parcelable java interface.

```
public class FoafContact implements Parcelable {

    /**
     * Used to de-serialize a stream into a FoafContact.
     */
    public static final Parcelable.Creator<FoafContact> CREATOR = new
Parcelable.Creator<FoafContact>() {
        public FoafContact createFromParcel(Parcel in) {
            return new FoafContact(in);
        }


        public FoafContact[] newArray(int size) {
            return new FoafContact[size];
        }
    };


    /** First name of the contact. */
    private String firstName;
```

```
/** FOAF URI of the contact. */
private String foafUri;

/** Image representation of the contact. */
private String image;

/** Last name of the user. */
private String lastName;

/** Phone number of the user. */
private String phone;

public FoafContact() {
    super();
}

private FoafContact(Parcel in) {
    super();
    firstName = in.readString();
    lastName = in.readString();
    phone = in.readString();
}

public int describeContents() {
    return 0;
}
```

```
[  getters/setters ]


    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(firstName);
        dest.writeString(lastName);
        dest.writeString(phone);
    }


}
```

Implementation of the service (org.restlet.example.android.ContactService)

Basically, the service consists in getting the FOAF profile according to its URI, then parsing this RDF document (in this case, it is generated in N-Triples format), and retrieving the list of friends declared in this FOAF profile.

```
ClientResource foafProfile = new ClientResource(foafUri);

Representation rep = foafProfile.get();

if (foafProfile.getStatus().isSuccess()) {
    FoafRepresentation foafRep = new FoafRepresentation(rep);
    return foafRep.getFriends();
}
```

Contact activity

This activity is in charge to display the list of friends of a contact (assuming it has a correct foaf URI, in the "Note" field of the address book). It calls the "FOAF" service.

Declaration of the activity (AndroidManifest.xml)

```
<activity android:name=".ContactActivity" android:label="@string/contact">
    <intent-filter>
    <category android:name="android.intent.category.DEFAULT" />
    <action android:name="org.restlet.android.example.CONTACT_DETAIL" />
    </intent-filter>
```

```
</activity>
```

Declaration of its layout (res/layout/contact.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <ListView android:id="@android:id/list" android:layout_width="fill_parent"
      android:layout_height="fill_parent" android:layout_weight="1"
      android:drawSelectorOnTop="false" style="@style/contacts_list" />

   <TextView android:id="@+id/empty" android:layout_width="fill_parent"
      android:layout_height="fill_parent" style="@style/empty"
android:text="" />

   <ImageView android:id="@+id/imagefoaf"
android:src="@drawable/restletandroid"
      android:layout_width="fill_parent"
android:layout_height="wrap_content"
      android:adjustViewBounds="true" />

</LinearLayout>
```

Implementation (org.restlet.example.android.ContactActivity)

Connection to the remote service. Refreshes the list of friends of the current contact.

```
   private ServiceConnection connection = new ServiceConnection() {
      public void onServiceConnected(ComponentName name, IBinder
service) {
         contactService = IContactService.Stub.asInterface(service);
         // Once connected, then update the interface
```

```
            loadFriends();
        }


        public void onServiceDisconnected(ComponentName name) {
            contactService = null;
        }
    };
```

Load the list of friends of the current contact, from its foaf profile. It simply calls the contactService method "getFriends(String)".

```
    private void loadFriends() {
        if (contactService != null) {
            try {
                List<FoafContact> list =
contactService.getFriends(this.contact.getFoafUri());
                this.friends = new ArrayList<Contact>();
                for (int i = 0; i < list.size(); i++) {
                    Contact contact = new Contact();
                    contact.setFirstName(list.get(i).getFirstName());
                    contact.setLastName(list.get(i).getLastName());
                    contact.setPhone(list.get(i).getPhone());


                    friends.add(contact);
                }
            } catch (RemoteException e) {
                Log.e("contactFoaf", "error", e);
            }
        }
        handler.sendEmptyMessage(0);
    }
```

Used to unbind the service.

```
@Override
protected void onPause() {
    super.onPause();
    if (contactService != null) {
        this.unbindService(connection);
    }
}
```

Used to bind the service, when the activity starts.

```
@Override
protected void onStart() {
    super.onStart();
    if (contactService == null) {
        bindService(new Intent(
                "org.restlet.example.android.service.IContactService"),
                connection, Context.BIND_AUTO_CREATE);
    }
}
```

Here is the Eclipse project of the sample application (including dependencies jars): androidRestlet (application/zip, 623.6 kB)

Restlet edition for Android

Introduction

This document will cover the Restlet edition for Android, which is a port of the Restlet Framework to the Android mobile OS.

A Web server on a mobile phone

There are several reasons for having a Web server on a mobile phone:

- Test locally client Web applications without having to consume the network access which might be limited by cost or availability in some areas
- Allow third-party applications, on other phones or other platforms to access to your phone remotely. This requires strong security

mechanisms that are provided in part by the Restlet Framework as well as network level authorizations by the carrier.

- Run local Android applications that are leveraging the internal Web browser and behaving like regular hypermedia applications instead of using GUIs specific to Android.

A Web client on a mobile phone

The port of Restlet on Android also includes a full Web client to access to either local or remote Web servers. Android already has an HTTP client library, but with a much lower-level API while Restlet let you leverage higher-level features naturally (such as conditional methods, content negotiation, etc.). The support of other protocols than HTTP (like file system access via file:/// URIs) is also useful.

Specificities

Contrary to other editions, the Android edition can't leverage Restlet's autodiscovery mechanism for connectors and converters provided as Restlet extensions. This is due to a limitation in the way Android repackages JAR files, leaving out the descriptor files in the "META-INF/services/" packages used by the Restlet Framework for autodiscovery.

The workaround consist of manually registering those additional connectors and converter in the Restlet engine. Here is an example for the Jackson converter:

```
import org.restlet.engine.Engine;

import org.restlet.ext.jackson.JacksonConverter;


// ...


Engine.getInstance().getRegisteredConverters().add(new
JacksonConverter());
```

Here is another example for the Apache HTTP Client:

```
import org.restlet.engine.Engine;

import org.restlet.ext.httpclient.HttpClientHelper;


// ...
```

```
Engine.getInstance().getRegisteredClients().clear();

Engine.getInstance().getRegisteredClients().add(new HttpClientHelper(null));
```

Velocity extension

Introduction
Velocity is an embeddable tool that provides a template language used to reference java objects passed to the engine. Any kind of text output (from HTML to autogenerated source code) can be generated.

Description
This extension lets you generate Representations based on the Velocity template engine.

For additional details, please consult the Javadocs.

E4 extension

Introduction
To be done.

Javadocs.

GSON extension

Introduction
To be done.

Javadocs.

ROME extension

Introduction
Integration with ROME, supporting the formatting of syndicated feeds in various versions of the RSS and Atom languages.

For additional details, please consult the Javadocs.

GAE extension

Introduction
This extension provides an integration with the Google App Engine API. For details on Google App Engine (aka GAE), see the home page.

Description
At the time of writing, this extension provides an Authenticator filter that integrates the GAE UserService and detect if the current request is auithenticated using a Google account. For additional details, please consult the Javadocs.

User example

This sample code illustrates how to use the Authenticator and the Enroler provided by the GAE extension. These features are used by the Application deployed to the Google App Engine platform.

Usage of the GaeAuthenticator filter

The GaeAuthenticator is able to check wether or not the current request is sent by a logged user. In case the authenticator is not optional, the user is automatically redirected to the standard login page.

```
GaeAuthenticator guard = new GaeAuthenticator(getContext());

// Attach the guarded hierarchy of URIs to the Authenticator filter

guard.setNext(adminRouter);



// Attach this guarded set of URIs

router.attach("/admin", guard);
```

Usage of the GaeEnroler

The GaeEnroler checks wether the current logged user can adminstrate the deployed application and completes the list of user's roles with a defined aministrator role. It is used in conjunction with the GaeAuthenticator filter described above.

```
 /** Administrative role. */

 private Role adminRole = new Role("admin", "Administractive role");



 /**

  * Constructor.

  */

 public TestServerApplication() {

   // Add The administration role to the application.

   getRoles().add(adminRole);

 }



 @Override
```

```
  public Restlet createInboundRoot() {
    [...]
    // Guard this hierarchy of URIs
    GaeAuthenticator guard = new GaeAuthenticator(getContext());
    // Set the Enroler that will set the Admin Role according to the right of the
logged user's.
    guard.setEnroler(new GaeEnroler(adminRole));
    guard.setNext(adminRouter);
    [...]
  }
```

Thanks to the Enroler, a resource can check if the current request is sent by an administrator:

```
  @Override
  protected void doInit() throws ResourceException {
    if (isInRole("admin")) {
      // logged as admin
    [...]
    }
  }
```

Simple Framework extension

Introduction

This connector is based on Simple framework which is an open source embeddable Java based HTTP engine capable of handling high loads.

Description

This connector supports the following protocols: HTTP, HTTPS.

The list of supported specific parameters is available in the javadocs:

- Simple common parameters
- HTTP specific parameters
- HTTPS specific parameters

For additional details, please consult the Javadocs.

OSGi extension

Support for highly dynamic web APIs.

This extension contains support for the OBAP (OSGi Bundle Access Protocol) pseudo-protocol. It includes a client connector that lets you access to resources from other OSGi bundles in a RESTful way. Just use this syntax:

obap://{bundleSymbolicName}/{pathToResource}

For additional details, please consult the Javadocs.

Supporting AJP with Jetty

Introduction

Here are a few links that will get you started:

- What's AJP(Apache JServ Protocol)?
- What's Apache module mod_jk?
- What's Jetty,and how to get it?!
- How to embedding Jetty in program way?

This page needs to be updated for Restlet 2.x

Description

Embedding Jetty

JettyAJPApplication.Class

```
package com.bjinfotech.restlet.practice.demo.microblog;


import org.restlet.Application;

import org.restlet.Component;

import org.restlet.Directory;

import org.restlet.Restlet;

import org.restlet.Router;

import org.restlet.Server;

import org.restlet.data.Protocol;

import com.noelios.restlet.ext.jetty.AjpServerHelper;

import com.noelios.restlet.ext.jetty.HttpServerHelper;
```

```
import com.noelios.restlet.ext.jetty.JettyServerHelper;


public class JettyAJPApplication extends Application {
    public static void main(String[] argv) throws Exception{
        Component component=new Component();


        Application application=new Application(component.getContext()){
            @Override
            public Restlet createRoot(){
                final String
DIR_ROOT_URI="file:///E:/eclipse3.1RC3/workspace/RestletPractice/static_files
/";


                Router router=new Router(getContext());
                Directory dir=new Directory(getContext(),DIR_ROOT_URI);
                dir.setListingAllowed(true);
                dir.setDeeplyAccessible(true);
                dir.setNegotiateContent(true);
                router.attach("/www/",dir);
                return router;
            }
        };


        ...
        //create embedding jetty server
        Server embedingJettyServer=new Server(
                component.getContext(),
                Protocol.HTTP,
                8182,
```

```
            component
        );
    //construct and start JettyServerHelper
    JettyServerHelper jettyServerHelper=new
HttpServerHelper(embedingJettyServer);
    jettyServerHelper.start();

    //create embedding AJP Server
    Server embedingJettyAJPServer=new Server(
            component.getContext(),
            Protocol.HTTP,
            8183,
            component
        );

    //construct and start AjpServerHelper
    AjpServerHelper ajpServerHelper=new
AjpServerHelper(embedingJettyAJPServer);
    ajpServerHelper.start();

  }
}
```

Running JettyAJPApplication

```
2008-02-13 15:33:54.890::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
2008-02-13 15:33:54.953::INFO:  jetty-6.1.x
2008-02-13 15:33:55.140::INFO:  Started SelectChannelConnector @ 0.0.0.0:8182
2008-02-13 15:33:55.140::INFO:  jetty-6.1.x
2008-02-13 15:33:55.140::INFO:  AJP13 is not a secure protocol. Please protect port
8183
2008-02-13 15:33:55.140::INFO:  Started Ajp13SocketConnector @ 0.0.0.0:8183
```

Configuring Apache HTTPd server with mod_jk

- put mod_jk.so into your <apache-root>/modules/ directory
- you can download mod_jk.so here http://archive.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/
- add the entry below in your httpd.conf apache configuration file located in <apache-root>/conf/ directory:

```
<IfModule !mod_jk.c>

    LoadModule jk_module  modules/mod_jk.so

</IfModule>


<IfModule mod_jk.c>

    JkWorkersFile "conf/worker.properties"

    JkLogFile "logs/mod_jk.log"

    JkLogLevel info

    JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

    JkOptions +ForwardKeySize +ForwardURICompat

</IfModule>
```

- **LoadModule jk_module modules/mod_jk.so** tells your apache server to load the mod_jk libray and where it is located.
- **JkWorkersFile conf/worker.properties** tells mod_jk where your worker.properties is located.
- **JkLogFile logs/mod_jk.log** tells mod_jk where to write mod_jk related Logs.

After adding the mod_jk configuration you may add a **VirtualHost** Entry in the same file (httpd.conf) as long as its located below your mod_jk configuration entry:

```
<VirtualHost host:*>

  ServerName yourserver

   ServerAdmin user@yourserver
```

you may add further entries concerning log-files, log-level, URL-rewriting, ...

pass requests through to jetty worker

JkMount /* jetty

</VirtualHost>

- Add a worker file **worker.properties** in your <apache-root>/conf/
- add the entries below, and make sure to specify your ip-address or hostname in **worker.jetty.host** property entry to where your jetty application is runnning

worker.list=jetty worker.jetty.port=8009 worker.jetty.host=<server name or ip where your jetty will be running> worker.jetty.type=ajp13

mod_jk Compatibilities

Apache

Apache 1.3

Apache 2.0 (2.0.59)

Apache 2.2

Running Apache Httpd and test your application

[To be detailled]

More Resource

- Jetty doc:Configuring+AJP13+Using+mod_jk moved to Jetty/Tutorial/Apache
- Apache Httpd Document
- HttpServerHelper Class API
- Server Class API
- AjpServerHelper Class API

Eclipse Jetty extension

Introduction

This connector is based on the Eclipse Jetty open-source web server. Jetty is popular alternative to Tomcat developed by Mortbay Consulting and has a nice separation between its HTTP protocol implementation and its support for the Servlet API which led to the first HTTP server connector developed for the Restlet Framework.

Description

This connector supports the following protocols: HTTP, HTTPS, AJP.

The list of supported specific parameters is available in the javadocs:

- Jetty common parameters
- HTTP specific parameters
- HTTPS specific parameters

Here is the list of dependencies of this connector:

- Jetty
- Java Servlet

For additional details, please consult the Javadocs.

Usage example

Please consult connector configuration documentation

HTTPS

For general information on Jetty HTTPS/SSL configuration, please read this document. For configuration of the connector in a Restlet component, you will need to set some of the HTTPS parameters listed above, for example:

```
Server server = myComponent.getServers().add(Protocol.HTTPS, 8183);

server.getContext().getParameters().add("keystorePath", "<your-path>");

server.getContext().getParameters().add("keystorePassword",
"<your-password>");

server.getContext().getParameters().add("keyPassword",
"<your-password>");
```

FreeMarker extension

This extension provides an integration with FreeMarker 2.3. FreeMarker is a template engine, a generic tool to generate text output (anything from HTML to autogenerated source code) based on templates. For details on FreeMarker, see the home page.

When you need to generate a dynamic document based on a data model and a FreeMarker template (skeleton document similar to a JSP page or an XSLT stylesheet), you just need to create an instance of the TemplateRepresentation class with the matching parameters and to set it as the response entity.

For additional details, please consult the Javadocs. # JSON extension

JSON (JavaScript Object Notation) is a compact format for representing structured data. It became a popular alternative to XML in AJAX applications due to its native support by Web browser. JSON is indeed based on the JavaScript syntax. See JSON project for more details.

This extension adds support for JSON representations. It enables you to create JsonRepresentation instances from the two kinds of structure defined by JSON: objects and arrays, and to make the reverse conversion. JsonRepresentations can also be generated from String and any kind of Representation.

For additional details, please consult the Javadocs.

Servlet extension

Introduction

This extension is an adapter between the Servlet API and the Restlet API. It allows you to nearly transparently deploy a Restlet application in your preferred Servlet container such as Apache Tomcat.

Description

It enables you to run a Restlet based application inside any Servlet container such as Tomcat.

For additional details, please consult the Javadocs.

JiBX extension

Introduction

This extension provides an integration with JiBX. JiBX is a very flexible framework for binding XML data to Java objects.

Description

The extension is composed of just one class, the JibxRepresentation that extends the XmlRepresentation and is able to both serialize and deserialize a Java objects graph to/from an XML document.

For additional details, please consult the Javadocs.

APISpark extension

Introduction

This extension provides a tool to import the contract of your Restlet Web API in the APISpark full-stack PaaS for Web APIs.

This will allow you to:

- Introspect your Restlet-based Web API to retrieve documentation
- Display and edit this documentation within APISpark

- Synchronize Web API changes initiated from Restlet

In this scenario, we'll load a WebAPI definition into APISpark. We'll leverage the introspector tool provided in the APISpark extension. You can find a complete example of documentation generated via this extension here, the description fields cannot be retrieved from the code, they were added manually.

Launch process

In this example, we'll document a Web API whose Application class is MyContacts

```
java -cp "/path/to/your/lib/*" org.restlet.ext.apispark.Introspector -u
55955e02-0e99-47f8 -p 6f3ee88e-8405-44c8 org.restlet.api.MyContacts
```

Configuration

You must add the following jars (provided in restlet framework) in the "/path/to/your/lib" folder or manually to the classpath.

In Restlet framework lib directory:

- org.restlet.jar (Restlet API)
- org.restlet.ext.apispark.jar (Restlet APISpark extension with Introspector class)
- org.restlet.ext.jackson.jar (Restlet Jackson extension)
- org.restlet.ext.xml (Restlet xml extension in Restlet framework lib directory)
- org.restlet.ext.wadl (Restlet xml extension in Restlet framework lib directory)

In Restlet framework lib/com.fasterxml.jackson_2.2/ directory:

- com.fasterxml.jackson.annotations.jar
- com.fasterxml.jackson.core.jar
- com.fasterxml.jackson.csv.jar
- com.fasterxml.jackson.databind.jar
- com.fasterxml.jackson.smile.jar
- com.fasterxml.jackson.xml.jar
- com.fasterxml.jackson.yaml.jar

Your packaged Web API:

- org.restlet.api.jar (your packaged Web API)

APISpark tokens

The parameters -u and -p are mandatory, they correspond to your APISpark user name and secret key. You can get those here under the tab "tokens". You will need to sign up first.

Load Web API definition into APISpark (first call)

Here is the result, we get from the Introspector:

Process successfully achieved.

Your Web API contract's id is: 246

Your Web API documentation is accessible at this URL:
https://apispark.com/apis/246/versions/1

Update your Web API definition (Subsequent calls)

Web API definition

You need to add a parameter -d giving the id of the definition, hosted on APISpark, that you want to update. You can find the parameter -d in two ways.

- It will be in the response body when you first use the extension on your API.
- If you did not write it down then you can go to your dashboard, click on the Web API Contract you want to update and get it from the URL. The URL should look like this: https://apispark.com/apis/[definition_id]/version/1/

Debug the Web API introspection

If you want the introspector to display suggestions on how to improve your documentation, you can provide this parameter to the java command line:

-Djava.util.logging.config.file="/path/to/logging/properties/logging.properties"

More about the Introspector Tool

The Restlet extension for APISpark provides a source code introspector that takes a class (your Restlet class extending the class Application)

from your Web API as a parameter and instantiates its components to retrieve the contract of your API.

Here is its commande line help:

SYNOPSIS

org.restlet.ext.apispark.Introspector [options] APPLICATION

DESCRIPTION

Publish to the APISpark platform the description of your Web API, represented by APPLICATION, the full canonical name of your Restlet application class.

If the whole process is successfull, it displays the url of the corresponding documentation.

OPTIONS

-h

Prints this help.

-u

The mandatory APISpark user name.

-p

The mandatory APISpark user secret key.

-s

The optional APISpark platform URL (by default https://apispark.com).

-c

The optional full canonical name of your Restlet Component class. This allows to collect some other data, such as the endpoint.

-d

The optional id of an existing definition hosted by APISpark you want to update with this new documentation.

XStream extension

Introduction

This extension provides and integration of Restlet with XStream. XStreamis a simple library to serialize objects to XML or JSON and back again. For JSON support, it depends on Jettison 1.0 (note that usage Jettison 1.1 with this version of XStream is discouraged).

For additional details, please consult the Javadocs.

Usage instructions

The extension comes with an XstreamRepresentation that can either:

- wrap a Java object to serialize as XML or JSON
- wrap an XML or JSON representation to parse as a Java object

It also provides a plugin for the ConverterService which will automatically serialize and deserialize your Java objets returned by annotated methods in ServerResource subclasses.

Here is an example server resource:

```
import org.restlet.Server;

import org.restlet.data.Protocol;

import org.restlet.resource.Get;

import org.restlet.resource.Put;

import org.restlet.resource.ServerResource;


public class TestServer extends ServerResource {


   private static volatile Customer myCustomer = Customer.createSample();


   public static void main(String[] args) throws Exception {

      new Server(Protocol.HTTP, 8182, TestServer.class).start();

   }


   @Get
```

```
    public Customer retrieve() {
        return myCustomer;
    }

    @Put
    public void store(Customer customer) {
        myCustomer = customer;
    }

}
```

Here is the matching client resource:

```
import org.restlet.resource.ClientResource;
import org.restlet.resource.ResourceException;

public class TestClient {

    /**
     * @param args
     * @throws ResourceException
     */
    public static void main(String[] args) throws Exception {

        ClientResource cr = new ClientResource("http://localhost:8182");

        // Retrieve a representation
        Customer customer = cr.get(Customer.class);
        System.out.println(customer);
```

```
        // Update the target resource
        customer.setFirstName("John");
        customer.setLastName("Doe");
        cr.put(customer);

        // Retrieve the updated version
        customer = cr.get(Customer.class);
        System.out.println(customer);
    }

}
```

Note that our Customer and Address classes are just regular serializable beans, with no special parent classes and no special annotations.

Customization

What is nice is that the automatically generated JSON and XML representations can be customized via XStream mechanisms such as manual settings on the XstreamRepresentation#xstream object or via XStream annotations on the serialized beans. More details on annotations are available in XStream documentation.

Tutorial

Target audience

Java developers that want to access OData services in Java

Author

Thierry Boileau, co-founder of Restlet and core committer on the Restlet open source project.

Topics covered

- Accessing an OData service in Java with Restlet
- Handling associations between entities (if any)
- Handling queries
- Accessing secured services

Table of contents

Resources

- Javadocs of the Restlet extension for OData Services
- MSDN documentation on WCF Data Services

Introduction

REST can play a key role in order to facilitate the interoperability between Java and Microsoft environments. To demonstrate this, the Restlet team Restlet team worked with Microsoft in order to provide a new Restlet extension that will provide several high level features for accessing WCF Data Services.

Figure 1 - The WCF Data Services Framework Architecture

Initially known as “Astoria” then ADO.NET Data Services, the WCF Data Services technology has become the preferred way to RESTfully expose data sources (relational databases, in-memory data, XML files, etc.) in the .NET framework.

The “org.restlet.ext.odata” extension for the Restlet Framework provides a client API to access remote WCF Data Services or other services supporting the OData protocol. It allows access to the exposed entities, to their properties and their associations (when an entity is linked to other entities). We will illustrate this with a sample WCF Data Service with the following root URI: http://restlet.cloudapp.net/TestAssociationOneToOne.svc/

Note that the Azure service mentionned above isn't available anymore as the account has expired. Sorry for the inconvenience.

This service defines two kinds of entities:

- “Cafe”, and
- “Item”.

A “Cafe” defines several properties such as “ID”, “Name”, “City”, and an Item is simply defined by an “ID” and a “Description”. Cafe and Item are linked, for intentional simplification, in a “one to one” association. That is to say, a “Cafe” has one attribute called “Item”.

Figure 2 - Class diagram of the TestAssociationOneToOne service

Code generation

From the client perspective, if you want to handle the declared entities, you will have to create a class for each entity, defines their attributes, and pray that you have correctly spelled them and defined their type. Thanks to the Restlet extension, a generation tool will make your life easier. It will take care of this task for you, and generate the whole set of Java classes with correct types.

Figure 3 - Overview of code generation

Just note the URI of the target service, and specify the directory where you would like to generate the code:

```
java -jar org.restlet.ext.odata.jar
http://restlet.cloudapp.net/TestAssociationOneToOne.svc/
~/workspace/testADO
```

Please note that this feature requires the use of the core Restlet, and additional dependencies such as Atom (used by OData services for all exchanges of data), XML (required by the Atom extension) and FreeMarker (used for the files generation). The following jars (take care of the names) must be present on the current directory:

- org.restlet.jar (core Restlet)

- org.restlet.ext.odata.jar (OData extension)
- org.restlet.ext.atom.jar (Atom extension)
- org.restlet.ext.xml.jar (XML extension)
- org.restlet.ext.freemarker.jar (Freemarker extension)
- org.freemarker.jar (Freemarker dependency)

You can also used the full command line that includes the list of required archives for the class path argument (nb: take care of the OS specific classpath separator) and the name of the main class:

```
java -cp
org.restlet.jar:org.restlet.ext.xml.jar:org.restlet.ext.atom.jar:org.restlet.ext.fre
emarker.jar:org.restlet.ext.odata.jar:org.freemarker.jar
org.restlet.ext.odata.Generator

 http://restlet.cloudapp.net/TestAssociationOneToOne.svc/

 ~/workspace/testADO
```

This will generate the following java classes and directory:

```
testAssociationOneToOne/

  +-- Cafe.java

  +-- Item.java

TestAssociationOneToOneQuery.java
```

The classes that correspond to entities are generated in their corresponding package (in our case: “testAssociationOneToOne”), as defined by the meta data of the target Data Service. The last class (“TestAssociationOneToOneService”) is what we call a service object. Such object is able to handle the communication with the data service, and is able to store the state of the latest executed request and the corresponding response. The communication between the client and the server is still stateless.

Before using the generated classes, let’s explain how the Java code is generated from the metadata of the target OData service. Actually, the Generator class extracts a few elements of the metadata such as the schema and the entities as follow.

WCF concept

Java concept

Entity

Class

Entity name

Class name

Enclosing schema namespace

Package name

Entity property

Member variable with getter and setter

EDM data type

Java primitive types and classes

Transformation table from WCF concepts to Java equivalent concepts

Regarding the conversion of the data type, an equivalence table has been established as follow:

EDM data type

Binary

Boolean

DateTime

Decimal

Single

Double

Guid

Int16

Int32

Int64

Byte

String

Data type conversion table

We have finished for now of the theoretical aspects; let's see how to use the generated classes.

Get the current set of Cafes and Items

The code below gets the whole set of Cafe entities and displays some of their properties. It will display this kind of output on the console:

id: 1

name: Le Café Louis

id: 2

name: Le Petit Marly

```
TestAssociationOneToOneService service = new
TestAssociationOneToOneService();

Query<Cafe> query = service.createCafeQuery("/Cafes");


for (Cafe Cafe : query) {

   System.out.println(“id: ” + Cafe.getID());

   System.out.println(“name: ” + Cafe.getName());

}
```

Retrieve the set of “Cafe” entities.

The first step is the creation of a new service. This is the only required action, and it must be done once, but prior to any other one. Then, as, we want to get a set of “Cafe”, we just create a new query and specify the desired data. Under the hood, it actually makes a GET request to the “/Cafes” resource (relatively to the service's URI), and receive as a result a AtomXML feed document. This document is parsed by the query which provides the result as an Iterator. Finally, we can loop over the iterator and access to each “Cafe” instance.

Here is the content of the HTTP request:

GET /TestAssociationOneToOne.svc/Cafes HTTP/1.1

Host: restlet.cloudapp.net

User-Agent: Noelios-Restlet/2.0snapshot

Accept: */*

Connection: close

And here is the response of the server including both response headers and entity:

HTTP/1.1 200 OK

Cache-Control: no-cache

Content-Type: application/atom+xml;charset=utf-8

Server: Microsoft-IIS/7.0

DataServiceVersion: 1.0;

X-AspNet-Version: 2.0.50727

X-Powered-By: ASP.NET

Date: Fri, 24 Jul 2009 14:21:20 GMT

Connection: close

Content-Length: 2221

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<feed xml:base="http://restlet.cloudapp.net/TestAssociationOneToOne.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">

<title type="text">Cafes</title>

<id>http://restlet.cloudapp.net/TestAssociationOneToOne.svc/Cafes</id>

<updated>2009-07-24T14:21:20Z</updated>

<link rel="self" title="Cafes" href="Cafes" />

<entry>

<id>http://restlet.cloudapp.net/TestAssociationOneToOne.svc/Cafes('1')</id>

<title type="text"></title>

<updated>2009-07-24T14:21:20Z</updated>

```
    <author>
      <name />
    </author>
    <link rel="edit" title="Cafe" href="Cafes('1')" />
    <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Item"
          type="application/atom+xml;type=entry" title="Item"
href="Cafes('1')/Item" />
    <category term="TestAssociationOneToOne.Cafe"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" /
>
    <content type="application/xml">
      <m:properties>
        <d:ID>1</d:ID>
        <d:Name>Le Café Louis</d:Name>
        <d:ZipCode m:type="Edm.Int32">92300</d:ZipCode>
        <d:City>Levallois-Peret</d:City>
      </m:properties>
    </content>
  </entry>
  <entry>

<id>http://restlet.cloudapp.net/TestAssociationOneToOne.svc/Cafes('2')</id>
    <title type="text"></title>
    <updated>2009-07-24T14:21:20Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Cafe" href="Cafes('2')" />
    <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Item"
```

```
        type="application/atom+xml;type=entry" title="Item"
href="Cafes('2')/Item" />

   <category term="TestAssociationOneToOne.Cafe"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" /
>

   <content type="application/xml">

     <m:properties>

       <d:ID>2</d:ID>

       <d:Name>Le Petit Marly</d:Name>

       <d:ZipCode m:type="Edm.Int32">78310</d:ZipCode>

       <d:City>Marly Le Roi</d:City>

     </m:properties>

   </content>

 </entry>

</feed>
```

Getting the set of defined “Item” is quite similar:

```
Query<Item> queryItem = service.createItemQuery("/Items");


for (Item Item : queryItem) {

   System.out.println(“id: ” + Item.getID());

   System.out.println(“desc.: ” + Item.getDescription());

}
```

Retrieve the set of “Item” entities.

Please note that for the rest of the document, we assume the “service” object has already been instantiated.

Get a single Entity

Imagine we want to retrieve the first “Cafe” of the list, that is to say, the one which identifier is equal to “1”. As for the set of entities, you just have to create a new query, with a new parameter. The code below should produce this output:

```
id: 1
```

name: Le Café Louis

```
Query<Cafe> query = service.createCafeQuery("/Cafes('1')");


Cafe Cafe = query.iterator().next();

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());
```

Retrieve the “Cafe” by its identifier.

As for a set of entities, you have to create a new query, and precise the identifier of the target resource. WCF adopts its own naming convention. You surely notice that the identifier is enclosed between “(“ and “)” characters.

Now, you can ask the query to return the entity with a call to the “next” method, and if the entity exists, you can print its properties. An additional call to “next” will provide a “null” result.

Add a new Entity

Let's complete the current list of entities and add a new one. This process is quite simple and just requires you to firstly create and complete the new entity, then invoke the “addEntity” method as follow.

```
Cafe Cafe = new Cafe();

Cafe.setID("3");

Cafe.setZipCode(12345);

Cafe.setName("Bar des sports");

Cafe.setCity("Paris");


service.addEntity(Cafe);
```

Add a new Cafe.

The generated subclass of Service contains a dedicated method for each declared entities. In our case, there are two of them, one for the Cafe class, and the other for the Item class. Such method actually sends a POST request to the corresponding entity set resource. For example, adding a new Café sends a POST request to the “/Cafes” resource. Here is the sample content of such generated request:

POST /TestAssociationOneToOne.svc/Cafes HTTP/1.1

Host: restlet.cloudapp.net

User-Agent: Noelios-Restlet/2.0snapshot

Accept: */*

Content-Type: application/atom+xml

Transfer-Encoding: chunked

Connection: close

281

<?xml version="1.0" standalone='yes'?>

<entry xmlns="http://www.w3.org/2005/Atom">

  <content type="application/xml"><properties
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">

<ZipCode
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">12345</Zi
pCode>

<ID
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">3</ID>

<Name
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">Bar des
sports</Name>

<City
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">Paris</Cit
y>

<Article
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices"/></propert
ies></content>

</entry>

0

Before using this feature, ensure that you provide a correctly identified object, especially, don’t try to add an entity with a null value. It appears that the remote service does not prevent such cases which could lead to irremediably mess the content of the entity set.

Update an Entity

Once you have retrieved an entity, imagine that you want to update one of its properties. The sample code below illustrates this with the “Nom” property. It simply uses the “updateEntity” method. You can check that the value has really been taken into account by making a new query.

```
Query<Cafe> query = service.createCafeQuery("/Cafes('1')");


Cafe Cafe = query.iterator().next();

Cafe.setNom("Bar des sports");


service.updateEntity(Cafe);
```

Update a Cafe.

Under the hood, a PUT request is sent to the corresponding Web resource.

Delete an Entity

Let's finish the tour of the basic operations with the deletion of an entity. You just need to wall the deleteEntity method as shown just below.

```
Query<Cafe> query = service.createCafeQuery("/Cafes('1')");

Cafe Cafe = query.iterator().next();


service.deleteEntity(Cafe);
```

Delete a Cafe.

Basically, this operation requires a valid Java Object instance correctly identified: that is to say, the attribute that serves as identifier must be correctly valuated.

Get a single Entity and its associated entities

Until now, we looked at entities and their basic properties. However, we pointed out at the beginning of this Item that a “Cafe” also aggregates “Item” entities. You may have noticed, during your own tests, that the Item property was always null. By default, associations are not expanded, but they can be. If you run the following code, you will get this kind of trace at the console.

Cafe

id: 1

name: Le Café Louis

Item

id: 1

Description: Poulet au curry

```
Query<Cafe> query = service.createCafeQuery("/Cafes('1')").expand("Item");


Cafe Cafe = query.next();

System.out.println("Cafe");

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

System.out.println("Item");

System.out.println(“id: ” + Cafe.getItem().getID());

System.out.println(“Description: ” + Cafe.getItem().getDescription());
```

Retrieve a “Cafe” by its identifier, with the associated Item in one request.

Instead of just creating a query as seen above, you can complete it by calling the “expand” method. It takes one parameter which is the name of the entity attribute you want to expand. Please note that you can also invoke this when requesting the “/Cafes” entity set.

Other kinds of queries

The expansion of query is one of the features provided by the OData protocol. Other ones are available that aim at ordering, filtering, and limiting the set of the results returned by a query. This section is an exhaustive list of those available features.

Order

Let's say you want to order a list of ItemItems by their “description” attribute. For example, once applied to the set of all Items provided by our sample OData service, the following code should display at the console this king of trace:

id: 2

description: Pâté

id: 1

description: Poulet au curry

```
Query<Item> query =
service.createItemQuery("/Items").orderby("Description");


for (Item Item : query) {
    System.out.println(“id: ” + Item.getID());
    System.out.println(“description: ” + Item.getDescription());
}
```

Order a set of entities.

Just as the “expand” method, it takes one parameter which is the name of the property used to order the set of results.

Filter

Do you want to filter the set of results? Using the “filter” method, you will be able to specify one or more criteria that the entities returned by the query must share. You will express these constraints using LINQ (Language-Integrated Query). LINQ has been originally designed to bridge the gap between the world of data and the world of objects (in the context of C# and Visual Basic programs). In the context of OData services, it helps filtering a set of data. Its syntax is very close to SQL's one.

Let's illustrate its use by limiting the set of Cafe objects to the one (there should be only one) having its ID equals to “1”. Applied to the list of current Cafes, the following code will produce this display on the console:

id: 1

nom: Le Café Louis

```
Query<Cafe> query = service.createCafeQuery("/Cafes").filter("Name eq 'Le
Café Louis'");
```

```
for (Cafe Cafe : query) {
   System.out.println(“id: ” + Cafe.getID());
   System.out.println(“nom: ” + Cafe.getNom());
}
```

Filter a set of entities

Skip

The “skip” method takes one parameter which is a number, and simply allows you to omit the first elements of the set theoretically returned by the query.

Let's say you want to omit the first “Cafe” of the list, just call the “skip” method as shown in the sample code below. It should display this kind of trace at the console.

```
id: 2
name: Le Petit Marly
```

```
Query<Cafe> query = service.createCafeQuery("/Cafes").skip(1);

for (Cafe Cafe : query) {
   System.out.println(“id: ” + Cafe.getID());
   System.out.println(“name: ” + Cafe.getName());
}
```

Skip the first entity

Top

Just as the “skip” method, “top” takes a number parameter which represents the maximum number of results that the query will return. Its use is very simple as shown below:

```
id: 1
name: Le Café Louis
```

```
Query<Cafe> query = service.createCafeQuery("/Cafes").top(1);
```

```
for (Cafe Cafe : query) {
    System.out.println(“id: ” + Cafe.getID());
    System.out.println(“name: ” + Cafe.getName());
}
```

Limit the number of returned entities.

The “skip” and “top” can be used together for paging a set of results.

Support of projections

Projections allow you to retrieve a subset of the properties of queried entities. It is a way to reduce the amount of data transfered from the server.

You can restrict your query to set of properties, either simple or complex or even navigation properties (links to other entities). The OData protocol specifies the usage of the **$select** query parameter here.

The result of such projection is a list of entities as usual, except that these entities are only populated with the selected properties.

For example, the following code will only get the name of the Cafe entities:

```
id: null
name: Le Café Louis
```

```
Query<Cafe> query =
service.createCafeQuery("/Cafes").top(1).select("Name");
```

```
for (Cafe Cafe : query) {
    System.out.println(“id: ” + Cafe.getID());
    System.out.println(“name: ” + Cafe.getName());
}
```

Limit the number of returned properties.

This applies also to associated entities, as far as the association is expanded:

```
id: null
```

name: Le Café Louis

Item

id: null

Description: Poulet au curry

```
Query<Cafe> query =
service.createCafeQuery("/Cafes").top(1).expand("Item").select("Name,Item.
Description");


for (Cafe Cafe : query) {
    System.out.println(“id: ” + Cafe.getID());
    System.out.println(“name: ” + Cafe.getName());
    System.out.println("Item");
    System.out.println(“id: ” + Cafe.getItem().getID());
    System.out.println(“Description: ” + Cafe.getItem().getDescription());
}
```

Limit the number of returned properties.

Server-side paging

This mechanism also the server to limit the amount of data to transfer in the response to a query made on an entity set. This is very interesting if the set can contain a large set of entities. Unfortunately, this feature has no impact on the way you use the Query object of the OData extension. This is made transparent for the user.

Get the row count

By invoking the **getCount** method on the Query object, you can retrieve the number of entities contained by the target entity set. having said that, you must be aware that there are two ways to get the number of entities of an entity set (once filtering, if any, have been applied). If the service supports the inlinecount feature, this count is obtained from the Feed document itself. This allows to retrieve both count data and entries in the same request. Another way is to send a dedicated request using the $count segment. You can control this behaviour by invoking the **inlineCount** method on the Query object. It accepts as a parameter a boolean value indicating that you want to retrieve the count number using the *inlinecount* query parameter (set

the parameter to "true") or directly using the *$count* segment (set the parameter to "false", this is the value by default). The following sample code illustrates how to get the count using the *inlinecount* query parameter.

