RESTful Webservices

mthree

Web Service

WileyNXT

The services that can be accessed over network are called web services.

The differences between web service and web application:

- Web applications are meant for users and to be accessed in browser having human readable format whereas web services are meant for applications to access data in the format of XML, JSON etc
- Web applications always use HTTP/HTTPS protocol whereas traditional web services use SOAP protocol. Recently REST is getting popularity that is an architecture style and almost all times run on HTTP/HTTPS protocol.
- Web applications are not meant for reusability whereas this is one of the benefit of web services. A single web service can be used by different kinds of applications.

- Web application can access web services to access some data or to perform some tasks, web services can’t access web applications to fetch some data.
- Web applications are capable to maintain user session, web services are stateless.

Types of Web Service

- SOAP: SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, it's platform and language independent. So our server can be based on JAVA and client can be on .NET, PHP etc. and vice versa.
- REST: REST is an architectural style for developing web services. It's getting popularity recently because it has small learning curve when compared to SOAP. Resources are core concepts of Restful web services and they are uniquely identified by their URIs.

Java Web Services

Java provides it's own API to create both SOAP as well as REST web services.

- JAX-WS: JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to build web services server and client application.
- JAX-RS: Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services.

RESTful Principles

- **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. E.g. @Path Annotation and URI Path Templates
- **Uniform interface:** Resources are manipulated using a fixed set of four create, read, update, delete operations such as 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.

- **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.
- **Stateful interactions through hyperlinks:** Every interaction with a resource is stateless. i.e. 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.

Rules to define REST API

- **Rule 1:** A trailing forward slash (/) should not be included in URIs.

Important rules to follow as the last character within a URI's path, a forward slash (/) adds no semantic value and may cause confusion. REST API's should not expect a trailing slash and should not include them in the links that they provide to clients.

Many web components and frameworks will treat the following two URIs equally:

http://myapplication.com/data

http://myapplication.com/data/

However, every character within a URI counts toward a resource's unique identity.

Rule 2: Forward slash separator (/) must be used to indicate a hierarchical relationship. The forward slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources.

Rule 3: Hyphens (-) should be used to improve the readability of URIs.

To make your URIs easy for people to scan and interpret, use the hyphen (-) character to improve the readability of names in long path segments. Anywhere you would use a space or hyphen in English, you should use a hyphen in a URI.

http://myapplication.com/products/all-categories/highest-priced-products

- **Rule 4:** Underscores (_) should not be used in URIs. Text viewer applications (browsers, editors, etc.) often underline URIs to provide a visual cue that they are clickable. Depending on the application's font, the underscore (_) character can either get partially obscured or completely hidden by this underlining. To avoid this confusion, use hyphens (-) instead of underscores.
- **Rule 5:** Lowercase letters should be preferred in URI paths.

When convenient, lowercase letters are preferred in URI paths since capital letters can sometimes cause problems. RFC 3986 defines URIs as case-sensitive except for the scheme and host components.

http://myapplication.com/products/all-products

HTTP://API.EXAMPLE.COM/my-folder/my-doc

Rule 6: File extensions should not be included in URIs.

On the Web, the period (.) character is commonly used to separate the file name and extension portions of a URI. A REST API should not include artificial file extensions in URIs to indicate the format of a message's entity body. Instead, they should rely on the media type, as communicated through the Content-Type header, to determine how to process the body's content.

http://api.college.com/students/3248234/courses/2005/fall.json
http://api.college.com/students/3248234/courses/2005/fall

Rule 7: Should the endpoint name be singular or plural?

The keep-it-simple rule applies here. Although your inner grammatician will tell you it's wrong to describe a single instance of a resource using a plural, the pragmatic answer is to keep the URI format consistent and always use a plural.

URI	description
http://api.college.com/students	Retrieves a list of all students
http://api.college.com/students/3248234	Retrieves a student with id as 3248234
http://api.college.com/students/3248234/courses	Retrieves a list of all courses that are learned by a student with id 3248234
http://api.college.com/students/3248234/courses/java	Retrieves course Java for a student with id 3248234

REST methods

RESTful APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations. REST guidelines suggest using a specific HTTP method on a specific type of call made to the server

- **GET :** Use GET requests **to retrieve resource representation/information only** and not to modify it in any way.

