# No nonsense RESTful ColdFusion web services, with

# Taffy

## (A framework for ColdFusion REST APIs)



BG: flickr.com/photos/stevendepolo/4605640584/

# A little about me...

- I'm Adam Tuttle: @TuttleTree, fusiongrokker.com
- I'm an Adobe Community Professional: 2010, 2011
- I work at Wharton (UPenn), in Philadelphia
- "I would die for hyperbole" - Anonymous
- I like candy

hyperbole -> hence the "SOAP is a lie" line in the session description

# Agenda

- Why create web services & REST 101
  - Why is REST more popular than SOAP?
  - How does it work?
- Other options for creating REST with ColdFusion
- Taffy 101
- Advanced Taffy
  - Integrating with ColdSpring
  - Parent/Child applications
- REST anti-patterns

# Why create Web Services?

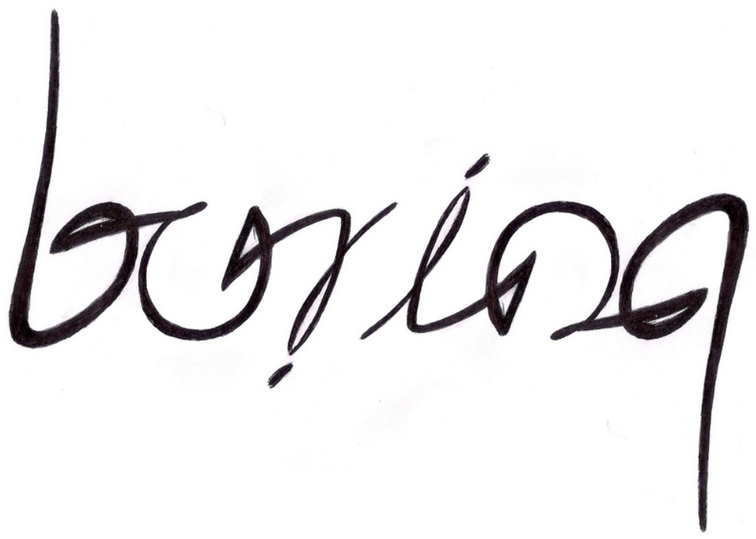
Boring reasons

- Data sharing between apps and platforms
- Prevent duplicated effort (Don’t reinvent the wheel)

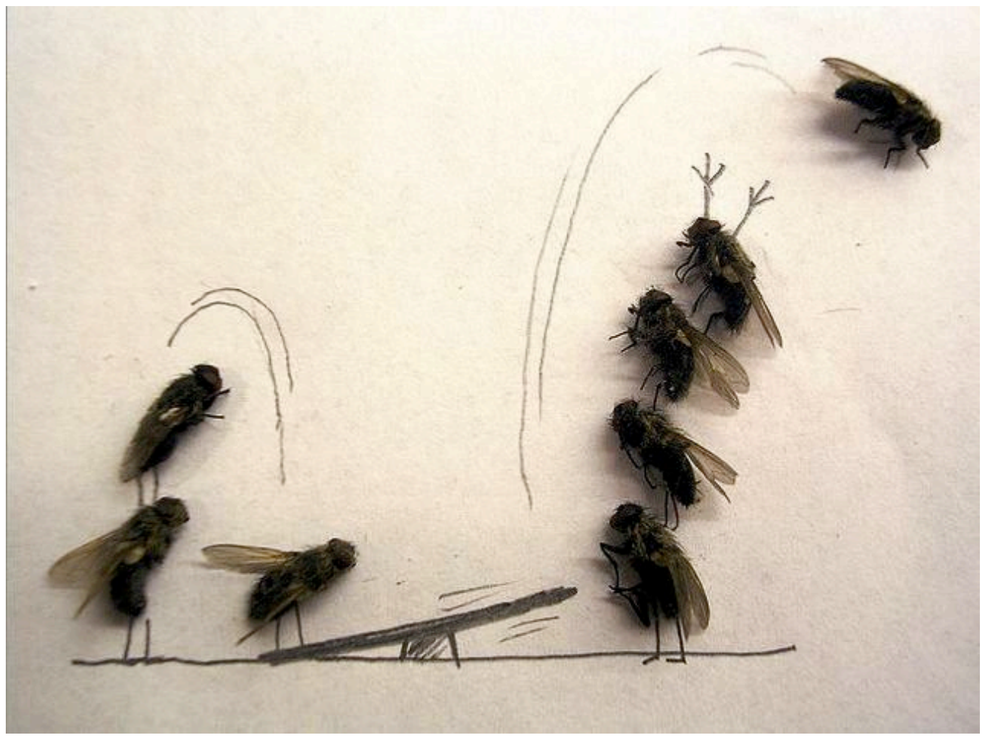
Fun reasons

- Allows richer interaction with your customers (“Mashups”)
  - Web
  - Native Desktop
  - Native Mobile Device
- Let others develop ideas you don’t have time for…
- … or that you would never have thought of!