```
Query<Cafe> query = service.createCafeQuery("/Cafes").inlineCount(true);


System.out.println("Number of entities: " + query.getCount());
```

Get the number of Cafes.

Please note that the *inlinecount* query parameter is not supported by every service.

Support of customizable feeds

Customizable feeds is a server-side feature that address specific use-case that requires that the Atom data feeds produced by the data service will use both standard Atom elements or custom feed elements. This of course has an impact on the way the Atom document must be parsed, but this has no impact on the way you use the OData extension.

Handling blobs (aka media link entries)

Media link entries are specific entities that allow to handle binary content such as images, documents, etc. Each entity can be seen as collection of metadata about the binary content, plus the binary content itself. Since the metadata are stored as basic attribute members of the entity class (as usual), there must be a specific way to handle the binary content. At that point the Service class introduces a set of new methods to handle the "value" of a blob entity:

- getValueRef(Object) that returns the URI of the target binary content
- getValue(Object) and getValue(Object, *) methods that allow you to retrieve as a Representation the binary content of a given entity.

Access to secured services

WCF Data Services can choose their security option. In order to access those kinds of services, you must have been given the following information:

- login and password
- security scheme such as HTTP BASIC, Windows Shared Key and Shared Key Lite, etc.

These data must be supplied to the service instance which takes care of all communications with the target service. This is generally done once, after the instantiation of the service. Here is an illustration of how to set the credentials:

```
service.setCredentials(
        new ChallengeResponse(ChallengeScheme.HTTP_BASIC,
                "login",
                "password")
        );


Query<Cafe> query = service.createCafeQuery("/Cafes").top(1);


for (Cafe Cafe : query) {
   System.out.println(“id: ” + Cafe.getID());
   System.out.println(“nname: ” + Cafe.getName());
}
```

Add credentials to access a secured service.

Note that Restlet framework supports a wide set of security schemes, including HTTP BASIC, HTTP DIGEST, Windows Shared Key and Shared Key Lite.

Access to NTLM secured services

Please refer to this document for how to access NTML secured services.

Support of capability negotiation based on protocol versions

As stated by the WCF documentation (here), client and server may talk different dialects of the OData protocol. The Service class provides several methods that allow you to control these aspects:

- getClientVersion and setClientVersion to handle the supported version of OData protocol.
- getMaxClientVersion and setMaxClientVersion to handle the max number version od supported version of the OData protocol
- getServerVersion to get the version of the dialect talked by the server.

The values setted by setClientVersion and setMaxClientVersion are not controlled by the framework, but they must follow a format specified here.

Conclusion

This article illustrates what can be done with the Restlet extension for OData services. We hope that you found it simple and useful to follow to read. It is a good demonstration of how adopting of REST and related standards such as HTTP and Atom facilitates the interoperability across programming languages and executions environments.

OData extension

Topics covered

- Accessing the OGDI data in Java with Restlet
- Handling queries

References

- Blog post - Restlet supports OData, the Open Data Protocol
- Javadocs - Restlet extension for OData
- Advanced tutorial on the OData extension
- OData - Protocol specification
- MSDN - WCF Data Services
- Open Government Data Initiative project

Table of contents

Introduction

REST can play a key role in order to facilitate the interoperability between Java and Microsoft environments. To demonstrate this, the Restlet team collaborated with Microsoft in order to build a new Restlet extension that provides several high level features for accessing OData services (Open Data Protocol).

The Open Government Data Initiative (OGDI) is an initiative led by Microsoft. OGDI uses the Azure platform to expose a set of public data from several government agencies of the United States. This data is exposed via a restful API which can be accessed from a variety of client technologies, in this case Java with the dedicated extension of the Restlet framework. The rest of the article shows how to start with this extension and illustrates its simplicity of use.

The OGDI service is located at this URI “http://ogdi.cloudapp.net/v1/dc” and exposes about 60 kinds of public data gathered in entity sets. Here are samples of such data:

- Ambulatory surgical centers (here is the name of the corresponding entity set: “/AmbulatorySurgicalCenters” relatively to the service root URI),
- Building permits (“/BuildingPermits”)
- Fire stations (“/FireStations”),
- etc.

Code generation

From the client perspective, if you want to handle the declared entities, you will have to create a class for each entity, defines their attributes, and pray that you have correctly spelled them and defined their type. Thanks to the Restlet extension, a generation tool will make your life easier. It will take care of this task for you, and generate the whole set of Java classes with correct types.

Just note the URI of the target service, and specify the directory where you would like to generate the code via the command line:

java -jar org.restlet.ext.odata Generator http://ogdi.cloudapp.net/v1/dc/ ~/workspace/testADO

Please note that this feature requires the use of the core Restlet, and additional dependencies such as Atom (used by OData services for all exchanges of data), XML (required by the Atom extension) and FreeMarker (used for the files generation). The following jars (take care of the names) must be present on the current directory:

- org.restlet.jar (core Restlet)
- org.restlet.ext.odata.jar (OData extension)
- org.restlet.ext.atom.jar (Atom extension)
- org.restlet.ext.xml.jar (XML extension)

- org.restlet.ext.freemarker.jar (Freemarker extension)
- org.freemarker.jar (Freemarker dependency)

You can also used the full command line that includes the list of required archives for the class path argument (nb: take care of the OS specific classpath separator) and the name of the main class:

```
java -cp
org.restlet.jar:org.restlet.ext.xml.jar:org.restlet.ext.atom.jar:org.restlet.ext.freemarker.jar:

 org.restlet.ext.odata.jar:org.freemarker.jar org.restlet.ext.odata.Generator

 http://ogdi.cloudapp.net/v1/dc/

 ~/workspace/testADO
```

or programmatically:

```
String[] arguments =

    { "http://ogdi.cloudapp.net/v1/dc/",

      "/home/thierry/workspace/restlet-2.0/odata/src" };

Generator.main(arguments);
```

Please note that this feature requires the use of the core Restlet, and additional dependencies such as Atom (used by OData services for all exchanges of data), XML (required by the Atom extension) and FreeMarker (used for the files generation). They must rely on the classpath.

This will generate the following Java classes and directory:

```
ogdiDc/

  +-- AmbulatorySurgicalCenter.java

  +-- BuildingPermit.java

  +-- etc


OgdiDcSession.java
```

The classes that correspond to entities are generated in their corresponding package (in our case: “ogdiDc”), as defined by the meta data of the target OData service.

The last class (“OgdiDcSession”) is what we call a session object. Such object is able to handle the communication with the data service, and is able to store the state of the latest executed request and the corresponding response. You probably think that such session looks like a Servlet session. Actually, this is not true. The communication between the client and the server is still stateless.

We have finished for now of the theoretical aspects; let's see how to use the generated classes.

Get the two first building permits

The code below gets the two first entities and displays some of their properties. It will display this kind of output on the console:

```
*** buildingPermit

Owner   :DARYL ADAIR

City    :WASHINGTON

District:THIRD

Address :447 RIDGE ST NW

State   :DC


*** buildingPermit

Owner   :RUTH D PROCTOR

City    :WASHINGTON

District:FIFTH

Address :144 U ST NW

State   :DC
```

The listing below shows how to rRetrieve the two first “BuildingPermits” entities:

```
OgdiDcSession session = new OgdiDcSession();

Query<BuildingPermit> query =

    session.createBuildingPermitQuery("/BuildingPermits").top(2);


if (query != null) {

  for (BuildingPermit buildingPermit : query) {
```

```
        System.out.println("*** building permit");
        System.out.println("Owner   :" + buildingPermit.getOwner_name());
        System.out.println("City    :" + buildingPermit.getCity());
        System.out.println("District:" + buildingPermit.getDistrict());
        System.out.println("Address :" + buildingPermit.getFull_address());
        System.out.println("State   :" + buildingPermit.getState());
        System.out.println();
    }
}
```

The first step is the creation of a new session. This is the only required action, and it must be done once, but prior to any other one. Then, as we want to get a set of “BuildingPermit”, we just create a new query and specify the desired data. In addition, as the set of data is very large, we ask to limit its size by setting the “top” parameter.

Under the hood, it actually makes a GET request to the “/BuildingPermits?top=2” resource (relatively to the service's URI), and receive as a result a AtomXML feed document. This document is parsed by the query which provides the result as an Iterator. Finally, we can loop over the iterator and access to each “BuildingPermit” instance.

Filter the set of the building permits

The code below gets the five first entities located in the city of Washington and more precisely on the fifth district and displays some of their properties. It will display this kind of output on the console:

```
*** building permit
Owner   :RUTH D PROCTOR
Address :144 U ST NW


*** building permit
Owner   :212 36TH ST. LLC.
Address :1250 QUEEN ST NE


*** building permit
```

```
Owner   :GEORGE M CURRY JR TRUSTEE
Address :1902 JACKSON ST NE


*** building permit
Owner   :ADRIENNE WEAVER
Address :336 ADAMS ST NE


*** building permit
Owner   :CARL J HAMPTON
Address :2925 SOUTH DAKOTA AVE NE
```

The listing below shows how to retrieve the five first “BuildingPermits” entities in the fifht district of Washington:

```
Query<BuildingPermit> search =
  session.createBuildingPermitQuery("/BuildingPermits")
    .filter("((city eq 'WASHINGTON') and (district eq 'FIFTH'))")
    .top(5);


if (search != null) {
  for (BuildingPermit buildingPermit : search) {
    System.out.println("*** building permit");
    System.out.println("Owner   :" + buildingPermit.getOwner_name());
    System.out.println("Address :" + buildingPermit.getFull_address());
    System.out.println();
  }
}
```

As we want to get a set of “BuildingPermit”, we just create a new query and specify the desired data. In addition we add a filter based on the expression of two criteria: the name of the city and the district. This filter property uses a subset the WCF Data Services query syntax.

It makes a GET request to the “/BuildingPermits” resource and completes its URI with the addition of a query part including the filter and top parameters. As for the previous example, the received AtomXML feed document is parsed which produces the result as an Iterator. Finally, we can loop over the iterator and access to each “BuildingPermit” instance.

Conclusion

This document illustrates what can be done with the Restlet extension for the OData services. We hope that you found it simple and useful to follow to read. It is a good demonstration of how adopting of REST and related standards such as HTTP and Atom facilitates the interoperability across programming languages and executions environments.

Swagger extension

This extension provides a preview integration with Swagger including: * generation of Swagger descriptor in JSON * introspect JAX-RS API based applications

Additional work is required to: * ship the Swagger UI * parse a Swagger descriptor in JSON to set up an application * extend the introspection to Restlet API based applications

For additional details, please consult the Javadocs. EMF ===

Introduction

Eclipse EMF is a widely used modeling technology that can unify models defined in XML Schema, UML or Java code. This extension facilitates the usage of EMF to convert between "representation beans" generated from EMF and XML/XMI/ECore representations.

For additional details, please consult the Javadocs.

Description

This extension provides a main EmfRepresentation class that is able to both format and EObject as an XML/XMI/HTML representation and to parse an XML/XMI representation is generated back as an EObject. It comes with its EmfConverter which allows you to automatically benefit from the feature by simply adding org.restlet.ext.emf.jar in your classpath and using your EObject subclasses in resource annotated interface.

TODO - Add a simple yet comprehensive example of how to use EMF to save time to manage representations

HTML representations

In addition to XML/XMI support common for EMF model, we offer a generic HTML representation. It lists all its properties and can even generate HTML links when the proper EMF eAnnotation is detected. This is useful to be able to automatically navigate a web API whose resource representations are defined using EMF.

In order to display EMF attributes containing URI references into proper HTML hyperlinks, you just need to add eAnnotations to your EMF ECore metamodel using the "http://restlet.org/schemas/2011/emf/html" namespace and then add an entry with the "linked" name and a "true" value.

In order to change the title of the HTML document or the name of a property, you need to use the same namespace with the "label" name and the text you want to display as a value.

WADL extension

Introduction

This extension support WADL, the Web Application Description Language. It allows you to configure a Restlet component or application based on a WADL/XML document or to dynamically generate the WADL/XML document of an existing Restlet application. It also knows how to convert this WADL/XML document into a user friendly HTML document.

For additional details, please consult the Javadocs.

Usage example

This sample code is an extension of the "firstResource" sample application. The source code is available in the "org.restlet.example" extension, more precisely in the "org.restlet.ext.wadl" package.

The aim is to provide WADL documentation for this application. Basically, we must be able to generate a proper documentation for:

- the whole application
- each single defined resource, that is to say the "ItemsResource" and "ItemResource".

Here is the list of modifications.

FirstResourceApplication class

- Make the FirstResourceApplication class a subclass of WadlApplication.

- Implement the "getApplicationInfo" method in order to add a title and a simple description:

```
@Override
public ApplicationInfo getApplicationInfo(Request request, Response response) {
    ApplicationInfo result = super.getApplicationInfo(request, response);

    DocumentationInfo docInfo = new DocumentationInfo(
            "This sample application shows how to generate online documentation.");
    docInfo.setTitle("First resource sample application.");
    result.setDocumentation(docInfo);

    return result;
}
```

BaseResource class

- Make the BaseResource class a subclass of WADLServerResource, in order to provide the WADL features to all inheriting resources.

ItemsResource

- Implement the "describe" method,in order to set a proper title.

