RESTful Web Apps

Facts vs Fiction

Subbu Allamaraju
Yahoo! Inc || http://subbu.org
About

- Tech Yahoo!
  - Developing standards, patterns and practices for HTTP web APIs
- Past
  - Developer – web services and Java
  - Standards contributor at BEA
  - Wrote books on JEE web tier (so long ago)
- Current Passion
  - HTTP and REST
Disclaimer

All the opinions I express here are mine and do not necessarily represent those of my present or past employers.
A Confession

- Not a web developer
- Not directly interested in the internals of web frameworks
- Only interested in the visible aspects of web apps
Agenda

REST – The Architecture

About RESTfulness of Web Apps
Some History

1945 - Vannevar Bush
  arbitrary linking between pieces of information
1965 – Ted Nelson (Xanadu fame)
  hypertext and hypermedia
1990s – Tim Berners-Lee, Roy Fielding et al.
  WWW, HTML, HTTP and URI
  HTTP 1.1 - RFC 2616
  URI - RFC 2396 (superceded by 3986)
2000 – Roy Fielding
  Representational State Transfer
Two Sides

RPC

HTTP, URI, WWW

REST

Web frameworks

SOAP

"Pragmatic REST"

Machine Facing

Human Facing

This session

+ REST URIs (?)
WEB IS MOSTLY RESTFUL
Being RESTful

Resources

- Named via URIs: Uniquely identify resources
- Have representations: Mostly one/resource
- Reflect the state of the app
- Contain contextual links
- Uniform interface: Generic, Client-Server
- Idempotent, safe, cacheable, ...

HATEOAS
Resources and URIs

- A blog post, an image, a catalog, a shopping cart
- Some personalized, but most are not
- Some depend on sessions, but most do not
Cost of Personalized UI

http://my.example.org

vs

http://www.example.org/subbu
Representations

- Rich of representations
  - HTML, XML, JS, PDF, CSS, Flash, ...

- Poor content negotiation
- *Agent-driven* negotiation + Poor negotiation headers
- Media types on responses ignored sometimes
  - `<a href="mydoc.pdf">Click</a>`
  - `<a href="myfeed.rss">RSS Feed</a>`
  - `<a href="myfeed.atom">Atom Feed</a>`
Hypermedia and HATEOAS

- Rich Hypermedia
  - Links and forms
  - Contextual

- Auto-discoverable
  - `<link/>`, microformats

- Except some new breed of Web
  Too Oh + REST goo
Uniform Interface

- Links and Forms
  - GET and POST

- Still some misconceptions (e.g. POST is secure)
- Idempotency? Safety?
- GET URIs not always refreshable
- Still fighting the back button
Back Button

Cache-Control: no-cache, no-store

Redirect after POST

*a.k.a.* PRG (POST/redirect/GET)
Caching

- Web is read-most
- Cacheable
- Widely discussed

- Yet moderately ignored
- Cache busting
- Cache-ignorant frameworks
- Frameworks that prefer backend caching over HTTP caching
Caching Choices

- **Back-end caching**
  - Non-uniform interface
  - Need to explicitly program to it

- **HTTP caching**
  - Uniform interface
  - Pluggable
Ajax Apps

- More opportunities to be RESTful
  - Full support for the uniform interface
  - Content negotiation, optimistic concurrency, caching
  - HATEOAS
    - Can loosen this constraint as long as the client code is downloaded from the same server/app
Cross-Domain Hacks

- script, iframe
- Tunnel requests over GET
Web is Mostly RESTful

Take advantage of the web arch

Breaking is EXPENSIVE
Breaking is CONFUSING
Breaking LOWERS expectations
WEB FRAMEWORKS
State of Affairs

Ease of programming

Fragmentation and confusion

Innovation vs Correctness
Circa 1997-

- There were servlets
- Basic plumbing, closely reflecting HTTP 1.1
- A poor programming model
- But allowed a lot of frameworks to be built on top
Circa 2001-

Action Oriented
Circa 2004-

- Enter JSF & Co.
  - A component based UI framework
- With known limitations
  - Complex
  - Slow
  - Uses POST for almost everything
  - And so on...
- Third-party patches
Design Choice
Where to keep this stuff?
We know.
So what?
URI Overloading

Client 1 \rightarrow Resource (URI) \leftarrow \text{Representation}

Client 2 \rightarrow Resource (URI) \leftarrow \text{Different Representation} \textit{How was this chosen?}
URI Overloading

- One URI – multiple representations
  - No way to tell how a representation was chosen
  - Can get wrong content from a cache
- HTTP does allow URI overloading
  - Content negotiation *aka* “conneg”
Content Negotiation

Accept: application/atom+xml;q=1.0,text/html;q=0.1
Accept-Charset: UTF-8
Accept-Language: fr;q=1.0,en=0.8
Accept-Encoding: gzip, deflate

Content-Type: application/atom+xml;charset=UTF-8
Content-Encoding: deflate
Vary: Accept,Acept-Encoding
Some Options

- Vary by cookie
  - Not always recognized by caches
  - Complex given the parameters in a cookie
    - Domain, path, life time, etc.

