Laboratorium nr 8

Usługi sieciowe typu REST

Wstęp

Usługi typu REST (REpresentional State Transfer) to wzorzec oprogramowania sieciowego, którego główną jednostką jest zasób. W przeciwieństwie do typowych usług sieciowych opartych o wsdl/soap/uddi są lekkie do implementacji i wymagają jedynie obsługi protokołu http.

Standardem usług typy REST na platformie Java jest JAX-RS (JSR311), którego referencyjną implementacją jest projekt Jersey (http://jersey.java.net/).

Konfiguracja projektu webowego

Zadanie będzie realizowane jako aplikacja webowa uruchamiana w dowolnym kontenerze webowym. Ponieważ kontener sam w sobie nie zawiera bibliotek JAX-RS należy je dostarczyć razem z aplikacją. Wymagane są dwie biblioteki: jersey-bundle-1.16.jar i asm-3.3.1.jar. Biblioteki można znaleźć na stronie domowej projektu: http://jersey.java.net/nonav/documentation/latest/chapter_deps.html

Wszystkie żądania wywołania usług typu REST są obsługiwane przez Servlet dostarczony razem z implementacją. Należy go skonfigurować w standardowym deskryptorze wdrożenia (standard deployment descriptor) web.xml,

```
<servlet>
   <servlet-name>Jersey REST Service</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><param-value>pl.test.soa.rest</param-value></param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>Jersey REST Service</servlet-name>
   <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
```

Parametr *com.sun.jersey.config.property.packages* definiuje paczkę, w której będą się znajdowały implementacje usług. Element *<url-pattern>/resources/*</url-pattern>* definiuje pod jakim adresem będą wystawione usługi.

Implementacja usługi

Konfiguracja usługi realizowana jest za pomocą następujących adnotacji:

- @Path – relatywny adres pod, którym będzie dostępna klasa/metoda,
- żądania HTTP:
  - @GET,
  - @POST,
  - @PUT,
  - @DELETE,
  - @HEAD;
- @PathParam – parametry pobierane z URI żądania jako elementy URI,
- @QueryParam – parametry pobierane z URI żądania jako parametry żądania,
- @Consumes – specyfikacja typu MIME konsumowanego przez usługę,
- @Produces – specyfikacja typu MIME zwracanego przez usługę.

Adresowanie usług jest realizowane w następujący sposób:

- globalny adres dla całej klasy poprzez adnotację @Path("/base") na klasie,
- relatywny adres dla metody poprzez adnotację @Path("/relative") na metodzie.

Parametry przekazywane do usługi definiuje się następująco:

- elementy URI:
  - oznaczone w adnotacji @Path("/relative/{username}"),
  - pobierane w metodzie poprzez @PathParam("username"):
    - public void addUser(@PathParam("username") String username);
  - definiowanie filtrów poprzez wyrażenia regularne:
    - @Path("relative/{username:[a-z]+}");
- parametry żądania:
  - oznaczane w metodzie poprzez @QueryParam("name").

Wartości przyjmowane i zwracane przez usługę:

- prócz zwykłych argumentów może przyjmować dokumenty np.: xml i json,
- format definiowany za pomocą typu MIME:
  - text/plain, text/xml, application/xml;
- usługa może działać na obiektach dowolnych klas spełniających wymogi JAXB.

Przykład usługi

Poniżej znajdują się przykłady różnych metody zdefiniowanych jako usługi typu REST. Zakłada się że aplikacja webowa została wdrożona pod nazwą „rest", deskryptor web.xml wskazuje że usługi dostępne są pod adresem „/resources/*" a serwer nasłuchuje pod adresem „http://localhost:8084/".

```
@Path("/hello")
public class Hello {


   @GET
   @Produces(MediaType.TEXT_PLAIN)
   public String sayPlainTextHello() {
      return "Hello there!";
   }


   @GET
   @Produces(MediaType.TEXT_XML)
   public String sayXMLHello() {
      return "<?xml version=\"1.0\"?>" + "<hello> Hello there!" + "</hello>";
   }


   @GET
   @Produces(MediaType.TEXT_HTML)
   public String sayHTMLHello() {
      return "<html> " + "<title>" + "Hello Jersey" + "</title>"
         + "<body><h1>" + "Hello there!" + "</body></h1>" + "</html> ";
   }
}
```

