REST & AJAX
FRICTIONLESS FLOW OF INFORMATION

REST Web Services

- REST = **RE**presentational **S**tate **T**ransfer
- REST is an architectural style consisting of a coordinated set of architectural constraints
- First described in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine.
- RESTful is typically used to refer to web services implementing a REST architecture.
- Alternative to other distributed-computing specifications such as SOAP.
- Simple HTTP client/server mechanism to exchange data
- Everything – the UNIVERSE is available through a URI
- Utilizes HTTP: GET/POST/PUT/DELETE operations

Architectural Constraints

- Client–server
  - Separation of concerns. A uniform interface separates clients from servers.
- Stateless
  - The client–server communication is further constrained by no client context being stored on the server between requests.
- Cacheable
  - Basic WWW principle: clients can cache responses.
- Layered system
  - A client cannot necessarily tell whether it is connected directly to the end server, or to an intermediary along the way.
- Uniform interface
  - Individual resources are identified in requests, i.e.,using URIs in web-based REST systems.

RESTful compared to SOAP Services

RESTful

- No significant tools required to interact with the Web service
- Smaller learning curve
- Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
- Fast (no extensive processing required)
- Closer to other Web technologies in design philosophy
- RPC style – short burst messages
- Explosive growth in commercial end user applications

SOAP

- Language, platform, and transport independent (REST requires use of HTTP)
- Works well in distributed enterprise environments (REST assumes direct point-to-point comm.)
- Standardized
- Built-in error handling
- Provided document style to better represent domain model
- Strong enterprise adoption – B2B

RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as http://example.com /resources	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element URI, such as http://examples .com /resources/item17	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection

Spring MVC REST-style Controller

- Essentially means receive & send the content directly as the message body instead of structuring HTML pages.

We are NOT using HTML

We are using well-formed XML OR JSON

Spring support is based on the

@REQUESTBODY & @RESPONSEBODY annotations

@ResponseStatus(value = HttpStatus.***NO_CONTENT)***

For deletes, creates, updates…

RequestBody & ResponseBody

- @ResponseBody
  - Spring framework uses the "Accept" header of the request to decide the media type to send to the client
- @RequestBody
  - Spring framework will use the "*Content-Type*" header to determine the media type of the Request body received.

  - To get XML, MIME media type = "application/xml"
  - To get JSON, MIME media type = "application/json "

JSON (JavaScript Object Notation)

- {
- "productId":"P1235",
- "name":"Dell Inspiron",
- "unitPrice":700,
- "description":"Dell Inspiron 14-inch Laptop (Black) with 3rd Generation Intel Core processors",
- "manufacturer":"Dell",
- "category":"Laptop",
- "unitsInStock":1000,
- "unitsInOrder":0,
- "discontinued":false,
- "condition":null
- }

RESTful Web Service Controller

```
@RequestMapping(value = "/add/{productId}", method = RequestMethod.PUT)
@ResponseStatus(value = HttpStatus.NO_CONTENT)
 public void addItem(@PathVariable String productId,
                                          HttpServletRequest request) {
```

- `@RequestMapping("/showProduct“, method = RequestMethod.GET)`

```
public @ResponseBody Product getRestProduct(HttpServletRequestrequest){
```

RESTful input Validation

- How does THAT work?

- **XML - Schema validation is generally not a good idea in a REST service**.
- A major goal of REST is to decouple client and server so that they can evolve separately.

- **What about JSON validation/consistency?**
- API producers have frequently developed their own JSON response formats in the absence of well-defined standards.

ALTERNATIVE OPTION: JSR-303 Bean Validation

Main Point

- REST is defined by architectural constraints. It is able to access information through the ubiquitous URI. Everything on the web is available through a URI.

- *Everything in creation is known through understanding and experience of the Unified Field of Consciousness*

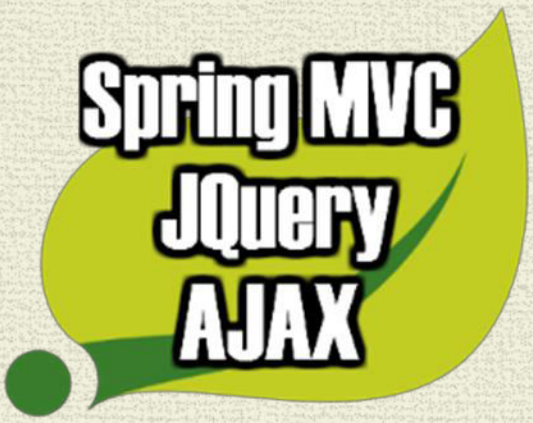
AJAX

- **A**synchronous **J**avascript **A**nd **X**ML
- Web applications are able to make quick, incremental updates to the client without reloading the entire browser page
- The use of XML is not required; JSON is often used instead (AJAJ)
- Examples :
  - * Validate values of form fields before saving
  - * Dynamically load dropdown values from database
  - * Load table data and paginations
  - * Live Search

***Ajax* -** a broad group of Web technologies that communicates with a server in the background, without interfering with the current state of the page.

