Restlet: How I Learned to Stop Worrying and Finally REST

OJUG 2012 February

Paul Heaberlin

Outline

- What is REST?
- Features of Restlet
- Demo
  - Source code available on github.com/pheaber

- How many here have heard of REST? Who's used it on a real project? Well, for those of you who are unfamiliar with it we'll start off with a brief overview.
- After the quick REST overview, we'll get to the real meat. What makes Restlet so special?
- Then we'll get into a couple examples I've cobbled together. For those who want to play along at home, the source is available up on my github.

What is REST?

What is REST?

- REST: REpresentational State Transfer
  - Roy Fielding 2000 doctoral dissertation
- More than just Web Services with URLs
- Well-defined interface
- Stateless communication
- Access resources as representations
- Perform actions to transfer state

- REST is an architecture, not a protocol. It defines a way of accessing systems that follow certain constraints that allow for scalable, performant systems to remain simple and easily extensible in the future.
- Some people wrongly view rest as just a twist on SOAP and it's just a web service with URLs. REST separates concerns in a clean manner to go beyond that.
- REST separates the client and server by placing a well defined interface between the two. This allows both servers and clients to be replaced and developed independently as long as the interface between them is maintained.
- REST also requires for all communication to be stateless. The server may choose to maintain some state in order to optimize performance, but the client *must* pass all necessary information with each interaction.
- A REST API will have resources (books, bank accounts, etc) that are accessed using verbs to change state between the client and server.

REST: Nouns

- Things we care about
- Any coherent and meaningful concept that may be addressed
  - http://someserver/books
  - http://someserver/books/Vonnegut
  - http://someserver/authors
  - http://someserver/authors/Vonnegut
- Resources are not related to their representations

- The first part of REST is what you're representing. The Users of a system. The status updates. The bank accounts. Basically anything that is meaningful and unique enough to have a "name" or way of addressing it.
  - Examples of resources include..
- Logical names allow you to pass references to data rather than the data itself. This can minimize network traffic since a client can choose whether or not to access that data via the reference at a later point. It's also beneficial because when the reference is requested later, security can be checked at that point and therefore be more fine-grained.
- Resources are what you access, not how you access them or what they look like. Data may be stored in a database, but could be viewed as HTML, JSON or XML.

REST: Representations

- How we represent the nouns
- Structural form of a resource
- Content Negotiation eases communication

- Representations are how the client and server speak to each other about a resource.
- Content negotiation allows clients and servers to agree on what form they want to use

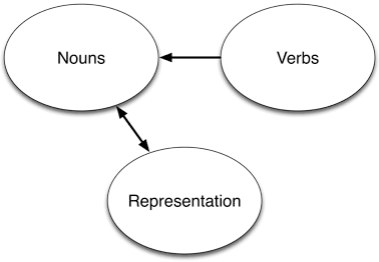
REST: Verbs

- How we manipulate the nouns
- GET
  - list all resources of certain type
  - retrieve specific resource
- POST
  - add a new instance of resource
- PUT
  - replace entire list or single resource
- DELETE
  - delete entire list or single resource

- GET: how clients ask for the content they want
- POST and PUT are somewhat similar. The difference comes from whether you know the name of the resource you are accessing or not.
  - POST: creates new instance (the instance isn't there, so you can't know its name)
  - PUT: updates existing instance (since it exists, it has a name and you can access it via that reference)
- DELETE: get rid of a resource
- GET, PUT and DELETE are idempotent
- idempotent: can do something multiple times without changing the result

Benefits of REST

- Separation of concerns
  - Things we care about
  - How we refer to them
  - How we manipulate them
  - How we represent them
- Simple and flexible



- By separating the pieces of what we care about, how we refer to and manipulate those things, and how they're represented we have much simpler and more flexible systems.
- As long as we follow the defined interface, the client and server can change their implementation without the other knowing any different. Sure, this is possible to some extent with SOAP, but REST has more opportunities to ease those changes.
- The ability to name requests and results via URLs makes it easier to identify points to cache.

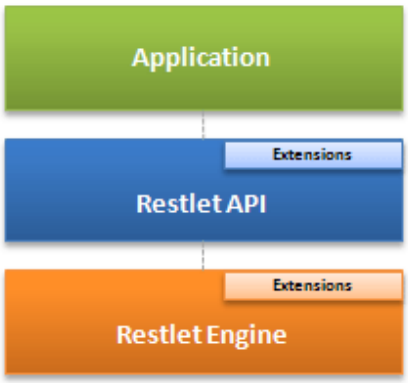
Features of Restlet

History of Restlet

- Jérome Louvel working with REST in Java
- Three goals for Restlet
  - Simple actions should be simple for basic usage
  - Code should be portable across containers
  - Enrich client side of using RESTful interfaces
- Separates transport from application
- Adding support for new tech