# Why create Web Services?




http://www.flickr.com/photos/45162296@N08/4199852432/



http://www.flickr.com/photos/johnbullas/4346719298/

Boring reasons

- Data sharing between apps and platforms
- Prevent duplicated effort (Don’t reinvent the wheel)

Fun reasons

- Allows richer interaction with your customers (“Mashups”)
  - Web
  - Native Desktop
  - Native Mobile Device
- Let others develop ideas you don’t have time for…
- … or that you would never have thought of!

# Why REST, not SOAP?

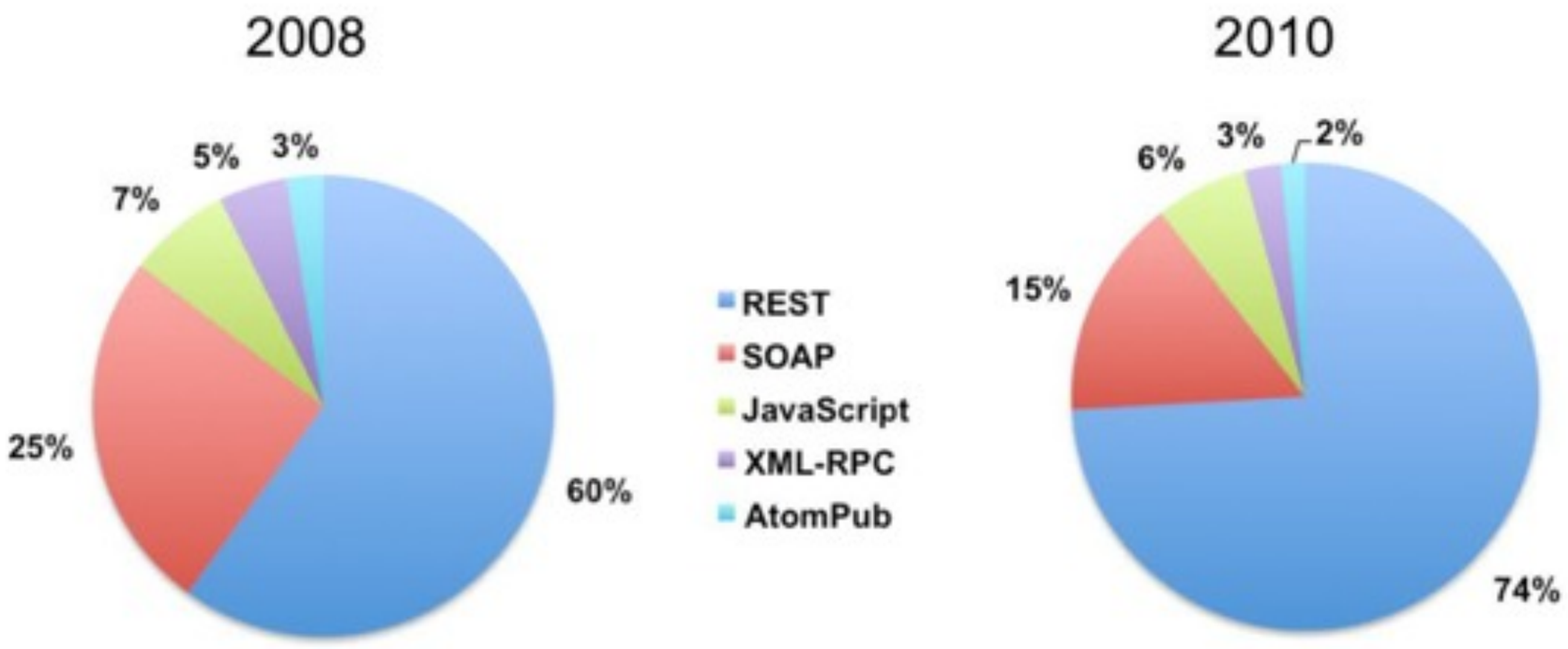
- SOAP is a lie
- REST is everything SOAP wishes it could be
  - Easier (“Simple”)
  - Optional Arguments
  - Return format is open/variable
  - The future of web services

SOAP:
- Terrible Acronym
  - Not Simple, Not Object Oriented, Not a Protocol, so just Access
- A lot like implementing CORBA on the internet :(

Easier:
- Consume - simple http request
- Scale & Cache - Leverage existing HTTP scaling & caching solutions

Future:
- VHS vs Betamax
- BR vs HDDVD
- (Did I just insinuate that the adult entertainment industry chose REST over SOAP?)

