# REST, RESTful and JAX-RS



jfernan@ua.pt

The Business of IT Blog

# REST vs CRUD: What's The Difference?

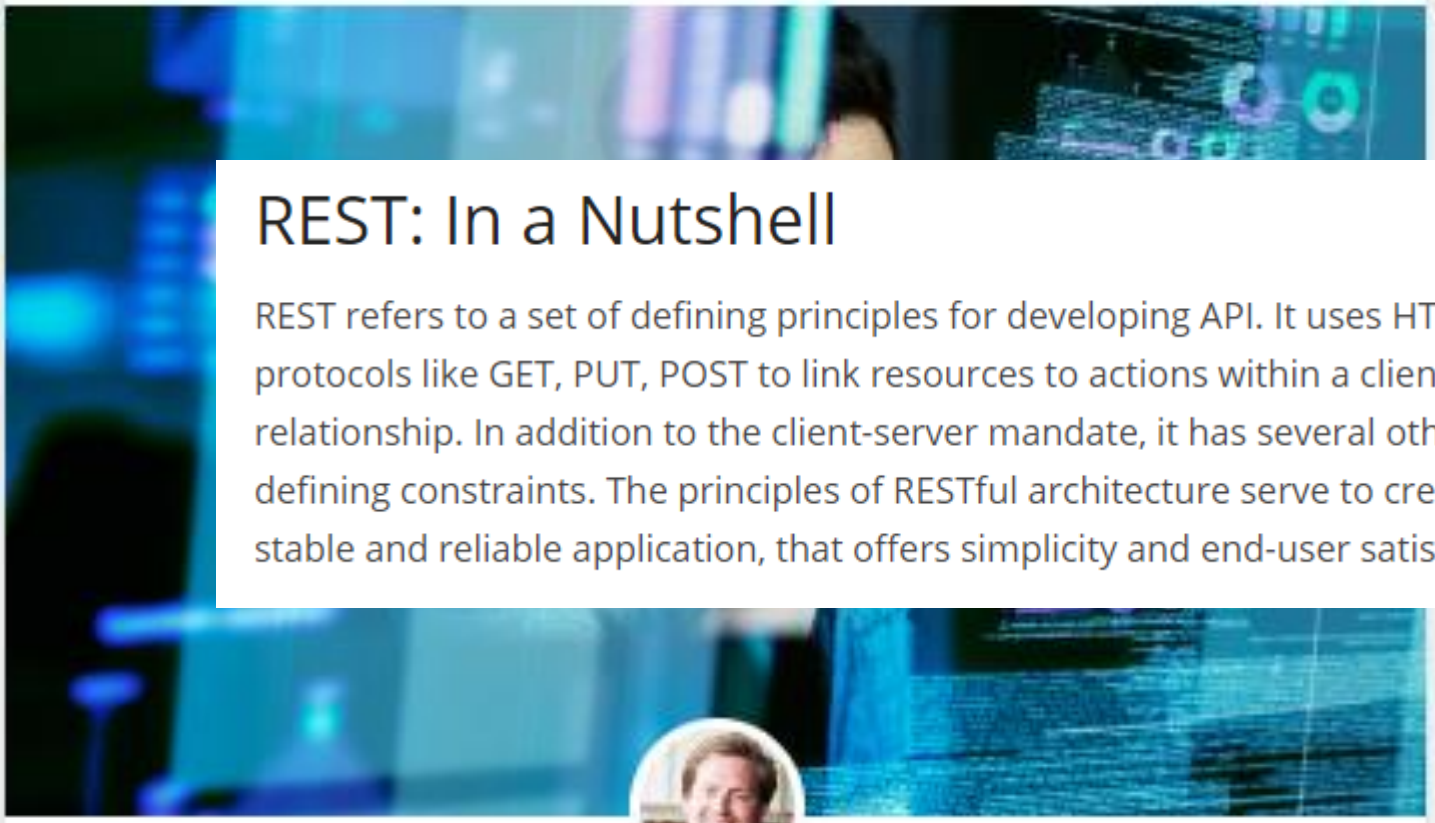


January 18, 2018 · by Stephen Watts

REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

The Business of IT Blog

# REST vs CRUD: What's The Difference?



## REST: In a Nutshell

REST refers to a set of defining principles for developing API. It uses HTTP protocols like GET, PUT, POST to link resources to actions within a client-server relationship. In addition to the client-server mandate, it has several other defining constraints. The principles of RESTful architecture serve to create a stable and reliable application, that offers simplicity and end-user satisfaction.



