REST & Caching: Web Services, Accelerated

JAOO 2009

Stefan Tilkov, innoQ
The REST Uniform Interface

identification of resources  resource manipulation through representations  hypermedia as the engine of application state  self-descriptive messages
The REST Uniform Interface

| identification of resources | resource manipulation through representations | hypermedia as the engine of application state | self-descriptive messages |

http://example.com/customers/1234
http://example.com/orders/2007/10/776654
http://example.com/products/4554
http://example.com/processes/sal-increase-234
The REST Uniform Interface

- Identification of resources
- Resource manipulation through representations
- Hypermedia as the engine of application state
- Self-descriptive messages

GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml

<customer>...
</customer>

GET /customers/1234
Host: example.com
Accept: text/x-vcard

begin:vcard
...
end:vcard
The REST Uniform Interface

| identification of resources | resource manipulation through representations | hypermedia as the engine of application state | self-descriptive messages |

<order self='http://example.com/orders/3321'>
  <amount>23</amount>
  <product ref='http://example.com/products/4554' />
  <customer ref='http://example.com/customers/1234' />
  <link rel='edit' rel='http://example.com/order-edit/ACDB' />
</order>
The REST Uniform Interface

- Identification of resources
- Resource manipulation through representations
- Hypermedia as the engine of application state
- Self-descriptive messages

GET /service/customers/1234 HTTP 1.1
Host: www.example.com
User-Agent: XYZ 1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Fri, 02 Oct 2009 16:47:31 GMT
If-None-Match: "600028c-59fb-474f6852c9dab"
Cache-Control: max-age=60

HTTP/1.1 304 Not Modified
Date: Sun, 04 Oct 2009 19:36:25 GMT
Server: Apache/2.2.11 (Debian)
Last-Modified: Fri, 02 Oct 2009 16:47:31 GMT
Etag: "600028c-59fb-474f6852c9dab"
Cache-Control: max-age=300
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 7160
Keep-Alive: timeout=15, max=91
Connection: Keep-Alive
Content-Type: application/xml

<?xml version='1.0' encoding='utf-8'?>
...
getOrderDetails()
submitApplicationData()
updateQuote()
findMatchingBid()
initiateProcess()
cancelSubscription()
listAuctions()
getUsers()
<table>
<thead>
<tr>
<th>GET</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>getOrderDetails()</td>
<td>initiateProcess()</td>
</tr>
<tr>
<td>findMatchingBid()</td>
<td>submitApplicationData()</td>
</tr>
<tr>
<td>listAuctions()</td>
<td></td>
</tr>
<tr>
<td>getUsers()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PUT</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateQuote()</td>
<td>cancelSubscription()</td>
</tr>
</tbody>
</table>

Tuesday, October 6, 2009
interface Resource {
    Resource(URI u)
    Response get()
    Response post(Request r)
    Response put(Request r)
    Response delete()
}

class CustomerCollection : Resource {
    ...
    Response post(Request r) {
        id = createCustomer(r)
        return new Response(201, r)
    }
    ...
}
Caching Models

Note: Thanks to Ryan Tomayko for letting me steal-base some work on his diagrams from http://tomayko.com/writings/things-caches-do
The diagram illustrates the process of sending a "Hello World" message from a Client to a Backend using a Cache. The Client initiates the process by sending a GET /welcome request to the Cache. The Cache then forwards this request to the Backend. The Backend returns a 200 OK response with the message "Hello World" to the Cache. The Cache then forwards this response to the Client, completing the process.
Expiration
GET /welcome

GET /welcome

"Hello World"
(calculating max tolerable age)

200 OK
Cache-Control: max-age=600
Hello World!

200 OK
Cache-Control: max-age=600
Hello World!
GET /welcome

... because the cache is fresh!

200 OK
Age: 30
Cache-Control: max-age=600
Hello World!

This does not happen...
Validation
GET /welcome

200 OK
Last-Modified: Tue, 28 Oct...
ETag: a3e455afddd
Hello World

generate validators;
"Hello World"

200 OK
Last-Modified: Tue, 28 Oct...
ETag: a3e455afddd
Hello World
Bob

GET /welcome

Cache

GET /welcome
If-Modified-Since: Tue, 28 Oct...
If-None-Match: a3e455afdd

Backend

generate validators;
DO NOT generate response

304 Not Modified

Backend

Cache

Bob

200 OK
Last-Modified: Tue, 28 Oct...
ETag: a3e455afdd
Hello World
Combination
GET /welcome

200 OK
Cache-Control: max-age=60
Last-Modified: Tue, 28 Oct...
Hello World

generate validators: "Hello World"
GET /welcome

... because the cache is fresh!

200 OK
Cache-Control: max-age=60
Age: 30
Last-Modified: Tue, 28 Oct...
Hello World
GET /welcome

GET /welcome
If-Modified-Since: Tue, 28 Oct...