# You say you want proof?!




http://blog.programmableweb.com/2010/06/09/new-job-requirement-experience-building-restful-apis/

What's that? You say you want proof that REST is more popular?

REST +14%
SOAP -10%
Other 4% at expense of JS, XML-RPC, AtomPub

# REST 101 in 5 minutes or less

- Requests contain 3-4 parts
  - Resource (URI)
  - Verb
  - Mime Type
  - Document (optional)
- Responses contain 2 parts
  - HTTP Headers
    - Status Code
    - Status Message
    - ...
  - Response Document (optional)

30k foot view - there is nuance

Request + Response, each consist of a couple of parts

**Request:** Noun, Verb, Language, Description
**Response:** Headers, Data

# REST 101 in 5 minutes or less

- What is a Resource?
  - 2 types of Resources
    - Collection (Query)
    - Member (Struct)
  - Unique Uniform Resource Identifier (“URI”) per Resource
    - /art -or- /artists/42/art
    - /artists/1337

# REST 101 in 5 minutes or less

- What is an HTTP Verb?

```
<form method="POST">
  <input type="text" .../>
</form>
```

Tells app server what to do to the requested resource

# REST 101 in 5 minutes or less

- 4 basic verbs
  - ★ POST
  - ✪ GET
  - ☆ PUT
  - ☆ DELETE

✪ Safe ☆ Idempotent ★ Unsafe

You're already using 2 of them today (the ones in purple)

The internet, in its basic form of Gets, Form posts, and HTML documents is an implementation of REST

Safe -vs- Idempotent -vs- Unsafe

# REST 101 in 5 minutes or less

- 4 basic verbs
  - ★ POST => Create
  - ✪ GET => Read
  - ☆ PUT => Update
  - ☆ DELETE => Delete

✪ Safe ☆ Idempotent ★ Unsafe

You're already using 2 of them today (the ones in purple)

The internet, in its basic form of Gets, Form posts, and HTML documents is an implementation of REST

Safe -vs- Idempotent -vs- Unsafe

# REST 101 in 5 minutes or less

- 4 Extended verbs
  - OPTIONS
  - HEAD
  - TRACE
  - CONNECT

99% of APIs don't implement these, but you should be aware of them.

- OPTIONS: get supported/allowed features
- HEAD: Same headers as GET, but no data.
- TRACE: Echo back all request headers as a 200(OK) response for testing/debug purposes. Most web servers (including Apache) disable this by default for security reasons.
- CONNECT: Reserved for dynamic proxy/tunnel switching