January 18, 2018 · by Stephen Watts

REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

The Business of IT Blog

# REST vs CRUD: What's The Difference?

- REST is an architectural system centered around resources and hypermedia, via HTTP protocols
- CRUD is a cycle meant for maintaining permanent records in a database setting
- CRUD principles are mapped to REST commands to comply with the goals of RESTful architecture

In REST:

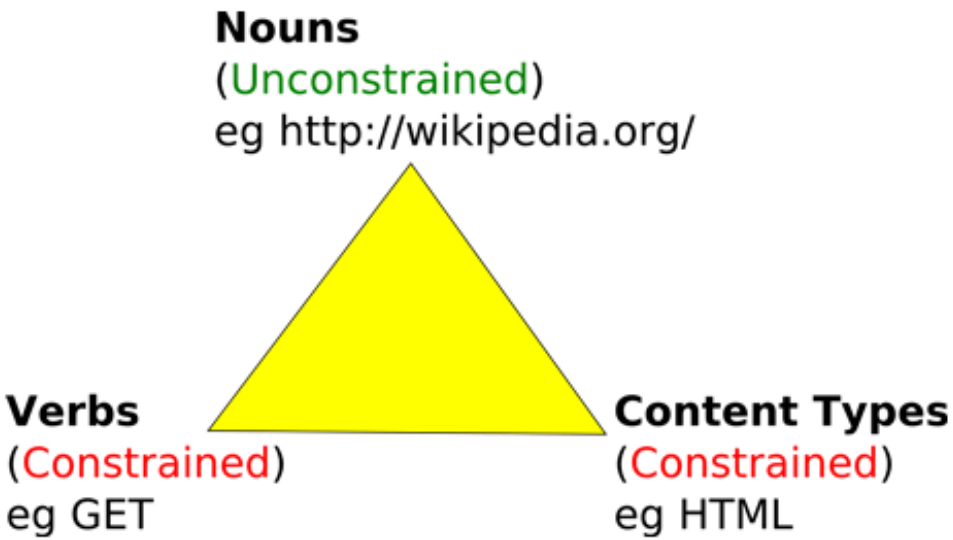
- Representations must be uniform with regard to resources
- Hypermedia represents relationships between resources
- Only one entry into an API to create one self-contained interface, then hyperlink to create relationships