```
@Override
protected Representation describe() {
    setTitle("List of items.");
    return super.describe();
}
```

- Implement the"describeGet" method

```
@Override
protected void describeGet(MethodInfo info) {
    info.setIdentifier("items");
    info.setDocumentation("Retrieve the list of current items.");
```

```
        RepresentationInfo repInfo = new
RepresentationInfo(MediaType.TEXT_XML);
        repInfo.setXmlElement("items");
        repInfo.setDocumentation("List of items as XML file");
        info.getResponse().getRepresentations().add(repInfo);
    }
```

- Implement the "describePost" method

```
    @Override
    protected void describePost(MethodInfo info) {
        info.setIdentifier("create_item");
        info.setDocumentation("To create an item.");

        RepresentationInfo repInfo = new RepresentationInfo(
                MediaType.APPLICATION_WWW_FORM);
        ParameterInfo param = new ParameterInfo("name",
ParameterStyle.PLAIN,
                "Name of the item");
        repInfo.getParameters().add(param);
        param = new ParameterInfo("description", ParameterStyle.PLAIN,
                "Description of the item");
        repInfo.getParameters().add(param);
        repInfo.getStatuses().add(Status.SUCCESS_CREATED);

        repInfo.setDocumentation("Web form.");
        info.getRequest().getRepresentations().add(repInfo);

        FaultInfo faultInfo = new FaultInfo(Status.CLIENT_ERROR_NOT_FOUND);
        faultInfo.setIdentifier("itemError");
```

```
        faultInfo.setMediaType(MediaType.TEXT_HTML);
        info.getResponse().getFaults().add(faultInfo);
    }
```

ItemResource

- Implement the "describe" method,in order to set a proper title.

```
    @Override
    public Representation describe() {
        setTitle("Representation of a single item");
        return super.describe();
    }
```

- Implement the"describeGet" method

```
    @Override
    protected void describeGet(MethodInfo info) {
        info.setIdentifier("item");
        info.setDocumentation("To retrieve details of a specific item");

        RepresentationInfo repInfo = new
RepresentationInfo(MediaType.TEXT_XML);
        repInfo.setXmlElement("item");
        repInfo.setDocumentation("XML representation of the current item.");
        info.getResponse().getRepresentations().add(repInfo);

        FaultInfo faultInfo = new FaultInfo(Status.CLIENT_ERROR_NOT_FOUND,
                "Item not found");
        faultInfo.setIdentifier("itemError");
        faultInfo.setMediaType(MediaType.TEXT_HTML);
        info.getResponse().getFaults().add(faultInfo);
    }
```

- Implement the "describeDelete" method

```
    @Override
    protected void describeDelete(MethodInfo info) {
        info.setDocumentation("Delete the current item.");

        RepresentationInfo repInfo = new RepresentationInfo();
        repInfo.setDocumentation("No representation is returned.");
        repInfo.getStatuses().add(Status.SUCCESS_NO_CONTENT);
        info.getResponse().getRepresentations().add(repInfo);
    }
```

- Implement the "describePut" method

```
    @Override
    protected void describePut(MethodInfo info) {
        info.setDocumentation("Update or create the current item.");

        RepresentationInfo repInfo = new RepresentationInfo(
                MediaType.APPLICATION_WWW_FORM);
        ParameterInfo param = new ParameterInfo("name",
ParameterStyle.PLAIN,
                "Name of the item");
        repInfo.getParameters().add(param);
        param = new ParameterInfo("description", ParameterStyle.PLAIN,
                "Description of the item");
        repInfo.getParameters().add(param);
        repInfo.getStatuses().add(Status.SUCCESS_OK);
        repInfo.getStatuses().add(Status.SUCCESS_CREATED);

        repInfo.setDocumentation("Web form.");
        info.getRequest().getRepresentations().add(repInfo);
```

```
        super.describePut(info);

    }
```

Getting the documentation

The documentation is available via the OPTIONS method. and is available under 2 formats: WADL or HTML.

WADL documentation

Here is a way to programmatically obtain the WADL documentation of the application:

```
ClientResource app = new
ClientResource("http://localhost:8182/firstResource");



// Displays the WADL documentation of the application

app.options().write(System.out);
```

Here is a way to programmatically obtain the WADL documentation of the "items" resource:

```
ClientResource items = new
ClientResource("http://localhost:8182/firstResource/items");



// Displays the WADL documentation of the application

items.options().write(System.out);
```

HTML documentation

This format is an XML transformation of the WADL representation with an XSL sheet, developped and maintained by Mark Nottingham: here

Here is a way to programmatically obtain the HTML documentation of the application:

```
ClientResource app = new
ClientResource("http://localhost:8182/firstResource");



// Displays an HTML documentation of the application

app.options(MediaType.TEXT_HTML).write(System.out);
```

In order to work properly, you will certainly have to update your classpath with the archive of a convenient transformation engine. Xalan 2.7.1 has been successfully tested.

Advanced topics

Indicate schema relative to an XML representation

A bit of theory

Quoting the WADL specification:

The "grammars" element acts as a container for definitions of the format of data

exchanged during execution of the protocol described by the WADL document. Such

definitions may be included inline or by reference using the include element.

For example:

```
<grammars>
  <include href="NewsSearchResponse.xsd"/>
  <include href="Error.xsd"/>
</grammars>
```

At this time, the WADL extension of the Restlet framework supports only "included" and not "inline" schemas via the GrammarsInfo#includes attribute.

Then, for XML-based representations, the "element" attribute specifies the qualified name of the root element as described within the grammars section.

For example:

```
<representation mediaType="application/xml" element="yn:ResultSet"/>
```

Assuming that the "yn" namespace is declared in the document:

```
<application [...] xmlns:yn="urn:yahoo:yn" >
```

Implementation with Restlet

At the level of the subclass of WadlApplication, override the getApplicationInfo method:

```
@Override
public ApplicationInfo getApplicationInfo(Request request, Response response) {
    ApplicationInfo appInfo = super.getApplicationInfo(request, response);
```

```
    appInfo.getNamespaces().put("urn:yahoo:yn", "yn");
    GrammarsInfo grammar = new GrammarsInfo();
    IncludeInfo include = new IncludeInfo();
    include.setTargetRef(new Reference("NewsSearchResponse.xsd"));
    grammar.getIncludes().add(include);
    appInfo.setGrammars(grammar);
    return appInfo;
}
```

Then, at the level of the subclass of WadlResource, update the RepresentationInfo#element attribute:

```
RepresentationInfo formRepresentation = new RepresentationInfo();
formRepresentation.setXmlElement("yn:ResultSet");
```

jSSLutils extension

Introduction

The SSL extension provides concrete implementations of the SslContextFactory that rely on jSSLutils.

For additional details, please consult the Javadocs.

Description

JsslutilsSslContextFactory

The JsslutilsSslContextFactory class is a wrapper for jsslutils.sslcontext.SSLContextFactory. It has to be constructed with the instance to wrap and is therefore only suitable to be used in the context sslContextFactory *attribute*, not *parameter*. This is more likely to be used for more specialised features such as the key or trust manager wrappers of jSSLutils.

PkixSslContextFactory

The PkixSslContextFactory class is a class that uses jsslutils.sslcontext.PKIXSSLContextFactory. It provides a way to configure the key store, the trust store (required for client-side authentication) and the server alias. As part of its trust manager configuration, it provides a way to set up certificate revocation lists (CRLs).

Example using the Component XML configuration:

```
<component xmlns="http://www.restlet.org/schemas/1.1/Component"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.restlet.org/schemas/1.1/Component">

    <client protocol="FILE" />

    <server protocol="HTTPS" port="8183">
        <parameter name="sslContextFactory"
value="org.restlet.ext.ssl.PkixSslContextFactory" />
        <parameter name="keystorePath" value="/path/to/keystore.p12" />
        <parameter name="keystoreType" value="PKCS12" />
        <parameter name="keystorePassword" value="testtest" />
        <parameter name="keyPassword" value="testtest" />
        <parameter name="truststorePath" value="/path/to/truststore.jks" />
        <parameter name="truststoreType" value="JKS" />
        <parameter name="truststorePassword" value="testtest" />
        <parameter name="crlUrl" value="file:///path/to/crl.crl" />
        <parameter name="wantClientAuthentication" value="true" />
    </server>

    <defaultHost>
        <attach uriPattern=""
targetClass="org.restlet.example.tutorial.Part12" />
    </defaultHost>
</component>
```

There can be multiple `crlUrl` parameters. In addition, two other parameters can be set:

- `sslServerAlias`, which will use a particular alias from the key store.
- `disableCrl`, which should be set to "true" if CRLs are not to be used.

The `wantClientAuthentication` parameter is handled by this the SslContextFactory, but is often used in conjunction with it.

Example embedded within the program, using two connectors:

```
Component component = new Component();


Server server1 = component.getServers().add(Protocol.HTTPS,

  "host1.example.org", 8083);

Series<Parameter> param1 = server1.getContext().getParameters();


param1.add("sslContextFactory","org.restlet.ext.ssl.PkixSslContextFactory");

param1.add("keystorePath","/path/to/keystore1.p12");

param1.add("keystorePassword","...");

param1.add("keystoreType","PKCS12");

//...


Server server2 = component.getServers().add(Protocol.HTTPS,

  "host2.example.com", 8083);

Series<Parameter> param2 = server2.getContext().getParameters();


param2.add("sslContextFactory","org.restlet.ext.ssl.PkixSslContextFactory");

param2.add("keystorePath","/path/to/keystore2.p12");

//...
```

This example uses two certificates depending on which server connector (and thus which listening socket) is used.

JAAS extension

Introduction

This extension facilitate the integration between the Restlet security API introduced in version 2.0 and the JAAS standard for authentication and authorization.

A typical use case is the creation of a JAAS Subject including principals for the authenticated Restlet user and its granted roles. This is achieved by the JaasUtils#createSubject(ClientInfo) method.

In addition, a JaasVerifier support the verification of user credentials based on a JAAS pluggable authentication mechanism and more precisely based on JAAS login modules.

For additional details, please consult the Javadocs.

Authenticating with LDAP

This extension can be used for LDAP authentication, for example. Considering this JAAS configuration:

```
BasicJaasAuthenticationApplication {

  com.sun.security.auth.module.LdapLoginModule REQUIRED


    userProvider="ldap://ldap.example.net/"


    authIdentity="uid={USERNAME},ou=people,dc=example,dc=net"


};
```

Using a verifier configured like this:

```
JaasVerifier verifier = new JaasVerifier("BasicJaasAuthenticationApplication");

verifier.setConfiguration(jaasConfig);

verifier.setUserPrincipalClassName("com.sun.security.auth.UserPrincipal");

authenticator.setVerifier(verifier);
```

A successful JAAS login will add principals like these to the subject:

```
 [LdapLoginModule] added LdapPrincipal
"uid=bruno,ou=people,dc=example,dc=net"
```

to Subject

```
 [LdapLoginModule] added UserPrincipal "bruno" to Subject
```

Thus, the resulting principals in ClientInfo are:

```
LdapPrincipal with name: "uid=bruno,ou=people,dc=example,dc=net"