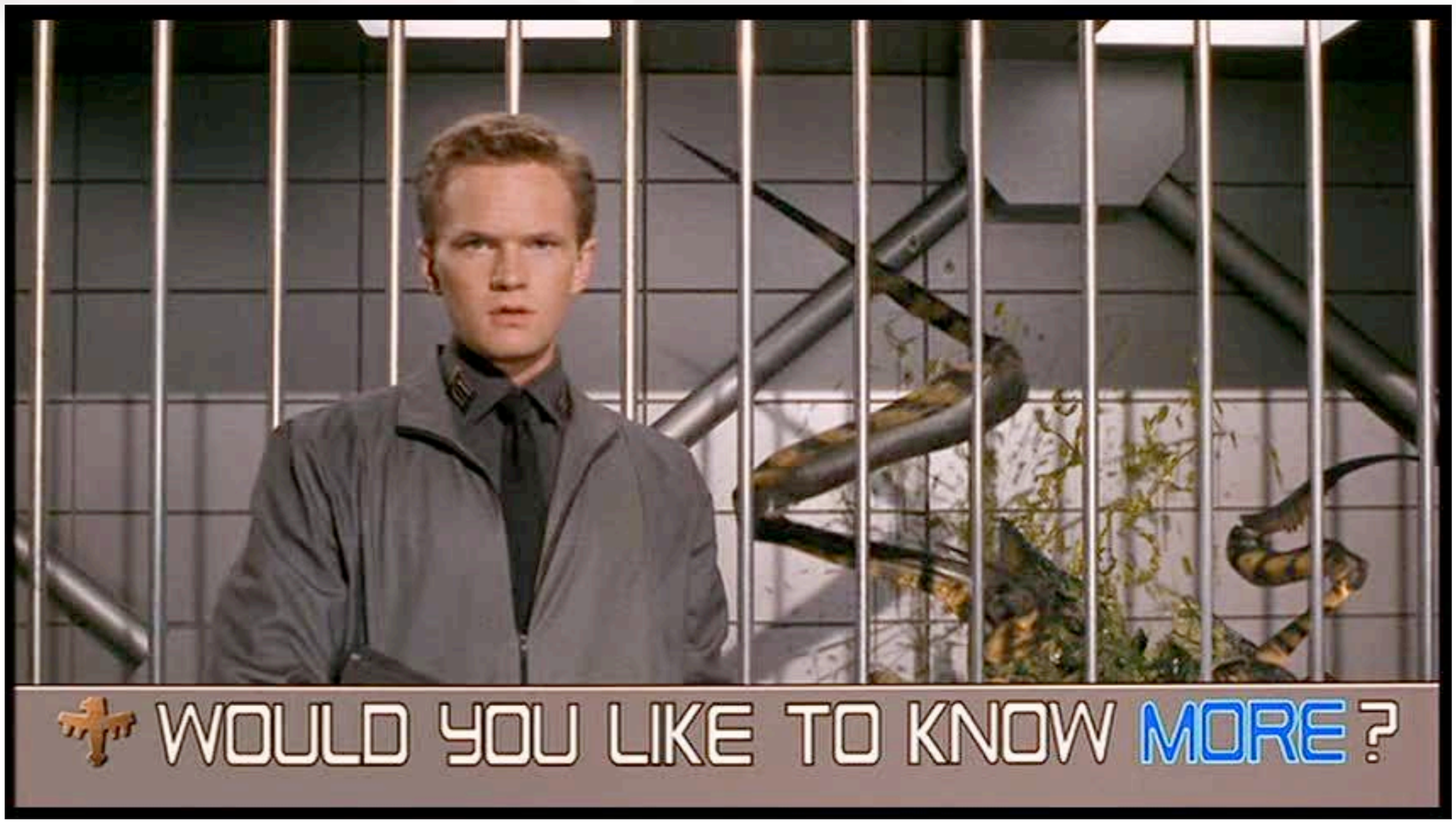
# REST 101 in 5 minutes or less

- Mime types are variable & open-ended
  - JSON
  - BSON
  - XML
  - YAML
  - WDDX
  - Your proprietary format
  - ∞

Do whatever you want

# Would you like to know more?

Everything you wanted to know about REST and more
-- Simon Free
-- Ballroom A12-B12



# Would you like to know more?

Everything you wanted to know about REST and more
-- Simon Free
-- Ballroom A12-B12

# Implementation Choices

- Do it the hard way (manually)
- Existing Frameworks
  - PowerNap
  - ColdBox
  - Mach-ii

...So that's REST in a nutshell

You want to write REST web services

* How hard could doing it manually be? I found out...
  - too much boilerplate code
  - end up doing a lot of copy+paste
  - poor maintainability, especially boilerplate
  - not elegant
  - worst of all: not fun!

# Do it the hard way...

gist.github.com/887610

* Ok, so that sucks. What are those existing frameworks again???
  - Powernap
  - Coldbox, Mach-ii

# Do it t

```
<cfsetting enablecfoutputonly="true" showdebugoutput="false" />

<cffunction name="StructKeyRequire" output="false">
	<cfargument name="s" type="struct" />
	<cfargument name="k" type="string" />
	<cfset var local = {} />
	<cfloop list="#arguments.k#" index="local.i">
		<cfif not structKeyExists(s, local.i)>
			<cfthrow type="rest.invalidRequest" message="Required field `#local.i#` is missing from your requ
		</cfif>
	</cfloop>
</cffunction>

<cftry>
	<!--- get verb --->
	<cfset verb = cgi.request_method />
	<cfset httpMethodOverride = GetPageContext().getRequest().getHeader("X-HTTP-Method-Override") />
	<cfif isDefined("httpMethodOverride")>
		<cfset verb = httpMethodOverride />
	</cfif>

	<!--- get url params --->
	<cfset params = {} />
	<cfset structAppend(params,url,true) />

	<!--- get request body --->
	<cfset requestBody = getHTTPRequestData().content />
	<cfif len(requestBody)>
		<cfset args = listToArray(requestBody,"&") />
		<cfloop from="1" to="#arrayLen(args)#" index="arg">
			<cfif findNoCase("=",args[arg])>
				<cfset params[listGetAt(args[arg],1,'=')] = listRest(args[arg],'=') />
			</cfif>
		</cfloop>
	</cfif>

	<!--- get headers --->
	<cfset headers = getHTTPRequestData().headers />

	<!--- get ID --->
	<cfif len(cgi.path_info)>
		<cfset params.id = listGetAt(cgi.path_info,1,'/') />
	</cfif>

	<!--- check for api key --->
	<cfif not structKeyExists(params, "apiKey")>
		<cfthrow type="rest.unauthorized" />
	</cfif>

	<!--- do stuff --->
	<cfswitch expression="#verb#">
		<cfcase value="get">
			<cfquery name="data" datasource="cfartgallery">
				select * from ARTISTS
				<cfif structKeyExists(params, "id")>
					where artistId = <cfqueryparam cfsqltype="cf_sql_numeric" value="#params.id#" />
				</cfif>
```