January 18, 2018 · by Stephen Watts

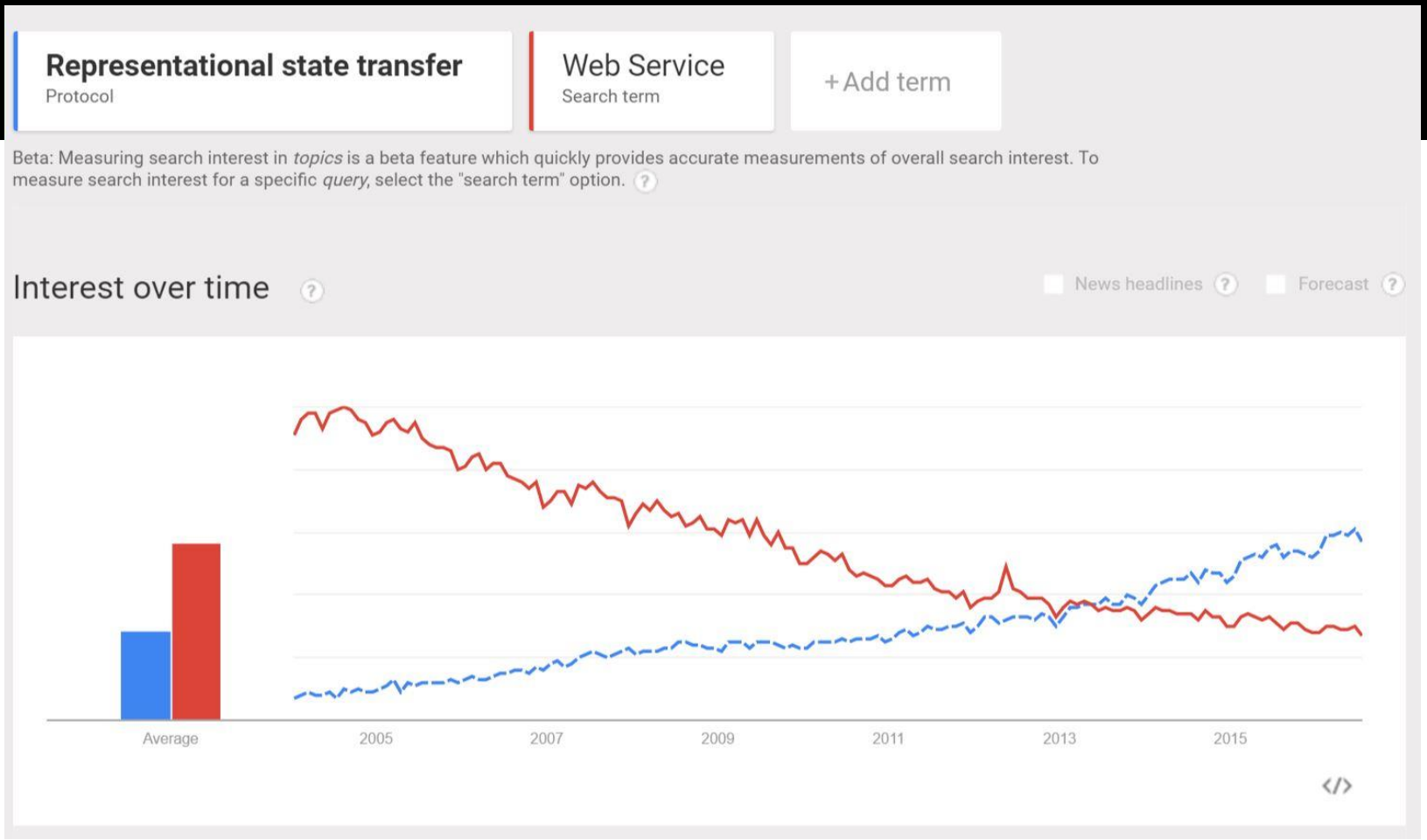
REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

# REST

- Noun - Resources
  - Published under a unified reference
  - Referentiable
- Verb - Operation
  - Standard HTTP:
    - GET,PUT,POST, DELETE
  - Uniform semantic
- Contents – format
  - Semantics on the programmer




"define some resources and they will have these methods".




REST API vs. SOAP Web Services Management
https://dzone.com/articles/rest-api-vs-soap-web-services-management

