Technologia i rozwiązania

REST
Najlepsze praktyki i wzorce w języku Java

Usprawnij wymianę danych z usługą REST!

Bhakti Mehta

Helion

[PACKT] PUBLISHING open source community experience distilled

Tytuł oryginału: RESTful Java Patterns and Best Practices

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-0644-8

First published in the English language under the title
„RESTful Java Patterns and Best Practices" (9781783287963).

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/restja
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

1

ROZDZIAŁ

Podstawy REST

Usługi sieciowe w tradycyjnej technologii SOA, umożliwiające zróżnicowaną komunikację między aplikacjami, istnieją już od pewnego czasu. Jednym ze sposobów obsługi tej komunikacji jest użycie technologii ***Simple Object Access Protocol*** (**SOAP**) i ***Web Service Description Language*** (**WSDL**). Są to standardy oparte na formacie XML doskonale sprawdzające się, gdy między usługami jest ścisły kontakt. Ale nastała era usług rozproszonych. Teraz różne klienty z internetu, urządzenia przenośne, jak również inne usługi (wewnętrzne i zewnętrzne) mogą używać interfejsów API udostępnianych przez różnych dostawców i różne platformy *open source*. To sprawia, że potrzebne są technologie łatwej wymiany informacji między usługami rozproszonymi w różnych miejscach, z przewidywalnymi, solidnymi, ściśle zdefiniowanymi interfejsami.

Protokół HTTP 1.1, zdefiniowany w dokumencie RFC 2616, jest powszechnie używany w rozproszonych systemach hipermedialnych. Technologia ***Representational State Transfer*** (**REST**) bazuje na HTTP i może być używana wszędzie tam, gdzie ten protokół. W tym rozdziale przedstawione są podstawowe wiadomości na temat projektowania usług typu RESTful oraz używania takich usług za pomocą standardowych interfejsów API Javy.

W rozdziale omówiono następujące zagadnienia:

- Wprowadzenie do technologii REST.
- Bezpieczeństwo i idempotentność.
- Zasady projektowe dotyczące budowy usług typu RESTful.
- Standardowe API Javy dla usług typu RESTful.
- Najlepsze techniki projektowania usług typu RESTful.

Wprowadzenie do REST

REST to styl architektoniczny zgodny z takimi standardami sieciowymi jak czasowniki HTTP i identyfikatory URI. Obowiązują w nim następujące zasady:

- Wszystkie zasoby określa identyfikator URI.
- Każdy zasób może mieć liczne reprezentacje.
- Każdy zasób można pobrać, zmodyfikować, utworzyć i usunąć standardowymi metodami HTTP.
- Na serwerze nie są przechowywane żadne informacje o stanie.

REST i bezstanowość

W REST obowiązuje zasada **bezstanowości**. Każde żądanie przesyłane przez klienta do serwera musi zawierać wszystkie informacje potrzebne do obsługi tego zdarzenia. To poprawia widoczność, niezawodność i skalowalność żądań.

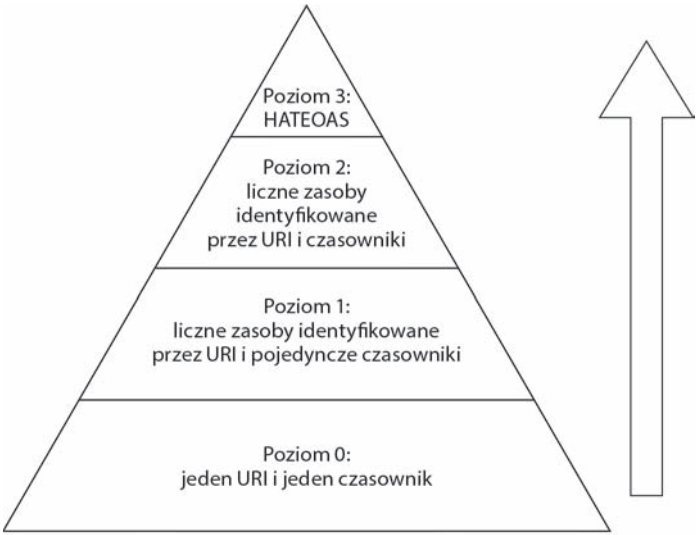
Poprawa *widoczności* wynika z tego, że system monitorujący żądania nie musi szukać szczegółów poza żądaniami. *Niezawodność* poprawia się dzięki wyeliminowaniu punktów kontrolnych i wznowienia w przypadku częściowych niepowodzeń operacji. Poprawa *skalowalności* jest efektem zwiększenia liczby żądań, które jest w stanie przetworzyć serwer, co jest możliwe dzięki temu, że serwer nie musi przechowywać informacji o stanie.