* Ok, so that sucks. What are those existing frameworks again???
  - Powernap
  - Coldbox, Mach-ii

# Do it t

```
68              </cfcase>
69              <cfcase value="put">
70                      <cfthrow type="rest.methodNotAllowed" />
71              </cfcase>
72              <cfcase value="post">
73                      <cfset StructKeyRequire(params, "firstname,lastname,addres,city,state,postalcode,phone,email,fax
74                      <cfquery name="doInsert" datasource="cfartgallery" result="insertResult">
75                              insert into ARTISTS (FIRSTNAME,LASTNAME,ADDRESS,CITY,STATE,POSTALCODE,PHONE,EMAIL,FAX,THI
76                              values (
77                                      <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.firstname#" />
78                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.lastname#" />
79                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.address#" />
80                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.city#" />
81                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.state#" />
82                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.postalcode#" />
83                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.phone#" />
84                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.email#" />
85                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.fax#" />
86                                      , <cfqueryparam cfsqltype="cf_sql_varchar" value="#params.thepassword#" />
87                              )
88                      </cfquery>
89                      <cfheader statuscode="201" statustext="Created" />
90                      <cfcontent reset="true" type="application/json" />
91                      <!--- inserted id in sql server and apache derby --->
92                      <cfif structKeyExists(insertResult, "identitycol")>
93                              <cfheader name="x-inserted-id" value="#insertResult.identityCol#" />
94                      </cfif>
95                      <!--- inserted id in mysql --->
96                      <cfif structKeyExists(insertResult, "generated_key")>
97                              <cfheader name="x-inserted-id" value="#insertResult.generated_key#" />
98                      </cfif>
99                      <cfabort />
100             </cfcase>
101             <cfcase value="delete">
102                     <cfthrow type="rest.methodNotAllowed" />
103             </cfcase>
104     </cfswitch>
105
106     <!--- serialize result --->
107     <cfset output=serializeJson(data) />
108
109     <!--- set return headers --->
110     <cfheader statuscode="200" statustext="OK" />
111     <cfcontent reset="true" type="application/json" />
112
113     <!--- output return body --->
114     <cfoutput>#output#</cfoutput>
115
116     <!---
117             handle potential errors
118     --->
119     <cfcatch type="rest.invalidRequest">
120             <cfheader statuscode="400" statustext="Bad Request" />
121             <cfcontent reset="true" type="application/json">
122             <cfoutput>{"errors":["#cfcatch.message#"]}</cfoutput>
123     </cfcatch>
124     <cfcatch type="rest.unauthorized">
```

* Ok, so that sucks. What are those existing frameworks again???
  - Powernap
  - Coldbox, Mach-ii

# Do it t

* Ok, so that sucks. What are those existing frameworks again???
  - Powernap
  - Coldbox, Mach-ii

# Do it t

```
119         <cfcatch type="rest.invalidRequest">
120                 <cfheader statuscode="400" statustext="Bad Request" />
121                 <cfcontent reset="true" type="application/json">
122                 <cfoutput>{"errors":["#cfcatch.message#"]}</cfoutput>
123         </cfcatch>
124         <cfcatch type="rest.unauthorized">
125                 <cfheader statuscode="401" statustext="Unauthorized" />
126                 <cfcontent reset="true" type="application/json">
127                 <cfoutput>{"errors":["API Key is required for all requests"]}</cfoutput>
128         </cfcatch>
129         <cfcatch type="rest.notFound">
130                 <cfheader statuscode="404" statustext="Not Found" />
131                 <cfcontent reset="true" type="application/json">
132                 <cfoutput>{"errors":["The ID you specified was not found"]}</cfoutput>
133         </cfcatch>
134         <cfcatch type="rest.methodNotAllowed">
135                 <cfheader statuscode="405" statustext="Method Not Allowed" />
136                 <cfcontent reset="true" type="application/json">
137                 <cfoutput>{"errors":["The method you used is now allowed for this resource"]}</cfoutput>
138         </cfcatch>
139 </cftry>
```