**Representational state transfer**
Protocol

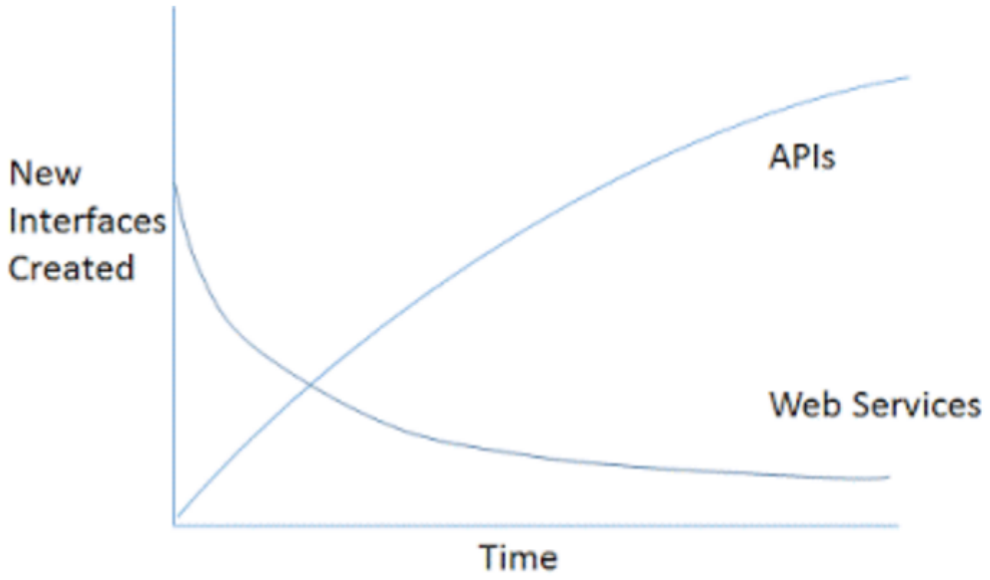
Web Service
Search term

+ Add term

Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the "search term" option.

Interest over ti

Average




**I'm convinced that we will see fewer and fewer web services used internally and externally, but they aren't dead just yet. I expect standards to play a bigger part in API management but its emphasis on simplicity and complexity hiding will stop standards being API's death knell.**

BIAS?

REST API vs. SOAP Web Services Management
https://dzone.com/articles/rest-api-vs-soap-web-services-management

# REST and SOAP

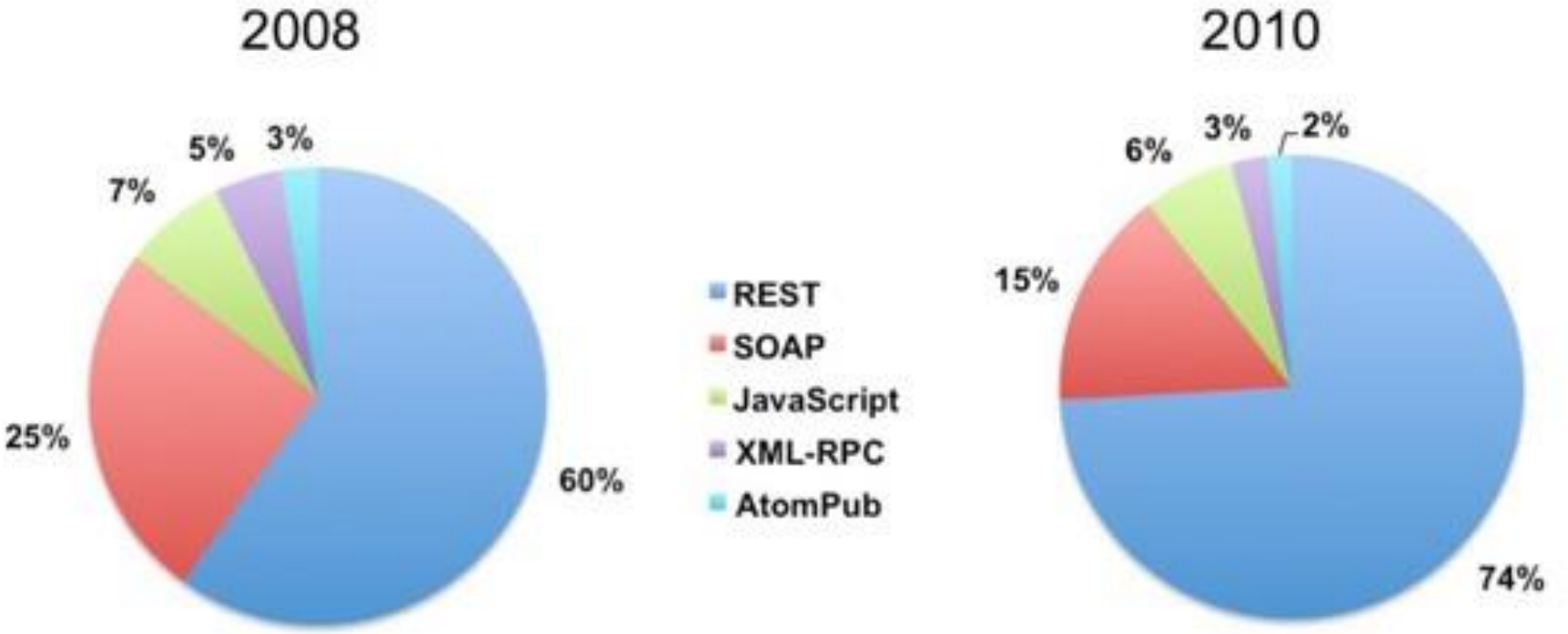
# JAX-RS

- Relies on servlets
- **RESTful services based on REST**
- Annotations allow automation
- Conversion
  - Handling requests
  - …
- Jersey is java official implementation…