- URLEncode
  - Encode session ID into URIs
    - ;jsessionid=d8sdasg7312

- Cache-control: no-cache, no-store
JSF Compromise

```xml
<context-param>
  <param-name>
    javax.faces.STATE_SAVING_METHOD
  </param-name>
  <param-value>client</param-value>
</context-param>
```
Implementation

Every request must be a POST

```html
<form method="POST" action="..."
   enctype="...">
  <input type="hidden"
     name="javax.faces.ViewState"
     value="H4s...zogsAAA==" />

  ...
</form>
```

State stuffed into forms as a hidden field
Consequence

Breaks the uniform interface
JSF vs REST

Caught between two extremes

- URIs no longer sufficient to identify a resource
- Interactions assume existence of session state – i.e. no longer stateless
- Uniform interface limited to POST
- Interactions not idempotent
- Representations not cacheable
JSF – Takeaway

- Focused heavily on a UI component model (AWT for the Web)
- Misinterpreted the web architecture
- Made some fundamental questionable choices
- You can patch, but can not fix
  - 15+ Ajax patches!
WEB 2.0 MOTIVATED
GWT

- A cross-compilation based framework
  - Write Java – generate JavaScript
  - Mashes up client and server code into single source
  - These layers communicate using GWT-RPC
- Typical RPC concern does not apply
  - Coupling due to code generation
  - Client downloaded from the same app
Object result = someServ.doIt(new MyCallback());
public class MyCallback extends AsyncCallback{
{
    public void onsuccess(Object result)
    {
        ...
    }
}
}
GWT-RPC over HTTP

- POST http://www.contactoffice.com/gwt (321ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (197ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (308ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (243ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (292ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (702ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (640ms) ECF1349473B6B07F9... (line 9754)
- POST http://www.contactoffice.com/gwt (488ms) ECF1349473B6B07F9... (line 9754)
GWT-RPC over HTTP

- Method calls POSTed to the server
  - Transport object graphs
  - Uses HTTP like a transport layer
- Non-uniform interface
GWT – Request Builder

String url = "...";
RequestBuilder builder =
    new RequestBuilder(RequestBuilder.GET,
                       URL.encode(url));
builder.sendRequest(data, callback);
RequestBuilder over HTTP

- More control over HTTP requests
- Supports GET and POST
- Allows so-called RESTful layers
  - GWT-REST
  - GWT-Restlet
GWT Takeway

- Focused heavily on ease of use
  - Javascript agnostic
- Modeled after RPC
  - Breaks uniform interface
  - Backend caching over HTTP caching
- Fixable
Fixing GWT?

Object result = someServ.doGet(new MyCallback());

Object result = someServ.doPost(new MyCallback());

Object result = someServ.doPut(new MyCallback());

Object result = someServ.doDelete(new MyCallback());

Object result = someServ.doHead(new MyCallback());
SOFEA

- Central premise – SOA
  - Business logic as reusable services
    - Change less often
  - Presentation app calling those services
    - Change more often
  - Separation of concerns and Loose coupling
- Misinterprets HATEOAS
HATEOAS vs SOFEA

App download

Download server

App flow

Data interchange

App container

Representation

Server

Shared/persistent state

Business logic server

Representation with new state

Client
Appcelerator

- A SOFEOA style framework with RoR like usability
  - Attend Matt Raible’s session on “Building Rich Internet Applications with Appcelerator”

- SOAP/HTTP style
  - Message passing
  - POST to a single URI
XML over POST

```
POST http://www.skyblox.com/servicebroker?maxwait=0&instanceid=4377-832&an
```

```
POST http://www.skyblox.com/servicebroker?maxwait=0&instanceid=4377-832&a
```

```xml
<?xml version="1.0" encoding="UTF-8"?><messages version='1.0' sessionid="%0ASGFzaHsABjoKQHVzZWR7AA%3D%3D--423c023e18027a2f15a8e24acceb702d99ca478
="'OUTGOING' datatype='JSON' type='portal.get.bloxlist.response' scope='c
"location":"180 Walker St SW, 30313","id":131},"count":20,"date":"10
:true,"non_customers":["location":"177 Peters St, 30313","id":137},"loc
,"id":126},"location":"253 Peters St, 30313","id":264},"location":"281
,"location":"144 Walker St, 30313","id":138},"location":"322 Peters St
:"274 Walker St SW, 30313","id":133},"location":"261 Walker Street SW,
:"291 Peters St SW, 30313","id":128},"location":"180 Walker Street SW S
:"263 Walker St SW, 30313","id":129},"location":"244 Peters Street SW,
:"264 Peters Street SW, 30313","id":135},"location":"263 Peters St, 303
Peters Street, 30313","id":130},"location":"144 Walker Street, 30313",
```
Interesting ... but

- Loosens HATEOAS
  - Hypermedia to Code + Data
- Introduces a different kind of coupling
  - Clients deal with POX and not links
  - Breaks URI opacity
- Examples
  - Appcelerator – SOAP like
  - [http://www.contactoffice.com](http://www.contactoffice.com) - uses XML/POST
Thanks

- Don’t fight the architecture
  - Innovate, enhance
  - Don’t break
  - Or break judiciously