* Ok, so that sucks. What are those existing frameworks again???
  - Powernap
  - Coldbox, Mach-ii

# Do it t

```
119         <cfcatch type="rest.invalidRequest">
120                 <cfheader statuscode="400" statustext="Bad Request" />
121                 <cfcontent reset="true" type="application/json">
122                 <cfoutput>{"errors":["#cfcatch.message#"]}</cfoutput>
123         </cfcatch>
124         <cfcatch type="rest.unauthorized">
125                 <cfheader statuscode="401" statustext="Unauthorized" />
126                 <cfcontent reset="true" type="application/json">
127                 <cfoutput>{"errors":["API Key is required for all requests"]}</cfoutput>
128         </cfcatch>
129         <cfcatch type="rest.notFound">
130                 <cfheader statuscode="404" statustext="Not Found" />
131                 <cfcontent reset="true" type="application/json">
132                 <cfoutput>{"errors":["The ID you specified was not found"]}</cfoutput>
133         </cfcatch>
134         <cfcatch type="rest.methodNotAllowed">
135                 <cfheader statuscode="405" statustext="Method Not Allowed" />
136                 <cfcontent reset="true" type="application/json">
137                 <cfoutput>{"errors":["The method you used is now allowed for this resource"]}</cfoutput>
138         </cfcatch>
139 </cftry>
```

gist.github.com/887610

* Ok, so that sucks. What are those existing frameworks again???
  - Powernap
  - Coldbox, Mach-ii

# PowerNap

```
<cfset map().get().uri("/myresource/{id}").to("myResource").calls
("someGetMethod") />

<cfset map().put().uri("/myresource/{id}").to("myResource").calls
("somePutMethod") />

<cfset map().post().uri("/myresource/{id}").to("myResource").calls
("somePostMethod") />

<cfset map().delete().uri("/myresource/{id}").to("myResource").calls
("someDeleteMethod") />
```

* each resource+verb is yet another line of config => 1-4 lines per resource

# ColdBox

```
addRoute(pattern="users/:user",
    handler="rest.Users",
    action={
        GET = "show",
        POST = "create",
        DELETE = "remove",
        HEAD = "info"
    }
);
```

Little more terse, but requires ColdBox.

# Mach-ii

```
<endpoints>
    <endpoint name="content"
              type="components.endpoints.ContentServiceEndpoint" />
</endpoints>


<cffunction name="getContentItem"
            access="public" returntype="String" output="false"
            rest:uri="/content/item/{key}"
            rest:method="GET">
     <cfargument name="event"
                 type="MachII.framework.Event"
                 required="true" />
</cffunction>
```

http://trac.mach-ii.com/machii/wiki/IntroToRESTEndpoints

**Released in Mach-ii 1.9 Milestone 2**

Surprisingly similar to Taffy...
...which tells me I had some good ideas!
 - developed somewhat in parallel w/out knowledge of each other

# Problems w/ current options

- Doing it the hard way sucks in ColdFusion! (lte CF9)
- PowerNap uses too much config for my taste
- ColdBox & Mach-ii are fine options, but...
  - More than just REST, and
  - Not the MVC framework used by my team or myself

* Doing it the hard way *is* possible, it just sucks
    * LTE-CF9 because REST will be in CF10 (but nobody knows what it will look like yet)

* Nothing against other framework inclusions, but if I'm already using Model Glue or FW/1, I'm not going to rewrite my app to add REST, and adding Mach-ii or ColdBox to the stack just to get REST is overkill.

# Solution?

After presenting current state of REST in CF in May 2010, used WAVE (only time it's ever been useful)

The one and only time I’ve found google wave useful...

# Solution?


After presenting current state of REST in CF in May 2010, used WAVE (only time it's ever been useful)

The one and only time I’ve found google wave useful...

# Solution?


Eventually we had an informal spec for a new framework

# Goals of Taffy

- Semantically correct URIs
- Eradicate configuration & “boilerplate”
- Separate logic from presentation
- Empower developer to conform to REST/HTTP spec
- Easy to learn & use
- Fast

1.0 was finalized in August 2010

Eradicate config:

- Simple mapping of URI to Resource
- Convention mapping HTTP verbs to resource methods
- Where config is necessary: use Metadata if possible