> Roy Fielding napisał doktorat na temat stylu architektonicznego REST, w którym szczegółowo opisał bezstanowość tej technologii. Więcej informacji na ten temat można znaleźć pod adresem *http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm*.

To są podstawowe wiadomości o technologii REST. Teraz zajmiemy się różnymi poziomami dojrzałości i zobaczymy, gdzie pośród nich mieści się ta technologia.

Model dojrzałości Richardsona

Model dojrzałości Richardsona to opracowany przez Leonarda Richardsona model opisujący podstawy REST pod względem zasobów, czasowników i hipermediów. Punktem początkowym tego modelu jest wykorzystanie HTTP jako warstwy transportowej. Ukazuje to poniższy schemat.



Model dojrzałości Richardsona

Poziom 0 — zdalne wywoływanie procedur

Do poziomu 0 zalicza się przesyłanie danych przy użyciu technologii SOAP i XML-RPC jako **POX** (ang. *Plain Old XML* — zwykły XML). Używana jest tylko metoda `POST`. Jest to najprostszy sposób budowania aplikacji SOA z jedną metodą `POST` i przy użyciu formatu XML do komunikacji między usługami.

Poziom 1 — zasoby REST

Na poziomie 1 używane są metody `POST`, a zamiast funkcji i przekazywania argumentów wykorzystuje się identyfikatory URI REST. Zatem nadal używana jest tylko jedna metoda HTTP. Zaletą tego poziomu w stosunku do zerowego jest podział złożonej funkcjonalności na kilka zasobów za pomocą jednej metody `POST` służącej do komunikacji między usługami.

Poziom 2 — dodatkowe czasowniki HTTP

Na poziomie drugim jest więcej czasowników, np. `GET`, `HEAD`, `DELETE`, `PUT` i oczywiście `POST`. Poziom ten reprezentuje rzeczywisty przypadek użycia technologii REST, w której wykorzystuje się różne czasowniki HTTP do wykonywania różnych żądań, a system może zawierać wiele zasobów.

Poziom 3 — HATEOAS

HATEOAS (ang. *Hypermedia as the Engine of Application State* — hipermedia jako mechanizm obsługi stanu aplikacji) reprezentuje najwyższy stopień dojrzałości w modelu Richardsona. Odpowiedzi na żądania klientów zawierają elementy hipermedialne, przy użyciu których klient może zdecydować, co robić dalej. Zasady te ułatwiają wykrywanie usług i sprawiają, że odpowiedzi są bardziej zrozumiałe. Toczą się dyskusje na temat tego, czy HATEOAS rzeczywiście spełnia wymagania RESTful, ponieważ reprezentacja zawiera o wiele więcej informacji niż tylko opis zasobu. W rozdziale 5. przedstawiam parę przykładów pokazujących, jak zaimplementowano HATEOAS jako część API niektórych platform, np. PayPal.

W następnym podrozdziale wyjaśniam pojęcia bezpieczeństwo i idempotentność, które są bardzo ważne w usługach RESTful.

Bezpieczeństwo i idempotentność

W poniższych dwóch podrozdziałach dokładniej wyjaśniam znaczenie bezpieczeństwa i idempotentności metod.

Bezpieczeństwo metod

Bezpieczna metoda to taka, która nie zmienia stanu na serwerze. Warunek ten spełnia na przykład metoda `GET /v1/coffees/orders/1234`.

> Bezpieczne metody, do których zaliczają się `GET` i `HEAD`, można buforować.
>
> Metoda `PUT` nie jest bezpieczna, ponieważ tworzy lub modyfikuje zasoby na serwerze. To samo dotyczy metody `POST`. Z kolei metoda `DELETE` nie jest bezpieczna, ponieważ usuwa zasoby z serwera.

Idempotentność metod

Metoda idempotentna to taka, która zwraca taki sam wynik niezależnie od tego, ile razy zostanie wywołana.

> Metoda `GET` jest idempotentna, ponieważ niezależnie od tego, ile razy się ją wywoła, zawsze zwraca taką samą odpowiedź.
>
> Metoda `PUT` też jest idempotentna, ponieważ wielokrotne jej wywołanie powoduje aktualizację tego samego zasobu i nie zmienia to wyniku.

Metoda POST nie jest idempotentna, ponieważ jej wielokrotne wywołanie może dawać różne skutki i powodować powstanie wielu zasobów. Metoda DELETE jest idempotentna, gdyż usunięty zasób znika i powtórne wywołanie tej samej metody niczego nie zmieni.

