Introduction

A bit of history

The Web and Java have a long history in common already. Since its creation in 1994, Java includes the java.net package and in particular a HTTP client via the HttpURLConnection class. At that time, Applets were a popular technology running in Web browsers that needed a way to call back their origin Web server. Using a HTTP client was a good way to do this, limiting the security issues with firewalls.

Later, around 1998, the Servlet API was introduced as a way to generate dynamic content on HTTP servers. It basically tries to represent a HTTP request/response cycle in an object-oriented model. Along with Java Server Pages (JSP), its sister specification, it became part of a larger effort to bring Java technologies inside companies. In applications, developers adapt the common object-oriented MVC (Model-View-Controller) design pattern, also known as the "Model 2" approach. In this model, Servlets are acting as controllers, JSP pages as views and JavaBeans objects as models.

At the same period, the XML standard emerged from a W3C working group. Along with related XSLT, XSL-FO and XPath standards, it provided a new way to generate dynamic pages competing with Servlets and JSP pages. The reaction was to embrace XML all over the Java platform and JSP evolved to be able to generate XML documents. Another path was followed by Stefano Mazzocchi who started Apache Cocoon, an XML publishing framework built around the concepts of separation of concerns and component-oriented design.

In 2000, the Struts project defined standard controllers, called Actions, following the existing Model2/MVC pattern. Application state is exchanged between the model and the view using ActionForms. It quickly became successful as it provided a higher level of abstraction than pure Servlets, especially for forms handling. Later, numerous other frameworks appeared to address similar problems. The most notable is Spring which is a more comprehensive Java/J2EE application framework than Struts, also implementing the MVC approach.

The same year, Roy T. Fielding, who co-authored the HTTP & URI specifications and co-founded the Apache HTTP server, wrote a dissertation on software architectures. In the chapter 5 he formally defined the architecture style supporting the Web and called it REST, for Representational State Transfer. It defines a new paradigm that could be called resource-orientation, in comparison to object-orientation, where resources represent identifiable concepts of your domain (comparable to objects). The resources are referenced using the standard URIs (URLs or URNs) and manipulated by components (browsers, servers, proxies, gateways, etc.) through a

uniform interface. This interface has a limited list of verbs, essentially the HTTP methods. Also, resources are never exchanged directly, only representations of their state are. Connectors are the architecture elements that enable the communication of representations between components, implementing for example the client-side of the HTTP protocol.

Servlet limitations

At the end of 2003, Greg Wilkins, the author of the Jetty Web container and contributor to the Servlet specifications, blogged about some issues with Servlets:

- No clear separation between the protocol concerns and the application concerns.
- Unable to take full advantage of the non-blocking NIO due to blocking IO assumptions.
- Full Servlet Web containers are overkill for many applications like serving XML documents for Web services.

He proposed to specify a new API that would be truly protocol-independent and define contentlets to expose content and its metadata. These ideas were both thought-provoking and inspiring for the creation of the Restlet project. In a later post, Greg Wilkins explains with detailed arguments why the current Servlet API, without a concept like contentlets, limits the efficient usage of the non-blocking NIO API. This traditionally imposes the use of a separate thread for each HTTP requests to handle.

Another major limitation is due to the possibility for Servlets to store state in-memory, at the application or user session level. Even though it looks like a nice feature for Web application developers, it became a major issue for the scalability and high-availability of Servlet containers. To compensate, complex load-balancing, session replication and persistence mechanisms must be implemented. But in the end, scalability inevitably suffers.

Restlet inception

When I recently started the development of a Web site, I wanted it to comply with the REST architectural style, as much as technically possible. After many researches, I noted the lack of a REST framework in Java. The only project that came close to it was 1060 NetKernel developed by 1060 Research but it had too many features for my needs and I found that the support of REST concepts was not as direct as I was expecting.

This led me to develop my own REST framework on top of the Servlet API. This worked well up to a point were the Servlet API was completely hidden. I remembered about Greg Wilkins's propositions and decided to completely bypass the Servlet API. Fortunately, Jetty 5 has a nice separation between its HTTP protocol implementation and its support for the Servlet API. In the end, I was able to develop the first Restlet connector, a HTTP server connector, directly issuing REST uniform calls.

Also, I wanted to get rid of the unnatural separation between the client-side and server-side view of the Web in Java, following the sound advice of Benjamin Carlyle in a recent blog post. In today's networked environment, we shouldn't have to make such differences; anybody should be able to act, at the same time, as a Web client and as a Web server. In REST, every component can have as many client and server connectors as useful, so I simply developed a client HTTP connector based on the HttpURLConnection class mentioned above. Of course, other implementations could be provided, like one based on the more advanced Jakarta Commons HTTP Client.

After several iterations, it became clear that it would be beneficial for developers to separate the Restlet project in two parts. The first part is a generic set of interfaces, called the Restlet API, including a few helper classes and a mechanism to register a Restlet implementation. The second part is a reference implementation, called the Noelios Restlet Engine, including a HTTP server connector; HTTP, JDBC and SMTP client connectors; a set of representations (based on strings, files, streams, channels or FreeMarker templates) and a DirectoryRestlet able to serve static files from a tree of directories with automatic content negotiation base on file extensions.

Conclusion

While powerful for complex centralized models, the object-oriented paradigm isn't the best suited for Web development. Java developers need realize this and start thinking more RESTfully when developing new Web services or new AJAX-based Web clients. The Restlet project is providing a simple yet solid foundation that can get you started right away on the Web 2.0.

Jérôme Louvel (blog)

Notes

- For an eye-opening view on the Web 2.0, read "Piggy Bank, Cocoon and the Future of the Web" by Stefano Mazzocchi.
- Thanks to Jean-Paul Figer for the insightful discussions on REST.
- Thanks to Thierry and Fabrice Boileau for the useful feed-back on Restlets.

Version 1.4, last modified on 2005/12/22