Jquery

- **SLOGAN:**
- The Write Less, Do More, JavaScript Library.



- Fast, small, and feature-rich JavaScript library.

- HTML document traversal and manipulation, event handling, animation, and Ajax

Cart.js

- RESTful services:

- addToCart - Add Item
- removeFromCart - Remove Item
- showProduct - Show details of product in Cart

Cart.js addToCart Function

```
addToCart = function(productId){
$.ajax({
      url: '/webstore8/rest/cart/add/' + productId,
      type: 'PUT',
      dataType: "json",
      success: function(response){
             alert("Product Successfully added to the Cart!");
    },
      error: function(){
             alert('Error while request..');
        }
    });
}
```

Jquery AJAJ example

- `<script type="text/javascript" src="http://code.jquery.com/jquery-1.10.1.min.js"></script>`
- `<script type="text/javascript" src="<spring:url value="/resource/js/cart.js"/>"></script>`

Invoke the Javascript add item function

- `<a href="#" class="btn btn-warning btn-large" onclick="addToCart('${product.productId}')">`

Invoke the Javascript remove item function

- `<a href="#" class="label label-danger" onclick="removeFromCart('${item.value.product.productId}')">`

RequestBody & ResponseBody

- **Form Rountrip done with RESTful Web Service**

```
@RequestMapping(value = "/add",method = RequestMethod.POST)
public @ResponseBody Employee add(@RequestBody Employee
employee)  {
            SAVE Employee”... OR
                throw new EmployeeException
           return employee;
     }
```

```
 @ExceptionHandler(EmployeeException.class)
public @ResponseBody String handleError( EmployeeException
exception) {
           String message = exception.getFullMessage();
           return message;
```

RequestBody & ResponseBody

- **employee.jsp**
- <form id="employeeForm" name="employeeForm" method="post">

.....

<input type="button" value="Ajax Submit“

onclick="makeAjaxCall();">

AJAX CALL:

```
function makeAjaxCall(){
    var send =
        JSON.stringify(serializeObject($('#employeeForm')));
$.ajax({
        url: '/RestEmployee/employee/add',
        type: 'POST',
        dataType: "json",
        data:send,
        contentType: 'application/json',
        success: function(employee){
```

Welcome Student Example

```
@RequestMapping(value = "/welcomeStudent", method = RequestMethod.GET)
    public @ResponseBody String[] displayWelcome( ) {
            .  .. .. ..
        String [] result = {
            String.valueOf(totalCells),
            String.valueOf(cellCounter),
            "Welcome to " + currentStudent
        };
      return result;
```

To Kick off AJAX: setInterval(welcome, 750);

```
function welcome() {
        $.ajax({
            url : 'welcomeStudent',
            success : function(data) {
            total =  parseInt(data[0]);
            counter =  parseInt(data[1]);
                $('.welcome').html(data[2]);
            duke();
```

Main Point

Ajax uses Javascript in a browser to access data or functionality residing on a server and then quickly and efficiently selectively update the browser.

Nature is maximally efficient.

How to set FireFox to Accept JSON

- In the address bar in FireFox, type “about:config”.
- Click the “I promise” button.
- In the search bar type “network.http.accept.default”.
- Double-click the setting.
- In the comma-separated list add a new value of “application/json” (without the quotes) .
- Use“application/json;q=0.9″ to lower the precedence compared to HTML.

Javascript [MVC] frameworks

- All the technologies follow from the view that serious JavaScript applications require proper data models and ability to do client-side rendering, not just server rendering plus some Ajax and jQuery code.
- Quote from Jeremy Ashkenas, the Backbone creator: "*At this point, saying 'single-page application' is like saying 'horseless carriage'*" (i.e., it's not even a novelty any more).

http://creativitykills.co/

If you're writing an application that will likely only be communicating with an API or back-end data service, where much of the heavy lifting for viewing or manipulating that data will be occurring in the browser, you may find a JavaScript MV* framework useful.

If, however, you're building an application that still relies on the server for most of the heavy-lifting of Views/pages and you're just using a little JavaScript or jQuery to make things a little more interactive, an MV framework may be overkill. There certainly are complex Web applications where the partial rendering of views can*