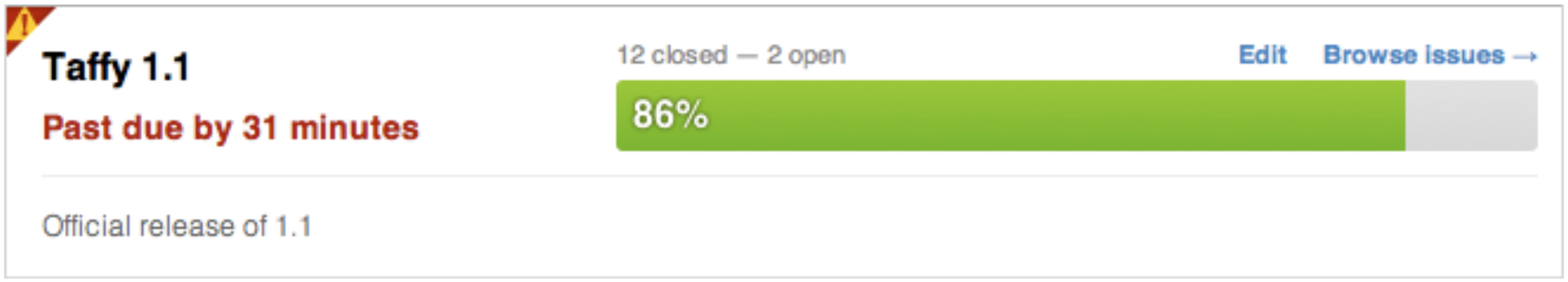
# Announcing...

# Taffy 1.1

officially released as of ~~today~~ soon!

1.1 released on 5/13

# Soon?




- Trying to finalize Railo & OpenBD compatibility testing
- Updating documentation
- A few examples are buggy

# Demo

(avoiding the law of live coding)

check time: ~15 minutes elapsed

# Section 0:
# Creating a new Taffy API

## /api/Application.cfc

```
<cfcomponent extends="taffy.core.api">
   <cfset this.name = "SomeApi" />
</cfcomponent>
```

## index.cfm

```
{empty}
```

Basic API, doesn't do anything but the framework is fully initialized w/ defaults

* Taffy ~= Parent Classes

index.cfm required (but not used for anything - ala FW/1)
- I like to put notes to future maintainers and lots of curse words in here.

# Section 1:
# Mapping requests to data

## /api/resources/artCollection.cfc

```
<cfcomponent
   extends="taffy.core.resource"
   taffy:uri="/art">

      <cffunction name="get">
         <cfreturn representationOf(artQuery) />
      </cffunction>

</cfcomponent>
```

Create resources folder (convention)
add a cfc for the art collection

extends parent class (required)
uri (required)
convention function name
representationOf helper method

* default serialization to json

## /api/resources/artCollection.cfc

```
<cfcomponent
   extends="taffy.core.resource"
   taffy:uri="/art">

      <cffunction name="FooBar"
         taffy:verb="get">
            <cfreturn representationOf(artQuery) />
      </cffunction>

</cfcomponent>
```

optional metadata to map custom function name to specific http verb

## /api/resources/artCollection.cfc

```
<cfcomponent
   extends="taffy.core.resource"
   taffy:uri="/art">

      <cffunction name="FooBar"
         taffy:verb="get">
            <cfreturn representationOf(artQuery)
            .withStatus(200) />
      </cffunction>

</cfcomponent>
```

method chaining

specify status code

## /api/resources/artCollection.cfc

```
<cfcomponent
   extends="taffy.core.resource"
   taffy:uri="/art">

      <cffunction name="FooBar"
         taffy:verb="get">
            <cfreturn representationOf(artQuery)
            .withStatus(200)
            .withHeaders({foo="bar"}) />
      </cffunction>

</cfcomponent>
```

*ADD* headers
(can't remove headers)

* note that struct syntax is CF9 only, just pass in an existing structure object for older versions

## /api/resources/artMember.cfc

```
<cfcomponent
   extends="taffy.core.resource"
   taffy:uri="/art/{artId}">

      <cffunction name="get">
         <cfargument name="artId" />
         <cfreturn noData()
            .withStatus(404)
            .withHeaders({foo="bar"}) />
      </cffunction>

</cfcomponent>
```

...That was collection, this is a Member -- member has an ID, right?

In this case, the requested resource was not found

URI contains token
function has arguments named after tokens
must be one of: noData() or representationOf() -> still works with withStatus and withHeaders

## /api/resources/artMember.cfc

```
<cfcomponent
   extends="taffy.core.resource"
   taffy:uri="/art/{artId}">

      <cffunction name="get">
         <cfargument name="artId" />
         <cfreturn noData()
            .withStatus(404)
            .withHeaders({foo="bar"}) />
      </cffunction>

</cfcomponent>
```

...That was collection, this is a Member -- member has an ID, right?