Zasady projektowe dotyczące budowy usług typu RESTful

Poniżej w punktach przedstawiam proces projektowania, tworzenia i testowania usług typu RESTful. Dalej znajduje się dokładniejszy opis każdego z tych etapów:

- Wyznaczenie identyfikatorów URI zasobów.

  Polega na wybraniu rzeczowników do reprezentowania zasobu.
- Identyfikacja metod obsługiwanych przez zasób.

  Polega na wybraniu metod HTTP do wykonywania operacji CRUD (ang. *create*, *read*, *update*, *delete* — utworzenie, odczytanie, aktualizacja, usunięcie).
- Identyfikacja różnych reprezentacji zasobu.

  Polega na wybraniu, czy zasób będzie reprezentowany w formacie JSON, XML, HTML, czy tekstowym.
- Implementacja usług RESTful przy użyciu API JAX-RS.

  Interfejs API należy zaimplementować na podstawie specyfikacji JAX-RS.
- Wdrożenie usług RESTful.

  Wdrożenie usługi w kontenerze aplikacji, np. Tomcat, GlassFish lub WildFly. Na przykładach pokazuję, jak tworzy się pliki WAR, i prezentuję sposób wdrożenia na serwerze GlassFish 4.0. Poza tym przedstawiony przykład działa w każdym kontenerze zgodnym z Java EE 7.
- Testowanie usług RESTful.

  Polega na napisaniu API klienta do testowania usług lub użyciu narzędzi cURL albo przeglądarkowych do testowania żądań REST.

Wyznaczenie identyfikatorów URI zasobów

Zasoby RESTful są identyfikowane przez identyfikatory URI. Dzięki temu technologia REST jest rozszerzalna.

Poniższa tabela zawiera przykłady identyfikatorów URI, które mogą reprezentować różne zasoby w systemie.

URI	Opis
`/v1/library/books`	Możliwa reprezentacja kolekcji zasobów książkowych w bibliotece.
`/v1/library/books/isbn/12345678`	Możliwa reprezentacja jednej książki identyfikowanej przez numer ISBN 123456.
`/v1/coffees`	Możliwa reprezentacja wszystkich kaw sprzedanych w kawiarni.
`/v1/coffees/orders`	Możliwa reprezentacja wszystkich zamówionych kaw w kawiarni.
`/v1/coffees/orders/123`	Możliwa reprezentacja pojedynczego zamówienia kawy o identyfikatorze 123.
`/v1/users/1235`	Możliwa reprezentacja użytkownika o identyfikatorze w systemie 1235.
`/v1/users/5034/books`	Możliwa reprezentacja wszystkich książek użytkownika o identyfikatorze 5034.

Wszystkie przedstawione identyfikatory są zbudowane wg jasnego wzorca, który klient może bez trudu zinterpretować. Wszystkie te zasoby mogą mieć liczne reprezentacje, np. w formacie JSON, XML, HTML lub tekstowym, i można nimi zarządzać za pomocą metod `GET`, `PUT`, `POST` i `DELETE`.

Identyfikacja metod obsługiwanych przez zasób

Czasowniki HTTP stanowią ważny składnik jednolitego ograniczenia interfejsu, które definiuje związek między czynnościami opisywanymi przez dany czasownik w stosunku do opisanego za pomocą rzeczowników zasobu REST.

Poniższa tabela zawiera zestawienie metod HTTP i opis powodowanych przez nie zdarzeń oraz prosty przykład kolekcji książek w bibliotece.

Metoda HTTP	URI zasobu	Opis
`GET`	`/library/books`	Pobiera listę książek.
`GET`	`/library/books/isbn/12345678`	Pobiera książkę o numerze ISBN 12345678.
`POST`	`/library/books`	Tworzy nowe zamówienie książki.
`DELETE`	`/library/books/isbn/12345678`	Usuwa książkę o numerze ISBN 12345678.
`PUT`	`/library/books/isbn/12345678`	Aktualizuje książkę o numerze ISBN 12345678.
`PATCH`	`/library/books/isbn/12345678`	Częściowo aktualizuje książkę o numerze ISBN 12345678.

W kolejnym podrozdziale znajduje się opis zastosowania każdego z czasowników HTTP w kontekście REST.

Czasowniki HTTP w REST

Czasowniki HTTP stanowią dla serwera informację o tym, co ma zrobić z otrzymanymi danymi.