http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

# JAX-RS rellies on http...

- Two major standard with reference implementations in java




http://blog.nicolashachet.com/wp-content/uploads/2012/06/rest-vs-soap.jpg

# JAX-RS and annotations

- Annotations allow automation
  - Conversion
  - Handling requests
  - …

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI**: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery. See The `@Path` Annotation and URI Path Templates for more information.
- **Uniform interface**: Resources are manipulated using a fixed set of four create, read, update, delete operations: `PUT`, `GET`, `POST`, and `DELETE`. `PUT` creates a new resource, which can be then deleted by using `DELETE`. `GET` retrieves the current state of a resource in some representation. `POST` transfers a new state onto a resource. See Responding to HTTP Methods and Requests for more information.
- **Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control. See Responding to HTTP Methods and Requests and Using Entity Providers to Map HTTP Response and Request Entity Bodies for more information.
- **Stateful interactions through hyperlinks**: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction. See Using Entity Providers to Map HTTP Response and Request Entity Bodies and "Building URIs" in the JAX-RS Overview document for more information.

http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

# A simple example

```
package com.mkyong.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.Response;

@Path("/hello")
public class HelloWorldService {

        @GET
        @Path("/{param}")
        public Response getMsg(@PathParam("param") String msg) {

                String output = "Jersey say : " + msg;

                return Response.status(200).entity(output).build();

        }

}
```

http://www.mkyong.com/webservices/jax-rs/jersey-hello-world-example/

# A simple example

```java
package com.mkyong.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.Response;

@Path("/hello")
public class HelloWorldService {

	@GET
	@Path("/{param}")
	public Response getMsg(@PathParam("param") String msg) {

		String output = "Jersey say : " + msg;

		return Response.status(200).entity(output).build();

	}

}
```

The resource path

The relative path and operation

http://www.mkyong.com/webservices/jax-rs/jersey-hello-world-example/

# JAX-RS relies on servlets

```
<web-app id="WebApp_ID" version="2.4"
        xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
        <display-name>Restful Web Application</display-name>

        <servlet>
                <servlet-name>jersey-serlvet</servlet-name>
                <servlet-class>
                     com.sun.jersey.spi.container.servlet.ServletContainer
                </servlet-class>
                <init-param>
                     <param-name>com.sun.jersey.config.property.packages</param-name>
                     <param-value>com.mkyong.rest</param-value>
                </init-param>
                <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet-mapping>
                <servlet-name>jersey-serlvet</servlet-name>
                <url-pattern>/rest/*</url-pattern>
        </servlet-mapping>

</web-app>
```

http://www.mkyong.com/webservices/jax-rs/jersey-hello-world-example/

# JAX-RS relies on servlets

```
<web-app id="WebApp_ID" version="2.4"
        xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
        <display-name>Restful Web Application</display-name>

    <servlet>
            <servlet-name>jersey-serlvet</servlet-name>
            <servlet-class>
                com.sun.jersey.spi.container.servlet.ServletContainer
            </servlet-class>
            <init-param>
                <param-name>com.sun.jersey.config.property.packages</param-name>
                <param-value>com.mkyong.rest</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
            <servlet-name>jersey-serlvet</servlet-name>
            <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>

</web-app>
```