UserPrincipal with name: "bruno"
```

A new user is created based on the first UserPrincipal name: 'bruno' in this example.

JDBC extension

Introduction

This connector is a client to a JDBC database. It is based on the JDBC API developed by Sun Microsystems and shipped with the JDK. Database connections are optionally pooled using Apache Commons DBCP. In this case, a different connection pool is created for each unique combination of JDBC URI and connection properties.

Description

This connector supports the following protocol: JDBC.

The SQL request and other kinds of parameters (such as pooling) are passed to the client connector via an XML representation. Please refer to the JDBC client Javadocs for more details.

The Response provides the result of the SQL request as a RowSetRepresentation which is a kind of XML representation of the ResultSet instance wrapped either in a JdbcResult or in a WebRowSet instance. See the RowSetRepresentation for more details.

Here is the list of dependencies of this connector:

- Apache Commons DBCP
- Apache Commons Pool
- Apache Commons Logging

For additional details, please consult the Javadocs.

RDF extension

Introduction

This extension provides supports for the core standard of the Semantic Web: the Resource Description Framework (RDF). It support all major RDF serialization formats (n3, Turtle, NTriples and RDF/XML) and can work in either a DOM-like or SAX-like way, with a unified API!

Features

Support for the RDF parsing and generation, either in DOM-like mode or in SAX-like mode.

References

For additional details, please consult the Javadocs. SLF4J extension
==========

Introduction

This extension is an integration with SLF4J, providing a log facade for SLF4J for the Restlet engine, allowing bridges to alternate logging mechanisms such as Log4J or LogBack.

For additional details, please consult the Javadocs.

Editions matrix

Introduction

This table presents the list of extensions provided by the various editions of the Restlet Framework.

The core module including both the Restlet API and the Restlet Engine is shipped by all editions.

Extensions availability

Extensions|Android|GAE|GWT|JEE|JSE|OSGi|Description --------- | ---------- | ---------- | ---------- | ---------- | ---------- | - org.restlet.ext.apispark||| ||||Injection of web API contract into the APISpark platform. org.restlet.ext.atom||| ||||Support for the Atom syndication and the AtomPub (Atom Publication Protocol) standards in their 1.0 version. org.restlet.ext.crypto||| ||||Support for cryptography. org.restlet.ext.e4| || ||||Support for the WADL specification. org.restlet.ext.emf| || ||||Integration with Eclipse Modeling Framework. org.restlet.ext.fileupload| || ||||Integration with Apache FileUpload. org.restlet.ext.freemarker| || ||||Integration with FreeMarker. org.restlet.ext.gae| || | | |Integration to the Google App Engine UserService for the GAE edition. org.restlet.ext.gson||| ||||Support for GSON representations. org.restlet.ext.guice| | | ||||Integration with Google Guice. org.restlet.ext.gwt| || ||||Server-side integration with GWT. org.restlet.ext.html||| ||||Support for the HTML (HyperText Markup Language) standard in its 4.0 version and above. org.restlet.ext.httpclient|| | ||||Integration with Apache Commons HTTP Client. org.restlet.ext.jaas||| ||||Support for JAAS based security. org.restlet.ext.jackson||| ||||Integration with Jackson. org.restlet.ext.javamail| | | ||||Integration with JavaMail. org.restlet.ext.jaxb| || ||||Integration with Java XML Binding. org.restlet.ext.jaxrs| || ||||Implementation of JAX-RS (JSR-311) org.restlet.ext.jdbc| | | ||||Integration with Java DataBase Connectivity (JDBC). org.restlet.ext.jetty| | | |||Integration with Jetty. org.restlet.ext.jibx| || ||||Integration with JiBX.

org.restlet.ext.json|||||||Support for JSON representations. org.restlet.ext.jsslutils|| | ||||Utilities to provide additional SSL support. org.restlet.ext.lucene| || ||||Integration with Apache Lucene, Solr and Tika sub-projects. org.restlet.ext.nio|| | ||||Integration with java.nio package. org.restlet.ext.oauth| | | ||||Support for OAuth HTTP authentication. org.restlet.ext.odata||| ||||Integration with OData services. org.restlet.ext.openid| || ||||Support for OpenID authentication. org.restlet.ext.osgi| | | | ||Support for the OSGi specification. org.restlet.ext.rdf||| ||||Support for the RDF parsing and generation. org.restlet.ext.rome| || ||||Support for syndicated representations via the ROME library. org.restlet.ext.sdc| | | ||||Integration with Google Secure Data Connector on the cloud side. org.restlet.ext.servlet| || || ||Integration with Servlet API. org.restlet.ext.simple| | | | |||Integration with Simple framework. org.restlet.ext.sip|| | ||||Support for Session Initiation Protocol (SIP). org.restlet.ext.slf4j| | | ||||Support for the SLF4J logging bridge. org.restlet.ext.spring| || ||||Integration with Spring Framework. org.restlet.ext.swagger| || ||||Integration with Simple framework. org.restlet.ext.thymeleaf||| ||||Integration with Thymeleaf. org.restlet.ext.velocity| || ||||Integration with Apache Velocity. org.restlet.ext.wadl| || ||||Support for the WADL specification. org.restlet.ext.xdb| | | || ||Integration within OracleJVM via the Oracle XML DB feature. org.restlet.ext.xml|||||||Support for the XML documents. org.restlet.ext.xstream| || ||||Integration with XStream.HTML ====

Introduction

This extension aims at providing a comprehensive support for HTML in the Restlet Framework. The most common requirement is the manipulation of HTML forms

Description

It is available in the **org.restlet.ext.html** package since version 2.1 (post-M5). Currently, it contains a replacement for the usual org.restlet.data.Form class that will be deprecated and adds support for multipart forms using the same Java API.

The FormDataSet class is a proper Restlet representation (which wasn't the case of Form) and is capable of parsing HTML form data in URL encoding and to write in either URL encoding or multipart form data based on the "multipart" boolean property.

For additional information on both encoding, please see the HTML 4.0 specification, section 17.13.

Usage example

Representation file = new FileRepresentation(***);

```
FormDataSet form = new FormDataSet();

form.setMultipart(true);

form.getEntries().add(new FormData("number", "5555555555"));

form.getEntries().add(new FormData("clip", "rickroll"));

form.getEntries().add(new FormData("upload_file", file));

form.getEntries().add(new FormData("tos", "agree"));


ClientResource cr = new ClientResource("http://mydomain.com/upload");

cr.post(form);
```

Jackson extension

This extension provides and integration of Restlet with Jackson. Jackson is a fast library to serialize objects to JSON and back again.

Usage instructions

The extension comes with a JacksonRepresentation that can either: - wrap a Java object to serialize it as JSON, JSON Binary (Smile), XML, CSV or YAML representation, - wrap a representation to parse it back as a Java object.

It also provides a plugin for the ConverterService which will automatically serialize and deserialize your Java objets returned by annotated methods in ServerResource subclasses.

Here is an example server resource:

```
import org.restlet.Server;

import org.restlet.data.Protocol;

import org.restlet.resource.Get;

import org.restlet.resource.Put;

import org.restlet.resource.ServerResource;


public class TestServer extends ServerResource {


    private static volatile Customer myCustomer = Customer.createSample();
```

```
    public static void main(String[] args) throws Exception {
        new Server(Protocol.HTTP, 8182, TestServer.class).start();
    }

    @Get
    public Customer retrieve() {
        return myCustomer;
    }

    @Put
    public void store(Customer customer) {
        myCustomer = customer;
    }

}
```

Here is the matching client resource:

```
import org.restlet.resource.ClientResource;
import org.restlet.resource.ResourceException;

public class TestClient {

    /**
     * @param args
     * @throws ResourceException
     */
    public static void main(String[] args) throws Exception {
```

```
        ClientResource cr = new ClientResource("http://localhost:8182");

        // Retrieve a representation
        Customer customer = cr.get(Customer.class);
        System.out.println(customer);

        // Update the target resource
        customer.setFirstName("John");
        customer.setLastName("Doe");
        cr.put(customer);

        // Retrieve the updated version
        customer = cr.get(Customer.class);
        System.out.println(customer);
    }

}
```

Note that our Customer and Address classes are just regular serializablecbeans, with no special parent classes and no special annotations.

What is nice is that the automatically generated representations can be customized via Jackson annotations on the serialized beans. More details on annotations are available in Jackson documentation.

For additional details, please consult the Javadocs. Crypto extension
==========

Introduction

This extension includes the HTTP_DIGEST, Amazon S3 and Windows Azure client authentication.

For additional details, please consult the Javadocs.

Lucene extension

Introduction

The goal of this extension is to provide an integration between Restlet and Apache Lucene project and subprojects such as Solr and Tika.

Its main feature is to allow an application to have access to Lucene indexing capabilities in a RESTful way thanks to Solr. The other feature is to provide a TikaRepresentation leveraging Lucene Tika subproject to extract metadata from any Representation.

For additional details, please consult the Javadocs.

Solr integration

The connector contributed allows you to interact with Solr with "solr://" references the same way you would do it through HTTP :

http://wiki.apache.org/solr/SolrRequestHandler

Initialization

The CoreContainer that will be helped by the SolrClientHelper can be initialized in two different ways.

First one is to specify directory and configFile parameters.

```
Client solrClient =
component.getClients().add(SolrClientHelper.SOLR_PROTOCOL);

solrClient.getContext().getParameters().getFirstValue("directory");

solrClient.getContext().getParameters().getFirstValue("configFile");
```

Second one is to create the CoreContainer.

```
Client solrClient =
component.getClients().add(SolrClientHelper.SOLR_PROTOCOL);

CoreContainer coreContainer = initSolrContainer(component);

solrClient.getContext().getAttributes().put("CoreContainer",coreContainer);
```

To configure your core container see solr documentation :
http://wiki.apache.org/solr/

Usage

It is possible to interact with the core container using "solr://" Restlet requests with the client dispatcher the same way it is described on Solr's wiki using http requests.

SolrClientHelper returns instances of SolrRepresentation from which you can get results encoded as JSON or XML.

To update a document like described here you can do :

```
StringRepresentation repr = new StringRepresentation(xml,
MediaType.TEXT_XML);

getContext().getClientDispatcher().post("solr://main/update", repr);
```

Solr Core Container

If you would like to interact with Solr through http without a Servlet container you can use this restlet. It can also be very useful for debug.

```
import org.restlet.Context;

import org.restlet.Restlet;

import org.restlet.data.Reference;

import org.restlet.data.Request;

import org.restlet.data.Response;


public class SolrForward extends Restlet {


    public SolrForward() {

    }


    public SolrForward(Context context) {

        super(context);

    }


    @Override

    public void handle(Request request, Response response) {

        super.handle(request, response);

        String path = request.getResourceRef().getRemainingPart();

        Reference solrRef = new Reference("solr://"+path);

        Request solrRequest = new
Request(request.getMethod(),solrRef,request.getEntity());
```

```
        Response solrResp =
getContext().getClientDispatcher().handle(solrRequest);

        response.setStatus(solrResp.getStatus());

        response.setEntity(solrResp.getEntity());