Pomimo implementacji kilku metod wszystkie nasłuchują pod adresem http://localhost:8084/rest/resources/hello. Spowodowane jest to faktem, że adnotacja @Path została nałożona tylko na klasę. Ponieważ metody mają różnie zdefiniowane typy zwracane (adnotacje @Produces) to fakt, która się wykona zależy od formatów akceptowanych przez klienta.

```
   @GET
   @Produces(MediaType.TEXT_XML)
   @Path("/xml")
   public String sayXMLHello2() {
      return sayXMLHello();
   }
```

Usługa dostępna pod adresem http://localhost:8084/rest/resources/hello/xml i zwraca jedynie dokument XML.

```
@XmlRootElement
public class Message {
    private String title;
    private String body;
    public Message() {
    }
    public Message(String title, String body) {
        this.title = title;
        this.body = body;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
}
```

Przykład klasy zgodnej z JAXB, która posłuży w komunikacji z usługą.

```
    @GET
    @Produces(MediaType.APPLICATION_XML)
    @Path("/message/xml")
    public Message sendMessage() {
        return new Message("title", "body");
    }
```

Usługa automatycznie konwertuje zwracany przez metodę obiekt Message do dokumentu XML.

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/message/json")
    public Message sendJSONMessage() {
        return new Message("title", "body");
    }
```

Usługa automatycznie konwertuje zwracany przez metodę obiekt Message do dokumentu JSON.

```
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/to/{name}")
public String sayHelloTo(@PathParam("name") String name) {
   return "Hello there " + name + "!";
}
```

Usługa dostępna pod adresem http://localhost:8084/rest/resources/hello/to/{login} gdzie wartość {login} należy zamienić na dowolny ciąg znaków. Ciąg znaków przekazany jako {login} zostanie automatycznie przekazany do parametru *name* metody.

```
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/to")
public String sayHelloTo2(@QueryParam("name") String name) {
   return "Hello there " + name + "!";
}
```

Usługa dostępna pod adresem http://localhost:8084/rest/resources/hello/to?name={login} gdzie wartość {login} należy zamienić na dowolny ciąg znaków. Ciąg znaków przekazany jako {login} zostanie automatycznie przekazany do parametru *name* metody.

```
@PUT
@Consumes(MediaType.APPLICATION_XML)
@Path("/message/put")
public void receiveXMLMessage(Message message) {
   System.out.println(message.getTitle());
   System.out.println(message.getBody());
}
```

Usługa automatycznie konwertuje przesłany dokument XML do obiektu Message (o ile składnia dokumentu jest z godna z implementacją klasy Message).

Przykład klienta

Wszystkie usługi z przykładu nasłuchujące na żądania GET protokołu HTTP można testować z poziomu przeglądarki. Usługa nasłuchująca na żądania PUT wymaga dodatkowej aplikacji.

W przykładzie klient usługi został zrealizowany jako desktopowa aplikacja Java SE. Biblioteki wykorzystane w projekcie są takie same jak w przypadku aplikacji webowej.

Klasy wykorzystywane po stronie klienta to:

- ClientConfig – konfiguracja klienta,

- Client – obiekt klienta służący do połączenia z kilkoma zasobami,
- WebResource – reprezentacja jednego zasobu na którym będą wywoływane żądania,
- URI – klasa wchodząca w skład standardowej edycji Java SE, reprezentuje adres zasobu.

```
URI uri = UriBuilder.fromUri("http://localhost:8084/rest/").build();
ClientConfig config = new DefaultClientConfig();

Client client = Client.create(config);

WebResource service = client.resource(uri);
```

Konfiguracja połączenia z zasobem.

```
System.out.println(service.path("resources").path("hello").accept(MediaType.TEXT_PLAIN).get(Strin
g.class));

System.out.println(service.path("resources").path("hello").accept(MediaType.TEXT_XML).get(String.
class));

System.out.println(service.path("resources").path("hello").accept(MediaType.TEXT_HTML).get(Strin
g.class));
```

Pobranie zasobów text, xml i html jako tekst.

```
Message message =

 service.path("resources").path("hello").path("message").path("xml").accept(MediaType.APPLICAT
ION_XML).get(Message.class);

System.out.println(message.getTitle());

System.out.println(message.getBody());
```

Pobranie zasobu jako dokumentu reprezentującego obiekty Message. Klient automatycznie konwertuje dokument xml do obiektu Message.

```
service.path("resources").path("hello").path("message").path("put").type(MediaType.APPLICATIO
N_XML).put(new Message("title", "body"));
```

Wywołanie żądania PUT na usłudze i przesłania obiektu Message automatycznie skonwertowanego do dokumentu XML.