http://www.mkyong.com/webservices/jax-rs/jersey-hello-world-example/

# JAX-RS relies on annotations

**Table 20-1 Summary of JAX-RS Annotations**

| Annotation | Description |
|---|---|
| `@Path` | The `@Path` annotation's value is a relative URI path indicating where the Java class will be hosted: for example, `/helloworld`. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: `/helloworld/{username}`. |
| `@GET` | The `@GET` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| `@POST` | The `@POST` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| `@PUT` | The `@PUT` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| `@DELETE` | The `@DELETE` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| `@HEAD` | The `@HEAD` annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP HEAD requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| `@PathParam` | The `@PathParam` annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the `@Path` class-level annotation. |
| `@QueryParam` | The `@QueryParam` annotation is a type of parameter that you can extract for use in your resource class. Query parameters are extracted from the request URI query parameters. |
| `@Consumes` | The `@Consumes` annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client. |
| `@Produces` | The `@Produces` annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, `"text/plain"`. |
| `@Provider` | The `@Provider` annotation is used for anything that is of interest to the JAX-RS runtime, such as `MessageBodyReader` and `MessageBodyWriter`. For HTTP requests, the `MessageBodyReader` is used to map an HTTP request entity body to method parameters. On the response side, a return value is mapped to an HTTP response entity body by using a `MessageBodyWriter`. If the application needs to supply additional metadata, such as HTTP headers or a different status code, a method can return a `Response` that wraps the entity and that can be built using `Response.ResponseBuilder`. |

# An example

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

# An example: the path

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

# URI, paths, substitutions

- name1: james
- name2: gatz
- name3:
- location: Main%20Street
- question: why

**Table 20-2 Examples of URI Path Templates**

| URI Path Template | URI After Substitution |
|---|---|
| http://example.com/{name1}/{name2}/ | http://example.com/james/gatz/ |
| http://example.com/{question}/{question}/{question}/ | http://example.com/why/why/why/ |
| http://example.com/maps/{location} | http://example.com/maps/Main%20Street |
| http://example.com/{name3}/home/ | http://example.com//home/ |

# An example: http request

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

# An example: will produce…

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

# Types produced and consumed

**Table 20-3 Types Supported for HTTP Request and Response Entity Bodies**

| Java Type | Supported Media Types |
|---|---|
| `byte[]` | All media types (`*/*`) |
| `java.lang.String` | All text media types (`text/*`) |
| `java.io.InputStream` | All media types (`*/*`) |
| `java.io.Reader` | All media types (`*/*`) |
| `java.io.File` | All media types (`*/*`) |
| `javax.activation.DataSource` | All media types (`*/*`) |
| `javax.xml.transform.Source` | XML media types (`text/xml`, `application/xml`, and `application/*+xml`) |
| `javax.xml.bind.JAXBElement` and application-supplied JAXB classes | XML media types (`text/xml`, `application/xml`, and `application/*+xml`) |
| `MultivaluedMap<String, String>` | Form content (`application/x-www-form-urlencoded`) |
| `StreamingOutput` | All media types (`*/*`), `MessageBodyWriter` only |

The following example shows how to use `MessageBodyReader` with the `@Consumes` and `@Provider` annotations:

```
@Consumes("application/x-www-form-urlencoded")
@Provider
public class FormReader implements MessageBodyReader<NameValuePair> {
```

The following example shows how to use `MessageBodyWriter` with the `@Produces` and `@Provider` annotations:

```
@Produces("text/html")
@Provider
public class FormWriter implements
        MessageBodyWriter<Hashtable<String, String>> {
```

# An example: … a message

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

# Java-RS relies on web-server

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

# Java-RS relies on web-server

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
publi
    /
    (
    /
    /
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

In this example, the URL pattern for the JAX-RS helper servlet, specified in `web.xml`, is the d

```
<servlet-mapping>
      <servlet-name>My JAX-RS Resource</servlet-name>
      <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
```

# Java-RS is exposed in …/resources/*

```
package com.sun.jersey.samples.helloworld.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
publi
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

In this example, the URL pattern for the JAX-RS helper servlet, specified in `web.xml`, is the d

```
<servlet-mapping>
      <servlet-name>My JAX-RS Resource</servlet-name>
      <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
```

# In JavaEE Jax-RS is support on EJB

- Webservices and REST
  - are coded as stateless EJB (at least in Glassfish)
  - With some extra annotations
- EJB are not webservices
  - EJB uses IIOP – binary, RMI based interface
  - Webservices uses SOAP – text, http-based
  - REST uses  XML/JSON – text, http-based
- The application server manages the different "facets" and pipes the data

# Session bean as a RESTfull