304 Not Modified
Cache-Control: max-age=60

200 OK
Cache-Control: max-age=60
Last-Modified: Tue, 28 Oct...
Hello World

generate validators;
DO NOT generate response
Implementation
Apache FileETag

FileETag  INode  MTime  Size

- file's i-node number
- date and time the file was last modified
- number of bytes in the file

Tuesday, October 6, 2009
ETag Depth
The diagram illustrates the interactions between a Client, Cache, and Smart Back End. The process involves:

1. The Client initiates a request with a GET /welcome operation to the Cache.
2. The Cache checks if the response is available in its cache.
3. If the response is in the cache, it returns the response to the Client.
4. If not, the Cache fetches the response from the Smart Back End and returns it to the Client.
5. The Client then sends a request to the Smart Back End with GET /welcome, including an ETag.
6. The Smart Back End checks the ETag and returns a response if the content has changed.

This cycle ensures efficient content delivery by leveraging caching mechanisms.
Edge Side Includes (ESI)
<esi:include
    src="http://example.com/1.html"
    alt="http://bak.example.com/2.html"
    onerror="continue"/>

<esi:include
    src="http://example.com/search?query=${QUERY_STRING{query}}"/>
GET /AB

... because the cache is fresh!

Aggregate A + C

GET /C

Return Response
Cache Topologies
Client only

Proxy Cache

Reverse Proxy Cache

Complex Topology

Tuesday, October 6, 2009
"Cache as ESB"
Web Sites vs. Web Services
Application Layers

Presentation/UI
- Browser

Data Formatting
- View

Control Flow
- Controller

Aggregate Logic
- Model

Business Rules
- Lib

Relations
- Database

Data (create, select, update, delete)

Utility Functions

Tuesday, October 6, 2009
Single Resource Model

Browser

(Other) REST Client

Application

Resource

Resource

Resource

Everything doable via UI ...

... becomes doable via API

UI Backend = API
Intermediaries
Squid

Full proxy cache w/ reverse proxy option
Mature/stable/old, widely used
Complicated configuration
Support for ESI
Support for external invalidation (PURGE)
(Experimental) support for cache channels
(much, much more)
mod_cache

mod_cache module for Apache HTTPD 2.x (production-ready in 2.2)

backends:
   mod_mem_cache and mod_disk_cache

Runs within Apache process

Simple solution in conjunction with Passenger

No support for ESI or explicit invalidation
<VirtualHost 1.2.3.4>
    ServerName example.com
    <Location /images>
        ExpiresActive On
        ExpiresDefault A3600
    </Location>
    <Location /user>
        ExpiresActive On
        ExpiresDefault "access plus 1 month"
    </Location>
    CacheEnable disk /images
    CacheRoot /var/www/cache
    ProxyPass / http://localhost:3000
    ProxyPassReverse / http://localhost:3000
</VirtualHost>
Varnish

Pure in-memory reverse proxy
Disk storage through OS swap mechanism
(Partial) ESI support
External invalidation
VCL configuration language
Easy configuration
sub vcl_recv {
    if (req.request != "GET" &&
        req.request != "HEAD" &&
        req.request != "PUT" &&
        req.request != "POST" &&
        req.request != "TRACE" &&
        req.request != "OPTIONS" &&
        req.request != "DELETE") {
        /* Non-RFC2616 or CONNECT which is weird. */
        return (pipe);
    }
    if (req.request != "GET" && req.request != "HEAD") {
        /* We only deal with GET and HEAD by default */
        return (pass);
    }
        /* Not cacheable by default */
        return (pass);
    }
    return (lookup);
}
Rack::Cache

Runs within Ruby process
Cross-process synchronization via memcached
Absurdly simple + transparent

```ruby
config.middleware.use(Rack::Cache,
  :verbose => true,
  :metastore => 'file:/var/cache/rack/meta',
  :entitystore => 'file:/var/cache/rack/body')
```
Clients
Cache-aware HTTP Clients

**Note**: Conditional GET does not require anything special

Your browser and news reader

.NET (HttpWebRequest)

Python httplib2

HttpCache4J
So what?
How to Exploit HTTP Caching

1. Provide “safe” resource access via GET
2. Include appropriate validation and cache control headers
3. Leave caching to a reverse proxy cache
4. Use conditional GET on the client
5. Profit :-)*
“I do think the REST-afarians are missing an opportunity by not driving home the secret sauce that is HTTP GET. […] GET is one of the most optimized pieces of distributed systems plumbing in the world. It's an absolute/objective slam dunk. No arguing/evangelism needed IMO. GET is the classic ‘the first bag is free’ kind of feature a platform builder dreams about.”

*Don Box, Co-inventor of SOAP*
Thank you!
Any questions?

http://www.innoq.com
Backup
Turning Session State ...
... into Resource or Client State
Resources

Nothingham, Mark: The State of Proxy Caching
http://www.mnot.net/blog/2007/06/20/proxy_caching

Nothingham, Mark: The State of Browser Caching
http://www.mnot.net/blog/2006/05/11/browser_caching

Nothingham, Mark: Caching Tutorial
http://www.mnot.net/cache_docs/

Squid, http://www.squid-cache.org/

Nothingham, Mark: Cache Channels for Squid
http://www.mnot.net/cache_channels/

The Varnish Project, http://varnish.projects.linpro.no/

Varnish: Notes from the Architect
http://varnish.projects.linpro.no/wiki/ArchitectNotes

Tomoyako, Ryan: Things Caches Do
http://tomayko.com/writings/things-caches-do

W3C, ESI Language Specification 1.0
http://www.w3.org/TR/esi-lang

Apache HTTP 2.2 Caching Guide
http://httpd.apache.org/docs/2.2/caching.html