”RESTful” services

Patryk Czarnik

XML and Applications 2016/2017
Lecture 7 – 28.04.2017

REST – motivation

Why REST services?

- Common practice of using HTTP not only to read/write files, but also automatically generated data
- Need to create easily accessible services, especially as a support for web applications
  - But: Complexity and inefficiency of SOAP-based services
- AJAX – JavaScript communicates with server(s) to load (or sometimes write) data
  - Later: also mobile applications communicating with servers in a similar way

REST is a concept that standardises and organises the way such services can be built.

REST – basic concept

- Representational State Transfer (**REST**)
  - The name first used by Roy Fielding (2002)
- Direct use of **HTTP** and its methods: **GET**, **PUT**, **DELETE**, ...
- **Resource**
  - is available under an **URL**
  - can be **represented** in one or more formats
  - can be read or written (=”transferred”)
    - It's the designer's decision which operations are available.
- A **service** is a set of resources (URLs) with some operations available on them.
  - Often called ”Web API”

REST – typical organisation

- Good practice for resources and their URLs:
  - URLs unique
  - resources organised in collections
    http://rest.example.org/service/orders/302312
  - even tree-like structure corresponding to logical structure of data or service functionality
- Data formats
  - most of the resources – records of data
    ”structural text document” is the best format
  - most popular: JSON, but XML also in use
  - binary data can be used in special cases:
    image, PDF document, ... anything
    (for HTTP it doesn't make a difference; sometimes the size...)

HTTP methods (in REST, but not only)

- **`GET`** – read the resource with no side effects
- **`PUT`** – write the resource
  - request body contains new content
  - for writing new and overriding existing resources
  - caller must use valid URL of the resource
- **`DELETE`** – delete the resource
- **`POST`** – “take this data and do something with it”
  - doesn't directly fit to REST *remote data access* concept
  - used in practice to write a new resource into collection (without knowing the final URL of the resource being added)
  - or to call any additional logic in RPC manner
- `OPTIONS`, `HEAD`, `PATCH` – sometimes used for technical purposes

Advanced techniques

- Content type negotiation
  - server supports many formats for the same resource; format chosen according to client's preference (`Accept` header)
- Caching and version tracking
  - according to HTTP solutions: using HEAD method and headers, based on modification time or *entity tag*
- Links and navigation
  - one resource can refer to another (instead of including it)
  - *redirect* implemented with HTTP response codes and headers
- Security
  - transport layer: HTTP**S**
  - authorisation: HTTP basic (user/password), SSL-certificates, token-based solutions (analogous to Kerberos, but usually implemented independently), growing pupularity of OAuth-2
  - limiting request rate

REST vs SOAP services

REST

- HTTP
- many data formats
- many URLs
- fixed set of operations and standard CRUD semantics
  (apart from general POST method)

Typical usage:

- server-side support for web (JS) and mobile apps
- ”web API” of business services related to web applications

SOAP

- different transport protocols, although HTTP mostly used
- only XML (attachments in other formats)
- one URL, for HTTP always **POST**
- unlimited set of operations

Typical usage:

- business services of large enterprises (banks etc.) or public administration
- integration with external partners' systems or internally in heterogeneous architectures

JAX-RS – REST in Java

- Java API for RESTful Services (JAX-RS)
- Annotation driven API
- Support for different ways of passing arguments
- Content-type negotiation
  - the same resource may be available in different formats
  - built-in support for XML (using JAXB) and JSON (implementation-dependent)
- Easy to write HTTP servers
  - REST-specific logic has to be written manually

JAX-RS – a simple resource class

```
@Path("/orders")
public class OrderService {
  @GET
  public List<Order> getOrders() {
    // ...
    List<Order> result =
       database.readOrders();
    return result;
  }
  ...
}
```

JAX-RS – URI processing

```
@Path("/orders/{id}")
public class OrderService {
  @GET
  public Order getOrderById(
    @PathParam("id") int orderId
  ) {
    return database.findOrder(orderId);
  }
  ...
}
```

JAX-RS – content negotiation

```
@Path("/orders/{id}")
public class OrderService {
  @GET
  @Produces({"application/xml","application/json"})
  public Order getOrderById(@PathParam("id") int id) {
    Order order = database.findOrder(id);
    return order;
  }
  @GET
  @Produces("text/html")
  public String getOrderById(@PathParam("id") int id) {
    Order order = database.findOrder(id);
    return order.getHTML();
  }
}
```

JAX-RS – reading and writing

```
@Path("/orders/{id}")
public class OrderService {
  @GET
  @Produces({"application/xml","application/json"})
  public Order getOrderById(@PathParam("id") int id) {
    Order order = database.findOrder(id);
    return order;
  }
  @PUT
  @Consumes({"application/xml","application/json"})
  public void writeOrder(@PathParam("id") int id, Order order) {
    database.writeOrderUnderId(id, order); // whatever it means :)
  }
}
```

JAX-RS – more possibilities

- `Response` class – full control of the returned HTTP response (response code, headers, etc.)
- Using HTTP headers and cookies
- Writing custom ”providers” to handle other data formats
- Reading/writing binary data (see `StreamingOutput`)
- Accessing servlet-technology-level objects from the context