    }


}
```

Guice extension

Introduction

To be done.

Javadocs.

OpenID extension

Support for OpenID 2.0 HTTP authentication. Leverages the OpenID4Java 0.9 library.

For additional details, please consult the Javadocs. Part IV - Restlet Extensions ==========

Introduction

The Restlet Framework is composed of two parts:

- The Restlet Core (Restlet API + Restlet Engine implementing the API)
- Optional Restlet extensions

Description

The Restlet Framework lets you build complete Web applications by itself and is capable of leveraging other open source projects to facilitate tasks such as generation of dynamic representations, management of uploaded files, sending of mails, etc. There are two types of extensions.

The first one is for extensions built only on top of the Restlet API and typically provide new type of representations (JAXB, FreeMarker, JiBx, etc.) or support of important standards (Atom, WADL).

The second one covers extensions to the Restlet engine such as client connectors, server connectors and authentication helpers.

Distribution

All extensions and their dependencies are shipped with the Restlet distribution by the way of JAR files. Adding an extension to your application is as simple as adding the archives of the chosen extension and its dependencies to the classpath.

SIP extension

Introduction

This extension provides support for the Session Initiation Protocol, largely used for voice over IP (VoIP). It ships both client and server SIP connectors over TCP, reusing the 90% of the logic used by the new NIO/HTTP internal connector, providing excellent scalability and performance.

For now, you can get more details on the Restlet/SIP specifications page.

Features

- SIP transport over TCP
- All SIP headers, including "Via", map to a Java API
- All base SIP methods supported
- Extension INFO, REGISTER, SUBSCRIBE, NOTIFY, PUBLISH and REFER methods supported
- OPTIONS method supported : useful for integration purposes as most gateways require it for redundancy

Mapping SIP headers

The complete list of SIP headers is available at the IANA site.

SIP header

Accept

Accept-Encoding

Accept-Language

Alert-Info

Allow

Allow-Events

Authentication-Info

Authorization

Call-ID

Call-Info

Contact

Content-Disposition

Content-Encoding

Content-Language

Content-Length

Content-Type

CSeq

Date

Error-Info

Event

Expires

From

In-Reply-To

Max-Forwards

Min-Expires

MIME-Version

Organization

Priority

Proxy-Authenticate

Proxy-Authorization

Proxy-Require

RAck

Record-Route

Refer-To

Reply-To

Require

Retry-After

Route

Rseq

Server

SIP-ETag

SIP-If-Match

Subject

Subscription-State

Supported

Timestamp

To

Unsupported

User-Agent

Via

Warning

WWW-Authenticate

JAXB extension

This extension provided an integration with JAXB. JAXB is a convenient way to process XML content using Java objects by binding XML schemas to Java classes.

The extension is composed of just one class, the JaxbRepresentation that extends the OutputRepresentation and is able to both serialize and deserialize a Java objects graph to/from an XML document.

For additional details, please consult the Javadocs.

JavaMail extension

Introduction
This connector is based on JavaMail that provides a platform-independent and protocol-independent framework to build mail and messaging applications.

Description
This connector supports the following protocols: SMTP, SMTPS.

The list of supported specific parameters is available in the Javadocs:

- JavaMail client Javadocs

The mail and its properties (sender, recipient, subject, content, etc) have to be specified as an XML representation.

For additional details, please consult the Javadocs.

Apache HTTP Client extension

Introduction
This connector is based on Apache Commons HTTP client. It provides an HTTP and HTTPS client connector with advanced multi-threading and connection reuse support.

Description
As pointed out by the Apache HTTPClient tutorial it is crucial to read entirely each response. It allows to release the underlying connection. Not doing so may cause future requests to block.

This connector supports the following protocols: HTTP, HTTPS. The list of supported specific parameters is available in the Javadocs:

- HTTP client parameters

For additional details, please consult the Javadocs.

Oracle XDB Restlet Adapter

Documentation
- Architecture - Introduction
- Installing
- Testing
- FAQ
- Future plans - Known caveats

- XdbRepresentation class

For additional details, please consult the Javadocs.

Using XDB Restlet Adapter within Maven projects

- Introduction, download, compiling and installing
- Running and testing
- Implemented services
- Todo List

XML extension

Introduction

This extension provides support for XML and XSLT representations.

For additional details, please consult the Javadocs.

Atom extension

Introduction

This extension provides support for the Atom standard. Atom is an evolution of the RSS standard that was introduced to remove know limitations and enable new features and usages, especially in the light of the REST principles.

Its main purpose it to define a format to represent Web feeds such as news feed. This is called the Atom Syndication Format.

The second purpose is to support the publication and modification of those Web feeds, remotely on the Web and in an interoperable manner. This is called the Atom Publishing Protocol (APP).

Description

The Restlet extension for Atom provides a complete Atom API for both Web feeds and publication documents. This API is capable of both parsing and formatting Atom and APP XML documents compliant with the 1.0 specifications.

The two main classes you should use are Feed (an Atom XML feed) and Service (an AtomPub XML service descriptor) which are both subclasses of SaxRepresentation. As such instances of both classes can be directly returned as representation of your Restlet resources. They also support parsing via the constructors accepting a Representation parameter.

For an usage example, check the source code of the org.restlet.example.book.restlet.ch8.resources.FeedResource class.

For additional details, please consult the Javadocs.

Links

- AtomEnabled.org
- Wikipedia entry

Net extension

Introduction

This connector is fully based on the NIO package.

Description

For additional details, please consult the Javadocs.

JAX-RS extension

Introduction

This Restlet Extension implements the Java Specification JAX-RS: Java API for RESTful Web Services. Note that this implementation is not final yet.

Description

To run this example, you need the Restlet libraries. Download a 2.3 version from restlet.org/downloads/. (For a general Restlet example take a look at the first steps examples).

Now create a new Java Project, and add the following jars (resp. projects) to the classpath (right click on project, Properties, Java Build Path, Libraries (resp.Projects), Add):

- org.restlet (the core Restlet API)
- org.restlet.ext.jaxrs (the JAX-RS Runtime)
- javax.ws.rs (the JAX-RS API and also the specification)

Depending of your needs you have to add the following:

- if you want to use the provider for javax.xml.transform.DataSource: add javax.activation and javax.mail
- if you want to use the provider for JAXB: add javax.xml.bind and javax.xml.stream
- if you want to use the provider for JSON: add org.json

Click "Ok" twice. Now you are ready to start. - First we will create an example root resource class and then show how to get it running by the Restlet JAX-RS extension.

For additional details, please consult the Javadocs.

Create JAX-RS example

Create a new package, e.g. test.restlet.jaxrs

Create a root resource class

First create an easy root resource class: Create a new java class named **EasyRootResource** in the previously created package and insert the following source code:

```
package test.restlet.jaxrs;


import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;


@Path("easy")

public class EasyRootResource {


    @GET

    @Produces("text/html")

    public String getHtml() {

        return "<html><head></head><body>\n"

                + "This is an easy resource (as html text).\n"

                + "</body></html>";

    }


    @GET

    @Produces("text/plain")

    public String getPlain() {

        return "This is an easy resource (as plain text)";

    }

}
```

Create Application

To provide a collection of root resource classes (and others) for a JAX-RS runtime you integrate these classes to an Application. Create a new class **ExampleApplication** in the same package with the following content:

```
package test.restlet.jaxrs;


import java.util.HashSet;

import java.util.Set;

import javax.ws.rs.core.Application;


public class ExampleApplication extends Application {


    public Set<Class<?>> getClasses() {

        Set<Class<?>> rrcs = new HashSet<Class<?>>();

        rrcs.add(EasyRootResource.class);

        return rrcs;

    }

}
```

The root resource class and the Application is specified by the JAX-RS specification. It can be used in any JAX-RS runtime environment.

Now create a runtime environment instance and pass the Application instance to it. This is runtime environment specfic. Below you see this for the Restlet JAX-RS environment:

Set up a JAX-RS server

A JAX-RS server using the Restlet JAX-RS extension is set up like any Restlet server. Create a third class in the same package, named **ExampleServer**:

```
package test.restlet.jaxrs;


import org.restlet.Component;

import org.restlet.Server;
```

```
import org.restlet.data.Protocol;
import org.restlet.ext.jaxrs.JaxRsApplication;

public class ExampleServer {

    public static void main(String[] args) throws Exception {
        // create Component (as ever for Restlet)
        Component comp = new Component();
        Server server = comp.getServers().add(Protocol.HTTP, 8182);

        // create JAX-RS runtime environment
        JaxRsApplication application = new
JaxRsApplication(comp.getContext());

        // attach Application
        application.add(new ExampleApplication());

        // Attach the application to the component and start it
        comp.getDefaultHost().attach(application);
        comp.start();

        System.out.println("Server started on port " + server.getPort());
        System.out.println("Press key to stop server");
        System.in.read();
        System.out.println("Stopping server");
        comp.stop();
        System.out.println("Server stopped");
    }
```

```
}
```

Start this class, open a browser and request http://localhost:8182/easy. Now you see the HTML representation. If you request the same URI with accepted media type "text/plain", you get a plain text representation.

This example (a little bit extended) is available in the project org.restlet.example. See package org.restlet.test.jaxrs. There is another root resource class with a reacheable resource class and also an example with user authentication.

A lot of more resource classes are available in the test project (org.restlet.test, packages starting with org.restlet.test.jaxrs). They are implemented for testing, and most of them do not do intelligent things ... :-) But they show the actual status of development of this JAX-RS runtime environment.

This runtime environment is still under development, and I'm very busy continuing it ...

Run in a Servlet Container

If you want to run the JAX-RS Application in a Servlet Container, create a subclass of the JaxRsApplication. In the constructor you could attach the Application and sets the Guard and the RoleChecker (if needed).

```
public class MyJaxRsApplication extends JaxRsApplication {


   public MyJaxRsApplication(Context context) {
      super(context);
      this.add(new ExampleApplication());
      this.setGuard(...); // if needed
      this.setRoleChecker(...); // if needed
   }
}
```

For details to run this Application in a Servet Container take a look at Restlet FAQ.

You could use this subclass also in the example above:

```
      // create JAX-RS runtime environment
      Application application = new MyJaxRsApplication(comp.getContext());
```

```
    // if you use this kind, you don't need to attach the Application again.
```

Comments are welcome to the Restlet mailing list or directly to Stephan.Koops<AT>web.de !

This extension is the result of a (german) master thesis.

OAuth extension

This extensions is a **preview support** of the OAuth v2.0 standard. It has been developed based on an initial contribution by Ericsson Labs (see original project here) and later enhanced to support the final 2.0 RFC thanks to another community contribution.

This is intended to be used with primarily following use-cases in mind: - Create a protected resource - Create a authenticating server - Create a web client accessing protected resources

It is very simple to create an OAuth server with just a few lines of code. It is also possible to implement a custom back end for data storage and retrieval. The default implementation stores only to memory, so a JVM restart flushes all data.

```
{
   public Restlet createInboundRoot(){
   ...
   OAuthAuthorizer auth = new OAuthAuthorizer(
      "http://localhost:8080/OAuth2Provider/validate");
   auth.setNext(ProtectedResource.class);
   router.attach("/me", auth);
   ...
}
```

Example 1. Creating a Protected Resource

```
{
   OAuthParameter params = new OAuthParameters("clientId", "clientSecret",
      oauthURL, "scope1 scope2");
   OAuthProxy proxy = new OauthProxy(params, getContext(), true);
   proxy.setNext(DummyResource.class);
```

```
    router.attach("/write", write);


    //A Slightly more advanced example that also sets some SSL client
parameters
    Client client = new Client(Protocol.HTTPS);
    Context c = new Context();
    client.setContext(c);
    c.getParameters().add("truststorePath", "pathToKeyStoreFile");
    c.getParameters(0.add("truststorePassword", "password");
    OAuthParameter params = new OAuthParameters("clientId", "clientSecret",
       oauthURL, "scope1 scope2");
    OAuthProxy proxy = new OauthProxy(params, getContext(), true, client);
    proxy.setNext(DummyResource.class);
    router.attach("/write", write);
}
```

Example 2. Creating a Proxies to access protected resources

For additional details, please consult the Javadocs.

SDC extension

Introduction

Description

Apache FileUpload extension

Introduction

This extension leverages the Apache FileUpload library to provide a robust, high-performance, Web-based file upload in Restlet server-side applications.

Description

This extension lets you receive files sent by POST or PUT requests and to parse the posted entity (which is actually an instance of the "Representation" class) and to extract a list of FileItems from it, each item corresponding to one file uploaded in the posted request, typically from a Web form.

For additional details, please consult the Javadocs.

Here is the list of dependencies for this extension:

- Java Servlet
- Apache Commons FileUpload

Usage example

This sample code illustrates how to upload files with the FileUpload extension. It is composed of 3 classes:

- a resource "MyResource" which responds to GET and POST requests,
- an application called "MyApplication" which routes all received requests to the resource,
- a component called "TestFileUpload" which creates a local HTTP server on port 8182 and contains only one application (one instance of "MyApplication") attached to the path "/testFileUpload".

Thus, each request to the following uri "http://localhost/testFileUpload" will be handled by a new instance of "MyResource".

The single representation of this resource is a web form with a file select control and a submit button. It allows to set up a request with an uploaded file that will be posted to the resource. The name of the file select control ("fileToUpload") is referenced by the resource.

Every Resource instance handles the POST request in method "accept" which accepts the posted entity as single parameter. The aim of the MyResource instance is to parse the request, get the file, save it on disk and send back its content as plain text to the client.

Here is the content of the MyResource#accept method:

```
@Post
public Representation accept(Representation entity) throws Exception {
    Representation rep = null;
    if (entity != null) {
        if (MediaType.MULTIPART_FORM_DATA.equals(entity.getMediaType(),
                true)) {
            String fileName = "c:\\temp\\file.txt";
```

```
// The Apache FileUpload project parses HTTP requests which
// conform to RFC 1867, "Form-based File Upload in HTML". That
// is, if an HTTP request is submitted using the POST method,
// and with a content type of "multipart/form-data", then
// FileUpload can parse that request, and get all uploaded files
// as FileItem.