In this case, the requested resource was not found

URI contains token
function has arguments named after tokens
must be one of: noData() or representationOf() -> still works with withStatus and withHeaders

# Section 2:
# Data Representations

That’s how you return data -- you don’t worry about serializing it or formatting at all, just return the raw data objects and headers that you want to be sent to the client.

Taffy will stuff it into an object that can serialize the data. That object is a representation class.

Default rep class built into framework -> Native JSON

## /api/resources/jsonRepresentation.cfc

```
<cfcomponent
   extends="taffy.core.baseRepresentation">

      <cffunction name="getAsJson"
         taffy:mime="application/json"
         taffy:default="true">
            <cfreturn serializeJson(variables.data) />
      </cffunction>

</cfcomponent>
```

Taffy comes with rep class that uses native JSON serialization
This is how you would re-implement it
(this is almost verbatim the actual code of the default class)

* Parent Class
* Convention method name
* Metadata for mime definition (required)
* Metadata to set default mime type (optional)
* variables.data is special (always named variables.data)

# /api/Application.cfc

```
<cfcomponent extends="taffy.core.api">
   <cfset this.name = "SomeApi" />

   <cffunction name="configureTaffy">
      <cfset setDefaultRepresentationClass("resources.jsonRepresentation") />
   </cffunction>

</cfcomponent>
```

configureTaffy method
setDefaultRepresentationClass method
can be cfc dot-notation path or ...

## /api/Application.cfc

```
<cfcomponent extends="taffy.core.api">
   <cfset this.name = "SomeApi" />

   <cffunction name="configureTaffy">
      <cfset setDefaultRepresentationClass("jsonRepresentation") />
   </cffunction>

</cfcomponent>
```

... beanId (if using any bean factory, including built in BF w/ resources folder)

# Section 3:
# Events & Configuration

# Section 3:
# Events & Configuration

# /api/Application.cfc

```
<cfcomponent extends="taffy.core.api">
   <cfset this.name = "SomeApi" />

   <cffunction name="configureTaffy">
      <cfset setDefaultRepresentationClass("jsonRepresentation") />
      <cfset setBeanFactory(application.beanFactory) />
      <cfset setDashboardKey("dashboard") />
      <cfset enableDashboard(true) />
      <cfset setDebugKey("debug") />
      <cfset setReloadKey("reinit") />
      <cfset setReloadPassword("true") />
   </cffunction>

</cfcomponent>
```

These are all inside configureTaffy()

# /api/Application.cfc

```
<cfcomponent extends="taffy.core.api">

   <!--- instead of onApplicationStart, use... --->
   <cffunction name="applicationStartEvent" returnType="void">
   </cffunction>

   <!--- instead of onRequestStart, use... --->
   <cffunction name="requestStartEvent" returnType="void">
   </cffunction>

   <cffunction name="onTaffyRequest">
      <cfreturn true />
   </cffunction>

</cfcomponent>
```

Event methods expose access to standard lifecycle events without overriding the methods (return void)

onTaffyRequest used for enforcing limitations (auth/apikey, rate limiting, payments, etc) - more when we talk about advanced Taffy APIs

# Section 4:
# Testing & Documenting

Inspired by test harness in Mach-ii REST preso I saw on CFMeetup, by Doug Smith

cmd+tab out to show taffy dashboard, mock client, generated documentation

# Advanced Taffy

- Require an API Key
- Multiple return formats
- Return Images or Binary Files
- ColdSpring integration
- Parent/Child Applications
- Rate limiting

cmd+tab out to show example API built for each point

# ColdSpring Integration

- Taffy has a simple bean factory built in
- Use setBeanFactory() to have Taffy use ColdSpring instead of or in addition to Taffy's own factory
  - Dependency injection for resources (in addition)
  - Control both resources and dependencies (instead)

How many people in the audience use a different DI framework? (Lightwire? What else?)

# Parent/Child Applications

- Share Application scope (this.name = same)
- Share physical files in different application contexts

Benefits of sharing application scope:
* Use of one application keeps the other alive
* Share ColdSpring instance to keep memory usage down

Benefits of different contexts:
* Easier to setup & understand
* Both apps will load faster (first time)

# REST anti-patterns

- Using GET for every request
- Everything returns 200
- Caching? Who does that?!
- Don’t link to resources
- Only offer a single data format

* Embrace verbs
* Learn the http status codes and use them
* Cache like a mofo (use existing solutions: ehcache, memcached)
* Instead of providing *just* entity id, provide the full URL you would run a PUT against to update it (kind of like hyperlinks)
* Offer multiple data formats

# Questions?

http://github.com/atuttle/Taffy/

Downloads, Source, Documentation, Bug DB

(or taffy.riaforge.org)

Google Group: http://tinyurl.com/taffy-groups