Tutorial Rest

v. 0.1.9-SNAPSHOT

Project Documentation

UniKnow **2015-05-13**

Table Of Content

1 Introduction

RESTful Web services

Within this tutorial we will create a simple restful blogging system.

When providing services in a specific technology (jax-ws, jax-rs, …), it's wise to avoid mixing the business logic of your service and the technology you are using to expose it to the outside world. This separation facilitates easier adaption of newer technologies and makes it easier to maintain and test services.

The project is separated in 2 packages.

1. api - defining the interfaces that are exposed
2. impl - Implementation of the interfaces

The `api` package contains the business interface `BlogService` within the package `org.uniknow.agiledev.tutorial.rest.api`, and the technology specific interface `BlogRestService` within the ' `jaxrs` package containing JAX-RS annotations. The `jaxrs` package also contains the DTO (Data Transfer Objects) classes required to provide and/or consume the REST services.

1.1 Versioning

Versioning of the REST service will be done via custom `Accept` headers containing custom media types (such as `application/vnd.agiledev.blog.v1+xml`) instead of using one of the defined media types. Within the `BlogRestService` the custom consume and produce media types are defined

```
/**
 * Version 1 of Blog Rest Service
 */
package org.uniknow.agiledev.tutorial.rest.api.jaxrs.V1;

@Path("/blog")
@Consumes({ "application/agiledev.blog.v1+xml",
        "application/agiledev.blog.v1+json" })
@Produces({ "application/agiledev.blog.v1+xml",
        "application/agiledev.blog.v1+json" })
public interface BlogRestService {
    ...
}
```

When the entities change in a non compatible way (V2), a new version of the API can co-exist with the first version, given that the entity/DTO namespace is also changed. The newer API will have its own interface and use an updated media type:

```
/**
 * Version 2 of Blog Rest Service
 */
package org.uniknow.agiledev.tutorial.rest.api.jaxrs.V2;

@Path("/blog")
@Consumes({"application/agiledev.blog.v2+xml", "application/agiledev.blog.v2+json"
@Produces({"application/agiledev.blog.v2+xml", "application/agiledev.blog.v2+json"
```

and the namespace would become something like this (`package-info.java`):

```
@XmlSchema(namespace = "http://www.uniknow.org/rest/blog/api/v2",
    elementFormDefault = XmlNsForm.QUALIFIED, xmlns = { @XmlNs(prefix = "api",
        namespaceURI = "http://www.uniknow.org/rest/blog/api/v2") })
package org.uniknow.agiledev.tutorial.rest.api.jaxrs.V2;

import javax.xml.bind.annotation.*;
```

Now the client is required to specify an `Accept` header when using the API. Whe can remedy this by adding default media types to the latest version of our API. This way clients invorking the API without an `Accept` header will be working with the latest version.

@Path(“/blog”) @Consumes({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON, “application/agiledev.blog.v2+xml”, “application/agiledev.blog.v2+json” }) @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON, “application/agiledev.blog.v2+xml”, “application/agiledev.blog.v2+json” })

1.2 Running the example application

To build and run the application perform:

```
mvn clean install cargo:run
```

Once the application is running the Rest services can be invoked by using for example the URL http://localhost:8080/rest/api/blog/posts and include header param `Accept` with for example value `application/agiledev.blog.v2+json`. If used without `Accept` header the response will be of content type `application/xml`.

Resources:

- http://codebias.blogspot.cz/2013/03/how-to-restful-web-services-with-jax-rs.html
- http://codebias.blogspot.nl/2014/03/versioning-rest-apis-with-custom-accept.html
- http://esofthead.com/restful-application-spring-resteasy/

2 Best practices REST API

Best Practices REST API.

Some best practives for REST API design are implicit in the HTTP standard, while others have emerged over the past few years. This part presents a set of REST API best practices that should answer clear and concise questions like:

- How do i map non-CRUD operations to my URI.
- What is the appropriate HTTP response status code for a given scenario
- etc.

2.1 URIs

REST APIs use Uniform Resource Identifiers (URIs) to address resources. The syntax of a URI is as follows

```
URI = scheme "://" authority "/" path ["?" query] ["#" fragment]
```

2.1.1 Indicate hierarchical relationships by forward slash.

The forward slash character (`/`) must be used in the path portion of the URI to indicate a hierarchical relationship between resources.

2.1.2 Trailing forward slash should not be included in URIs.

A forward slash as the last character within a URIs path adds no semantic value. Every character within a URI counts within a resource's unique identity and two different URIs map to two different resources. Therefore REST APIs should not expect a trailing slash and should not include them in the links that they provide to clients.

2.1.3 Hyphens should be used to improve 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 segment. As rule of thumb, use the hyphen character where you would use a space.

2.1.4 Underscores should not be used in URIs

Browsers often underline URIs to provide a visual cue that they are clickable. Depending on the application's font, the underscore character (_) can get partially obscured or completely hidden by this underlying. To avoid this confusion, use hyphens (`-`) instead of underscores.

See also (Hyphens should be used to improve readability of URIs

2.1.5 Lowercase letters should be preferred in URI paths

Lowercase letters are preferred in URI paths since RFC 3986 defines URIs as case sensitive expect for the scheme and host components. For example

```
http://uniknow.github.io/AgileDev (1)
http://uniknow.github.IO/AgileDev (2)
http://uniknow.github.io/agileDev (3)
```

URI 1 and 2 are considered identical. URI 3 is not the same as URIs 1 and 2, which may cause unnecessary confusion.

2.1.6 File extensions should not be included in URIs

A REST API should not include artificial file extensions in URIs to indicate the format of a message entity body. Instead, they should rely on the media type, as communicated through the `Content-Type` header.

2.1.7 A singular noun should be used for document names

A URI representing a document resource should be named with a singular noun.

For example, the URI for a single user document would have the following form:

```
http://api.tutorial.rest.org/authors/mark
```

2.1.8 A plural noun should be used for collection names

A URI identifying a collection should be named with a plural noun and should be chose in such way that it reflects what it uniformly contains.

For example, the URI for a collection of authors would be as follows:

```
http://api.tutorial.rest.org/authors
```

Page 17 …

2.2 Resources

- REST API Design Rulebook