- Jérome Louvel began working with REST in Java. He tried a couple existing frameworks, but wasn't completely satisfied with them. Since he was a software engineer, he decided to make his own.
- Restlet Goals:
  - Simple things should be simple. Seems simple enough. The defaults should just work, but be available for more complex configurations.
  - If code is written strictly to the Restlet API, it should run in whichever environment it is thrown into such as Tomcat, Jetty, Grizzly or anything else someone writes an extension for.
  - Why let the server have all the fun? Clients need to access these services too, so there should be an easy way to do so.
- By separating the transport protocol from the application connectors for different protocols can be added (e.g. HTTP, SMTP, JDBC). This frees Restlet from being tied to HTTP like Servlets are.
- We'll see in later slides how Restlet is expanding as new technologies become available.

Restlet Architecture



The application uses the Restlet API directly to provide a RESTful service.
The Restlet API is backed by the Restlet Engine.
The Engine handles tying the connectors within a component together.

Features of Restlet

- Native REST Support
- Complete Web Server
- Varied connectors available
- Support for many representations
- Integrated Security

- Native REST: Core REST concepts have Java classes (e.g. Resource, Representation, Connector, etc). Same API used by both client and server. Tunneling service allows browsers to invoke any HTTP method through HTTP POST.
- Web Server: Static file support. Transparent content negotiation. Standard W3C log output.
- Connectors: HTTP connectors for Jetty, Simple or Grizzly. AJP connector to sit behind Apache HTTP or IIS.
- Representations: XML support for JAXB, JiBX, DOM or SAX. Can use FreeMarker or Velocity templates. Uses Apache FileUpload to handle multi-part forms and large file uploads from browsers. JSON support via Jackson or JSON.org libraries.
- Security: HTTP Basic and Digest, Amazon S3, OAuth and more

Special Features

- Web App Description Language (WADL)
- Java API for RESTful Services (JAX-RS)
- Cloud deployment
- Google Web Toolkit support
- Android support

- Enterprisey Features:
  - WADL: Sort of like WSDL for RESTful APIs. WSDL 2.0 can also describe REST Web Services, so there's a little competition here.
  - JAX-RS: Also known as JSR-311. Uses annotations to define resources (@Path), behavior (@GET, @PUT, etc), and content type (@Produces, @Consumes). Public release is planned for Spring 2012.
- Cloud: Can deploy to Google App Engine, Amazon Beanstalk and Microsoft Azure

Demo

Hello World

Demo: Hello World

- Based on the Restlet First Steps
- SimpleServerResource
- SimpleClient
- Start the server, go to http://localhost:8182/

- When I say it's "based on" the Restlet documentation, I mean it's blatantly stolen from there. But we all gotta start somewhere.
- First up, SimpleServerResource. We'll use the @Get annotation to tell Restlet which method we want to use to handle GET requests.
- SimpleClient is just that, an insanely simple client. It just connects to the given URL
- So let's look at the source real quick and then run this thing.
  - Show how changing the path in the client reads it since we didn't limit the server
    - Go to http://localhost:8182/some/long/path
  - Change the server to be more finely scoped, show the difference
    - Router router = new Router(); router.attach("http://localhost:8182/simple", SimpleServerResource.class); new Server(Protocol.HTTP, 8182, router).start();
  - Add another method for a different media type, use REST Console to hit it
    - @Get("json") public String toJSON() { return "json"; }
  - Kill the server, watch as the client tries to connect but handles the failure gracefully

Demo

Stutter

Demo: Stutter

- Twitter Clone
  - User
  - Mumble
- Supporting Interfaces
  - UserListManager
  - UserManager
  - MumbleListManager
  - MumbleManager
- StutterServerApplication
- StutterClient
- FakeStorage

- Stutter is just a simple YATC (Yet Another Twitter Clone). You've got Users and Mumbles
- There are a handful of supporting interfaces, which as you'll see are straight Java with very little Restlet Magic. But the little that's there, is mighty powerful stuff.
  - UserListManager has methods to view all Users and make a new User
  - UserManager manages a specific User, so it allows you to view, update or delete that User
  - MumbleListManager is similar to UserListManager, except it views all Mumbles and makes new Mumbles
  - Anyone guess what MumbleManager does? Yep, it handles a single Mumble so you can view or delete it.
- These services are all tied together in the StutterServerApplication. This basically just maps the URIs to their appropriate service and allows a little magic to parse out identifying pieces for later use.
- The StutterClient is just a simple example of using Restlet's ClientResource class to do sweet Restlet magic. Similar to many Java containers, you ask for an instance of a given interface and a dynamic proxy is generated to handle calls to it. To your code, it appears to be a normal object but behind the scenes Restlet is making all the necessary HTTP calls and representation conversions for you!
- FakeStorage is exactly what it sounds like... the lazy presenter's way of persisting data! Not of real concern for the app and no tie to Restlet at all, just

- a few simple Concurrent Collections so that I didn't have to really do anything with a database.