A. GET requests do not change the state of the resource, these are said to be **safe methods**.

B. GET APIs should be **idempotent**, which means that making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

C. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

D. For any given HTTP GET API, if the resource is found on the server then it must return HTTP response code 200 (OK) – along with response body which is usually either XML or JSON content

E. In case resource is NOT found on server then it must return HTTP response code 404 (NOT FOUND)

http://www.appdomain.com/users

http://www.appdomain.com/users/123

http://www.appdomain.com/users/123/address

HTTP POST

Use POST APIs **to create new subordinate resources.** In terms of REST, POST methods are used to create a new resource into the collection of resources.

- If a resource has been created on the origin server, the response **SHOULD be HTTP** response code 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header.
- the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either HTTP response code 200 (OK) or 204 (No Content) is the appropriate response status.
- Responses to this method are **not cacheable**, unless the response includes appropriate Cache-Control or Expires header fields.

http://www.appdomain.com/users

http://www.appdomain.com/users/123

HTTP PUT

Use PUT APIs primarily to update existing resource.

- If the resource does not exist then API may decide to create a new resource or not.
- If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code 201 (Created) response and if an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

  http://www.appdomain.com/users

  http://www.appdomain.com/users/123

HTTP DELETE

DELETE APIs are used to delete resources

- A successful response of DELETE requests SHOULD be HTTP response code 200 (OK)
- if the response includes an entity describing the status, 202 (Accepted)
- if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.
- DELETE operations are idempotent. If you DELETE a resource, it's removed from the collection of resource.

- Repeatedly calling DELETE API on that resource will not change the outcome , however calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed.
- If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable

HTTP PATCH

HTTP PATCH requests are **to make partial update on a resource**.

- If you see PUT requests also modify a resource entity so to make more clear PATCH method is the correct choice for partially updating an existing resource and PUT should only be used if you're replacing a resource in its entirety.
- Support for PATCH in browsers, servers, and web application frameworks is not universal.
- PATCH method is not a replacement for the POST or PUT methods. It applies a delta (diff) rather than replacing the entire resource.

Spring Supports RESTful

- In Spring MVC, a controller can handle requests for all HTTP methods, which is a backbone of RESTful web services. @RequestMapping, @GetMapping, @PostMapping, etc
- In case of REST, the representation of data is very important and that's why Spring MVC allows you to bypass View-based rendering altogether by using the @ResponseBody annotation and various HttpMessgeConverter implementations.
- The Spring 4.0 release added a dedicated annotation **@RestController** to make the development of RESTful web services even easier. If you annotate your controller class using @RestController instead of @Controller then Spring applied message conversations to all handler methods in the controller.

Spring Supports RESTful

- One of the main difference between REST web services and a normal web application is that REST pass resource identifier data in URI itself e.g. /messages/101 while web application normally uses a query parameter e.g. /messages?Id=101.
  If you remember, we use @RequestParam to get the value of those query parameter but not to worry, Spring MVC also provides a @PathVariable annotation which can extract data from URL. It allows the controller to handle requests for parameterized URLs.
- Another key aspect of RESTful web services is Representation e.g. the same resource can be represented in different formats e.g. JSON, XML, HTML etc. Thankfully Spring provides several view implementations and views resolvers to render data as JSON, XML, and HTML.

  For example, ContentNegotiatingViewResolver can look at the file

Spring Supports RESTful

- Another key aspect of RESTful web services is Representation e.g. the same resource can be represented in different formats e.g. JSON, XML, HTML etc. Thankfully Spring provides several view implementations and views resolvers to render data as JSON, XML, and HTML.

  E.g. ContentNegotiatingViewResolver can look at the file extension of requests or Accept header to find out the correct representation of a resource for the client.
- Similar to @ResponseBody annotation, which is used for converting the response to the format client wants (by using HttpMessageConverts), Spring MVC also provides @RequestBody annotation, which uses HTtpMethodConverter implementations to convert inbound HTTP data into Java objects passed into a controller's handler method.

Spring Supports RESTful

- Spring Framework also provides a Template class, RestTemplate, similar to JdbcTemplate, and JmsTemplate, which can consume REST resources. You can use this class to test your RESTful web service or develop REST clients.