// 1/ Create a factory for disk-based file items
DiskFileItemFactory factory = new DiskFileItemFactory();
factory.setSizeThreshold(1000240);

// 2/ Create a new file upload handler based on the Restlet
// FileUpload extension that will parse Restlet requests and
// generates FileItems.
RestletFileUpload upload = new RestletFileUpload(factory);
List<FileItem> items;

// 3/ Request is parsed by the handler which generates a
// list of FileItems
items = upload.parseRequest(getRequest());

// Process only the uploaded item called "fileToUpload" and
// save it on disk
boolean found = false;
for (final Iterator<FileItem> it = items.iterator(); it
        .hasNext()
        && !found;) {
    FileItem fi = it.next();
```

```
                if (fi.getFieldName().equals("fileToUpload")) {
                    found = true;
                    File file = new File(fileName);
                    fi.write(file);
                }
            }

            // Once handled, the content of the uploaded file is sent
            // back to the client.
            if (found) {
                // Create a new representation based on disk file.
                // The content is arbitrarily sent as plain text.
                rep = new FileRepresentation(new File(fileName),
                        MediaType.TEXT_PLAIN, 0);
            } else {
                // Some problem occurs, sent back a simple line of text.
                rep = new StringRepresentation("no file uploaded",
                        MediaType.TEXT_PLAIN);
            }
        }
    } else {
        // POST request with no entity.
        setStatus(Status.CLIENT_ERROR_BAD_REQUEST);
    }

    return rep;
}
```

Before running this example, please add the following jars to the classpath:

- org.restlet (Restlet API)
- org.restlet.ext.fileupload (Restlet extension based on the Apache FileUpload project)
- org.apache.commons.fileupload (Apache FileUpload project)
- org.apache.commons.io (Apache FileUpload project)
- javax.servlet.jar (Servlet archive used by the FileUpload library)

Links

- Apache FileUpload library
- Server connectors
- Sample code of FileUpload extension (zip file)

Spring extension - Integration modes

Introduction

During the development of the 1.0 version of the Restlet API, several users attempted to integrate Restlet with Spring. They were especially trying to use the XML-based bean wiring feature of Spring. This resulted in several examples available on the community Wiki.

In order to facilitate this integration, two dedicated Spring extension were added to the Restlet. It allows us to provide several integration modes.

Restlet as main container

In the first mode, the goal is to leverage the concept of Restlet Application and all the associated services, as well as the transparent deployment to either a Servlet container (using the adapter ServerServlet extension class) or using a standalone HTTP server connector. For this, you can leverage the SpringContext class which is a Spring's GenericApplicationContext subclass. You can associate a list of XML or property configuration URIs (file:/// or war:/// URIs) in order to have Spring auto-instantiate and wire your Restlet beans.

Spring as main container

In the second mode, the goal is to leverage the concept of Spring Web Application as an alternative to the Restlet Application. This is sometimes required when the Restlet code is part of a larger Spring-based Web application, with dependencies on the Servlet API for example.

Initially, it was hard to achieve this integration because the Servlet extension, and especially the ServerServlet adapter class was assuming the usage of a Restlet Application. Later we added a lighter

adapter based on the ServletConverter class that lets you directly instantiate Restlet Routers, Finders and ServerResources from your existing Servlet-based Spring code. You can check the Javadocs for details.

Finally, there is also a SpringFinder class available in the Spring extension. It hasn't any specific dependency to Spring, but the addition of a parameter-less create() method allows the usage of the Spring's "lookup-method" mechanism.

In Restlet 1.1, the Spring extensions received several contributions, increasing the number of ways to integrate Restlet with Spring. There is now a RestletFrameworkServlet, a SpringServerServlet, SpringBeanFinder and SpringBeanRouter.

Spring extension - A complete example

Introduction

This example is a Spring-enabled but otherwise functionally equivalent version of the bookmarks example from chapter 7 of RESTful Web Services by Richardson and Ruby. The complete code for this version is available through CVS from :pserver:anonymous@cvs.cs.luc.edu:/root/laufer/433, module BookmarksRestletSpring. Project dependencies are managed using Apache Maven, and the example illustrates standalone and servlet-container deployment.

In a nutshell, Spring handles the configuration of the top-level Restlet Component and Router beans. The Restlet Resources had to be modified to support the init method and the injection of the dependency on the db4o ObjectContainer, which is also configured in Spring. As expected, the domain objects User and Bookmark remained unchanged.

Description

First, we show the configuration of the Restlet Component and top-level Router beans. The top-level Router is necessary only if an non-root context path is required for standalone deployment.

```
<bean id="top" class="org.restlet.ext.spring.SpringComponent">

   <property name="server">

      <bean class="org.restlet.ext.spring.SpringServer">

         <constructor-arg value="http" />

         <constructor-arg value="3000" />

      </bean>
```

```
    </property>
    <property name="defaultTarget" ref="default" />
</bean>


<bean id="default" class="org.restlet.ext.spring.SpringRouter">
    <property name="attachments">
        <map>
            <entry key="/v1" value-ref="root" />
        </map>
    </property>
</bean>
```

As a result, the main method has become very simple. It loads a Spring context based on two configuration metadata files, one for the preceding top-level beans, and one for the application-specific beans shown below. It then starts up the top-level Restlet Component.

```
    public static void main(String... args) throws Exception {
    // load the Spring application context
    ApplicationContext springContext = new ClassPathXmlApplicationContext(
        new String[] { "applicationContext-router.xml",
"applicationContext-server.xml" });


    // obtain the Restlet component from the Spring context and start it
    ((Component) springContext.getBean("top")).start();
    }
```

Next, we look at the configuration of the application-specific Router. We use a SpringRouter for this purpose, which is configured using a map of URI patterns to resources. The SpringFinder beans provide the extra level of indirection required to create Resource instances lazily on a per-request basis.

In this example, the last URI pattern has to be customized to accept complete URIs (possibly including slashes) as the last component of the pattern. We use Spring's nested properties to drill into the

configuration of the URI pattern along with Spring's mechanism for accessing a static field in a class.

```
<bean id="root" class="org.restlet.ext.spring.SpringRouter">
   <property name="attachments">
      <map>
         <entry key="/users/{username}">
            <bean class="org.restlet.ext.spring.SpringFinder">
               <lookup-method name="create"
                  bean="userResource" />
            </bean>
         </entry>
         <entry key="/users/{username}/bookmarks">
            <bean class="org.restlet.ext.spring.SpringFinder">
               <lookup-method name="create"
                  bean="bookmarksResource" />
            </bean>
         </entry>
         <entry key="/users/{username}/bookmarks/{URI}">
            <bean class="org.restlet.ext.spring.SpringFinder">
               <lookup-method name="create"
                  bean="bookmarkResource" />
            </bean>
         </entry>
      </map>
   </property>
   <property name="routes[2].template.variables[URI]">
      <bean class="org.restlet.util.Variable">
         <constructor-arg ref="org.restlet.util.Variable.TYPE_URI_ALL" />
      </bean>
```

```
    </property>
</bean>

<bean id="org.restlet.util.Variable.TYPE_URI_ALL"

class="org.springframework.beans.factory.config.FieldRetrievingFactoryBean
" />
```

Unlike the preceding singleton beans, we define the ServerResources as prototype beans so that they get instantiated separately for each request. All of the Resource beans depend on the db4o ObjectContainer and are configured analogously, so we show only UserResource here.

```
<bean id="userResource"
    class="org.restlet.example.book.rest.ch7.spring.UserResource"
    scope="prototype">
    <property name="container" ref="db4oContainer" />
</bean>
```

Using the db4o Spring Module, configuring the ObjectContainer is straightforward.

```
<bean id="db4oContainer"
    class="org.springmodules.db4o.ObjectContainerFactoryBean">
    <property name="configuration" ref="db4oConfiguration" />
    <property name="databaseFile" value="file://${user.home}/restbook.dbo"
/>
</bean>

<bean id="db4oConfiguration"
    class="org.springmodules.db4o.ConfigurationFactoryBean">
    <property name="updateDepth" value="2" />
    <property name="configurationCreationMode" value="NEW" />
</bean>
```

As mentioned above, we added the following elements to each application-specific Resource:

- An empty default constructor.
- An init method containing the code originally in the non-default constructor. That constructor now simply invokes the init method, although it is no longer used in this context.
- An instance variable and getter/setter pair for the db4o ObjectContainer.

The following code fragment summarizes these changes.

```
public class UserResource extends ServerResource {


    private ObjectContainer container;


    // other instance variables


    public UserResource() { }


    @Override
    public void init(Context context, Request request, Response response) {
        super.init(context, request, response);
        // code originally in non-default constructor
    }


    public UserResource(Context context, Request request, Response
response) {
        super(context, request, response);
        init(context, request, response);
    }


    public ObjectContainer getContainer() {
```

```
        return container;
    }

    public void setContainer(ObjectContainer container) {
        this.container = container;
    }

    // other methods
}
```

Spring extension - Configuring Restlet beans

Passing the parent context

One frequent issue that developers encounter when configuring their Restlet beans with Spring XML is that it is not easy to find a way to pass the Context instance to the Restlet subclasses such as Application, Directory or Router. What we actually need to do is to extract the context property from the parent Restlet (typically a Component or an Application) and pass it by reference to the constructor method.

Spring provides two mechanism to achieve this: either using the PropertyPathFactoryBean class to create a context bean such as:

```
<!-- Restlet Component bean -->
<bean id="component" class="org.restlet.ext.spring.SpringComponent">
    ...
</bean>

<!-- Component's Context bean -->
<bean id="component.context"
class="org.springframework.beans.factory.config.PropertyPathFactoryBean"/
>

<!-- Application bean -->
<bean id="application" class="org.restlet.Application">
```

```
    <constructor-arg ref="component.context" />
    ...
</bean>
```

The second mechanism is based on the Spring utilities schema and is actually more compact:

```
<!-- Restlet Component bean -->
<bean id="component" class="org.restlet.ext.spring.SpringComponent">
    ...
</bean>


<!-- Application bean -->
<bean id="application" class="org.restlet.Application">
    <constructor-arg>
        <util:property-path path="component.context" />
    </constructor-arg>
    ...
</bean>
```

You also have to make sure that the util namespace is properly declared in your XML configuration header. Here is a snippet for Spring 2.5:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.5.xsd">
```

```
<!-- Add you <bean/> definitions here -->


</beans>
```

This utilities mechanism is quite powerful and flexible, for more information check this page.

Spring extension

Table of contents

Introduction

This extension provides various modes of integration between the Restlet Framework and the popular Spring Framework. Historically, this extension emerged from the needs of Spring users, stuck between the Spring's mechanism of Dependency Injection mostly based on JavaBean setters (Setter Injection) and constructor arguments (Constructor Injection), and the conceptual choices of the Restlet Framework that didn't systematize the use of simple POJOs.

For additional details, please consult the Javadocs.

Spring extension - Configuration of Restlet resources

Configuration of basic properties

Restlet resources support only limited configuration beyond injecting custom dependencies such as the ObjectContainer in the example above. To make specific ServerResource classes more reusable, it would be helpful if their basic properties could be configured through Spring:

- available
- modifiable
- negotiateContent
- readable

Currently, the init method resets these properties to their default values but, in the Spring component life cycle, is invoked after Spring sets the properties. An obvious workaround is to refine the init method like so:

```
@Override
public void init(Context context, Request request, Response response) {
    final ResourcePropertyHolder backup = new ResourcePropertyHolder();
    BeanUtils.copyProperties(this, backup);
    super.init(context, request, response);
    BeanUtils.copyProperties(backup, this);
}
```

Configuration of representation templates

In addition, it would be quite useful if one could map media types to representation templates in Spring. In the following example, we explore this idea further by mapping different media types to different Freemarker and JSON representation factories. Whenever a Resource creates a concrete representation, it passes a uniform data model to the representation factory, which then instantiates the template with the data model and returns the resulting representation. (The Freemarker configuration is also handled by Spring.)

```
<bean id="resource" class="helloworldrestlet.HelloWorldResource"
    scope="prototype">
    <property name="available" value="true" />
    <property name="representationTemplates">
        <map>
            <entry key-ref="org.restlet.data.MediaType.TEXT_PLAIN"
                value-ref="hwFreemarkerTextPlain" />
            <entry key-ref="org.restlet.data.MediaType.TEXT_HTML"
                value-ref="hwFreemarkerTextHtml" />
            <entry key-ref="org.restlet.data.MediaType.APPLICATION_JSON"
                value-ref="jsonRepresentationFactory" />
        </map>
    </property>
```

```
</bean>

<bean id="hwFreemarkerTextPlain"
   class="edu.luc.etl.restlet.spring.FreemarkerRepresentationFactory">
   <property name="templateName" value="hw-plain.ftl" />
   <property name="freemarkerConfig" ref="freemarkerConfig" />
</bean>

<bean id="hwFreemarkerTextHtml"
   class="edu.luc.etl.restlet.spring.FreemarkerRepresentationFactory">
   <property name="templateName" value="hw-html.ftl" />
   <property name="freemarkerConfig" ref="freemarkerConfig" />
</bean>

<bean id="jsonRepresentationFactory"
   class="edu.luc.etl.restlet.spring.JsonRepresentationFactory" />

<!-- omitted beans for specific MediaType static fields -->

<bean id="freemarkerConfig"
   class="freemarker.template.Configuration">
   <property name="directoryForTemplateLoading"
      value="src/test/resources/presentation" />
   <property name="objectWrapper">
      <bean class="freemarker.template.DefaultObjectWrapper" />
   </property>
</bean>
```

When using this approach, the ServerResources themselves become very simple, for example:

```
public class HelloWorldResource extends ConfigurableRestletResource {

   @Override

   public Representation get(Variant variant) {

   Map<String, Object> dataModel = Collections.singletonMap("DATE",
(Object) new Date());

   return createTemplateRepresentation(variant.getMediaType(), dataModel);

   }

}
```

A working proof-of-concept for this approach is available through Subversion at http://luc-pervasive.googlecode.com/svn/trunk/webservices/ConfigurableRestletResource. Support for the missing configuration of representations tied to responses to non-GET requests is in the works.

GWT extension

This extension is a server-side integration with GWT 2.5.

It especially supports an object serialization format specific to GWT and facilitates the exchange of representation beans via Restlet annotated interfaces. This format is based on the GWT-RPC serialization mechanism, even though it is used in a regular REST/HTTP way thanks to Restlet Framework edition for GWT(client-side).

For additional details, please consult the Javadocs.

Thymeleaf extension

Introduction

This extension is an integration with Thymeleaf.

For additional details, please consult the Javadocs.