```
package
    com.abien.patterns.business.rest.boundary
    ;

import
    com.abien.patterns.business.rest.control.
    Service;
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Stateless
@Path("current")
public class ServiceFacade {

    @EJB
    Service service;

    @GET
    public String getDate(){
        return
    service.getCurrentDate().toString();
    }

}
```

```
@Stateless
public class Service {

    public Date getCurrentDate(){
        return new Date();
    }
}
```

Accessible from
http://<your app server>:<port>/<app>/ < ....>/**current**

http://www.adam-bien.com/roller/abien/entry/ejb_3_1_and_rest

# JAX-RS to implement REST

# Root resource classes

- Root resource classes are POJOs that are either annotated with @Path or have at least one method annotated with @Path or a request method designator, such as @GET, @PUT, @POST, or @DELETE.

```
@Path("/books")
public class BookController {
  private BookService bookService;

  public BookController(BookService bookService) {
    this.bookService = bookService;
  }

  @GET
  @Produces("application/json")
  public Collection getAllBooks() {
    return bookService.getAllBooks();
  }

  @GET
  @Produces("application/json")
  @Path("/{oid}")
  public Book getBook(@PathParam("oid") String oid) {
    return bookService.getBook(oid);
  }
```

https://www.gauravbytes.com/2017/02/spring-boot-restful-webservices-with.html ~

# Sample REST example: path

```
@Path("/orders/")
@Interceptors(CallAudit.class)
@Stateless
public class OrderService {

    @EJB BillingService billing;
    @EJB DeliveryService delivery;
    @EJB Warehouse warehouse;

    @PUT
    @Produces({"application/xml","application/json"})
    @Consumes({"application/xml","application/json"})
    public Order order(Order newOrder){
        Order order = warehouse.checkout(newOrder);
        billing.payForOrder(order);
        delivery.deliver(order);
        return order;
    }

    @GET
    @Path("{orderid}/")
    @Produces({"application/xml","application/json"})
    public Order status(@PathParam("orderid") long
     orderId){
       return delivery.status(orderId);
    }
}
```

```
@Stateless
public class DeliveryService {

    @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
        System.out.println("Delivered: " + order);
        order.setDelivered(true);
    }


    public Order status(long orderId) {
        Order found = this.em.find(Order.class,
     orderId);
        if(found == null)
            found = new Order();
        return found;
    }
}
```

http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_13

# Sample REST example: lifecycle

```
@Path("/orders/")
@Interceptors(CallAudit.class)
@Stateless
public class OrderService {

    @EJB BillingService billing;
    @EJB DeliveryService delivery;
    @EJB Warehouse warehouse;

    @PUT
    @Produces({"application/xml","application/json"})
    @Consumes({"application/xml","application/json"})
    public Order order(Order newOrder){
        Order order = warehouse.checkout(newOrder);
        billing.payForOrder(order);
        delivery.deliver(order);
        return order;
    }

    @GET
    @Path("{orderid}/")
    @Produces({"application/xml","application/json"})
    public Order status(@PathParam("orderid") long
     orderId){
       return delivery.status(orderId);
    }
}
```

```
@Stateless
public class DeliveryService {

    @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
        System.out.println("Delivered: " + order);
        order.setDelivered(true);
    }


    public Order status(long orderId) {
        Order found = this.em.find(Order.class,
     orderId);
        if(found == null)
            found = new Order();
        return found;
    }
}
```

http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_13

# Sample REST example: an EJB

```
@Path("/orders/")
@Interceptors(CallAudit.class)
@Stateless
public class OrderService {

    @EJB BillingService billing;
    @EJB DeliveryService delivery;
    @EJB Warehouse warehouse;

    @PUT
    @Produces({"application/xml","application/json"})
    @Consumes({"application/xml","application/json"})
    public Order order(Order newOrder){
        Order order = warehouse.checkout(newOrder);
        billing.payForOrder(order);
        delivery.deliver(order);
        return order;
    }

    @GET
    @Path("{orderid}/")
    @Produces({"application/xml","application/json"})
    public Order status(@PathParam("orderid") long
     orderId){
       return delivery.status(orderId);
    }
}
```