Demo: Stutter API v1

- User Access
  - /viewuser
  - /newuser
  - /updateuser
  - /removeuser
- Mumble Access
  - /viewmumble
  - /newmumble
  - /removemumble

So, now that we've gone over the classes, what should our API look like? A beginner's mistake is to assume you've made a RESTful API just by not using SOAP and exposing data at URIs. But really all you've done is exchanged SOAP-based RPC for REST-based RPC. You're using HTTP verbs for access, but you're not accessing resources. The API reads like it has behavior in it instead of just defining resources to be accessed. This is a code smell!

Demo: Stutter API v2

- User Access
  - /users
    - GET views list of all users
    - POST makes new user
  - /users/{username}
    - GET views requested user
    - PUT updates user
    - DELETE removes user

Here we've got a more proper RESTful API. When we want to get Users, we access that named resource. The behavior comes from the verb we choose to use with it.

Demo: Stutter API v2

- Mumble Access
  - /mumbles
    - GET views list of all mumbles
    - POST makes new mumble
  - /mumbles/{id}
    - GET views requested mumble
    - DELETE removes mumble

Similar stuff about the Mumble access.

Demo: Stutter API

- Start StutterServerApplication
- Open Chrome REST Console
  - Make a couple Users
  - View the User list
  - View a specific User
  - Make some Mumbles
  - View the Mumble list
  - View a specific Mumble
  - Delete a Mumble
  - Delete a User

- Who wants to see if this will run?

Demo: Stutter API Extras

- Add support for XML in addition to JSON
- Add support to list all Mumbles by User
  - /users/{username}/mumbles
- Move to the REST Holy Grail of HATEOS
  - Add references

- Now to add some flare to the Stutter API. We saw in the Hello World how easy it was to let Restlet handle the content negotiation, let's see if we can add XML support to a Mumble. Since this is just a proof of concept, I was lazy and used straight DOM. A better production implementation would use either JAXB or JiBX binding and let Restlet handle the heavy lifting. For now we'll just add capability to view and make new Mumbles with XML.
- If we've got time, how hard would it be to add a call for all Mumbles by a requested user?
- Something we won't cover here, but a big future improvement would be to advance to the REST Holy Grail of HATEOS (which I'll quickly brush over in a minute). HATEOS is the true dream of REST as put forth by Mr Fielding. From a simple starting point, all other access is discovered as the API is used. Also, the API contains references to other services available within it so that the locations of the specific resource can change.

Further Discussion on REST

- Richardson Maturity Model
- Haters gonna HATEOS
- Versioning REST Web Services
- Clean URL

- Richardson Maturity Model (RMM) comes from the REST in Practice book and was blogged about by Martin Fowler. It discusses moving from a simplified RPC that exists in the Swamp of POX (Plain ol' XML), to use of Resources, then the addition of HTTP Verbs and finally to the full Glory of REST.
- HATEOS: Hypermedia as the Engine of Application State
- Haters gonna HATEOS: Another discussion of the RMM
- Versioning REST Web Services: Interesting discussion about using custom content-types instead of versions within URLs
- Clean URL is a concept that refers to a URL without a query string to make URLs more easily understood and remembered by people, not just machines

Well, if we were on Chappelle's Show, I'd probably have gotten this a long time ago... but, are there any questions?

References

- How I Explained REST to My Wife
- Wikipedia - REST
- REST for Java Developers by Brian Sletten:
  - Part 1: It's about the information, stupid
  - Part 2: Restlet for the weary
  - Part 3: NetKernel
  - Part 4: The future is RESTful
- Restlet in Action
- Restlet.org
- Restlet API Javadoc
- Chrome REST Console

- How I Explained REST to My Wife is a fairly plain language explanation of REST and not as sexist as it sounds. Could've easily been called "How I explained REST to my non-computer-savvy friend"
- REST for Java Developers is a series of articles by Brian Sletten that explain REST, look at Restlet and NetKernel, and explore the future of REST
  - Part 1: Overview of REST
  - Part 2: Overview of Restlet. Good example of defining proper URL and worth it to check out the Jazzy spell checker
  - Part 3: Examines NetKernel, another Java Framework for REST and Resource Oriented Computing
  - Part 4: Future of REST. Resource description and discovery, linked data (LOD, RDF, Semantic Web). Sales pitch for REST.
- Restlet in Action is a forthcoming book by Jérôme Louvel (the lead author of the Restlet Framework), Thierry Templier and Thierry Boileau available from Manning
- Chrome REST Console is helpful for testing as you've seen in the demo