```
@Stateless
public class DeliveryService {

    @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
        System.out.println("Delivered: " + order);
        order.setDelivered(true);
    }


    public Order status(long orderId) {
        Order found = this.em.find(Order.class,
     orderId);
        if(found == null)
            found = new Order();
        return found;
    }
}
```

http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_13

# Sample REST example: using EJB

```
@Path("/orders/")
@Interceptors(CallAudit.class)
@Stateless
public class OrderService {

    @EJB BillingService billing;
    @EJB DeliveryService delivery;
    @EJB Warehouse warehouse;

    @PUT
    @Produces({"application/xml","application/json"})
    @Consumes({"application/xml","application/json"})
    public Order order(Order newOrder){
        Order order = warehouse.checkout(newOrder);
        billing.payForOrder(order);
        delivery.deliver(order);
        return order;
    }

    @GET
    @Path("{orderid}/")
    @Produces({"application/xml","application/json"})
    public Order status(@PathParam("orderid") long
     orderId){
       return delivery.status(orderId);
    }
}
```

```
@Stateless
public class DeliveryService {

    @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
        System.out.println("Delivered: " + order);
        order.setDelivered(true);
    }


    public Order status(long orderId) {
        Order found = this.em.find(Order.class,
     orderId);
        if(found == null)
            found = new Order();
        return found;
    }
}
```

http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_13

# Sample REST example: http response

```
@Path("/orders/")
@Interceptors(CallAudit.class)
@Stateless
public class OrderService {

    @EJB BillingService billing;
    @EJB DeliveryService delivery;
    @EJB Warehouse warehouse;

    @PUT
    @Produces({"application/xml","application/json"})
    @Consumes({"application/xml","application/json"})
    public Order order(Order newOrder){
        Order order = warehouse.checkout(newOrder);
        billing.payForOrder(order);
        delivery.deliver(order);
        return order;
    }

    @GET
    @Path("{orderid}/")
    @Produces({"application/xml","application/json"})
    public Order status(@PathParam("orderid") long
     orderId){
       return delivery.status(orderId);
    }
}
```

```
@Stateless
public class DeliveryService {

    @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
        System.out.println("Delivered: " + order);
        order.setDelivered(true);
    }


    public Order status(long orderId) {
        Order found = this.em.find(Order.class,
     orderId);
        if(found == null)
            found = new Order();
        return found;
    }
}
```

http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_13

# Sample REST example: using JPA

```
@Path("/orders/")
@Interceptors(CallAudit.class)
@Stateless
public class OrderService {

    @EJB BillingService billing;
    @EJB DeliveryService delivery;
    @EJB Warehouse warehouse;

    @PUT
    @Produces({"application/xml","application/json"})
    @Consumes({"application/xml","application/json"})
    public Order order(Order newOrder){
        Order order = warehouse.checkout(newOrder);
        billing.payForOrder(order);
        delivery.deliver(order);
        return order;
    }

    @GET
    @Path("{orderid}/")
    @Produces({"application/xml","application/json"})
    public Order status(@PathParam("orderid") long
     orderId){
       return delivery.status(orderId);
    }
}
```

```
@Stateless
public class DeliveryService {

    @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
        System.out.println("Delivered: " + order);
        order.setDelivered(true);
    }


    public Order status(long orderId) {
        Order found = this.em.find(Order.class,
     orderId);
        if(found == null)
            found = new Order();
        return found;
    }
}
```

http://www.adam-bien.com/roller/abien/entry/simplest_possible_ejb_3_13

# Take home

- JAX-RS
  - provides a way of implementing RESTful
  - can be used to implement REST

- REST != Restful
  - REST is not only URIs and HTTP
  - RESTful is a HTTP based API using URIs
  - In real life usually you may find hybrids

# Links and references

- Comprehensive list on tutorials and documentation
  - Servlets, JSON and XML
    - http://goo.gl/4dldbq
  - JAX-WS and webservices
    - http://goo.gl/T9NL1u
  - JAX-RS and REST
    - http://goo.gl/UMggJ7

# The END