GET

`GET` to najprostsza metoda HTTP pozwalająca uzyskać dostęp do zasobu. Gdy klient kliknie adres URL w przeglądarce internetowej, aplikacja ta wysyła żądanie `GET` pod ten właśnie adres. Metoda `GET` jest bezpieczna i idempotentna. Żądania wysyłane tą metodą są buforowane i mogą zawierać parametry.

Poniżej znajduje się proste żądanie `GET` pobierające wszystkich aktywnych użytkowników:

```
curl http://api.foo.com/v1/users/12345?active=true
```

POST

Metoda `POST` służy do tworzenia zasobów. Żądania wysyłane przy użyciu tej metody nie są idempotentne ani bezpieczne. Wielokrotne wywołania mogą spowodować utworzenie wielu zasobów.

Żądanie `POST` powinno powodować unieważnienie odpowiedniego elementu w buforze, jeśli taki istnieje. W żądaniach `POST` nie zaleca się stosowania parametrów zapytaniowych.

Poniżej znajduje się żądanie utworzenia użytkownika:

```
curl –X POST -d'{"name":"Jan Kowalski","username":"jkow", "phone":
  "412-344-5644"}' http://api.foo.com/v1/users
```

PUT

Metoda `PUT` służy do aktualizowania zasobów. Jest ona idempotentna, ale nie jest bezpieczna. Wielokrotne żądania tego typu powinny dawać taki sam efekt w postaci zaktualizowania zasobu.

Żądania `PUT` powinny unieważniać zawartość bufora, jeśli taka istnieje.

Poniżej znajduje się przykład żądania `PUT` aktualizującego użytkownika:

```
curl –X PUT -d'{ "phone":"413-344-5644"}'
http://api.foo.com/v1/users
```

DELETE

Metoda `DELETE` służy do usuwania zasobów. Jest idempotentna, ale nie jest bezpieczna. Idempotentność wynika z tego, że zgodnie ze specyfikacją RFC 2616 skutki uboczne dowolnej większej od zera liczby żądań są takie same jak jednego żądania. Oznacza to, że po usunięciu zasobu kolejne wywołania metody `DELETE` nic nie zmieniają.

Poniżej znajduje się przykładowe żądanie usuwające użytkownika:

```
curl –X DELETE http://foo.api.com/v1/users/1234
```

HEAD

Żądania typu `HEAD` są podobne do `GET`. Różnica między nimi polega na tym, że w odpowiedzi na żądanie `HEAD` zwracane są tylko nagłówki HTTP, bez treści. Metoda `HEAD` jest idempotentna i bezpieczna.

Poniżej znajduje się przykładowe żądanie `HEAD` wysyłane za pomocą narzędzia cURL:

```
curl –X HEAD http://foo.api.com/v1/users
```

> Jeśli zasób jest duży, to za pomocą żądania `HEAD` można sprawdzić, czy coś się w nim zmieniło, zanim się je pobierze przy użyciu żądania `GET`.

PUT czy POST

Zgodnie z dokumentem RFC różnica między metodami `PUT` i `POST` dotyczy identyfikatora URI żądania. W metodzie `POST` przesłany identyfikator URI definiuje jednostkę, która ma obsłużyć żądanie. W żądaniu `PUT` natomiast identyfikator URI zawiera tę jednostkę.

A zatem `POST /v1/coffees/orders` oznacza utworzenie nowego zasobu i zwrócenie opisującego go identyfikatora. `PUT /v1/coffees/orders/1234` oznacza natomiast aktualizację zasobu o identyfikatorze `1234`, jeśli taki istnieje. Jeśli nie ma takiego zasobu, zostanie utworzone nowe zamówienie, do którego identyfikacji zostanie użyty URI `orders/1234`.

> Zarówno metody `PUT`, jak i `POST` można używać do tworzenia i aktualizacji zasobów. Wybór jednej z nich zależy głównie od tego, czy potrzebna jest idempotentność metody, oraz od lokalizacji zasobu.

W następnym podrozdziale dowiesz się, jak identyfikować różne reprezentacje zasobu.

Identyfikacja różnych reprezentacji zasobu

Zasoby RESTful są jednostkami abstrakcyjnymi, które przed przesłaniem do klienta trzeba poddać serializacji do jakiegoś reprezentacyjnego formatu. Wśród najczęściej używanych reprezentacji można wymienić XML, JSON, HTML i zwykły tekst. Zasób może dostarczać klientowi reprezentację w zależności od tego, co klient ten jest w stanie przyjąć. Klient może określić preferowane przez siebie języki i typy mediów. Nazywa się to **negocjacją treści** i zostało szczegółowo opisane w rozdziale 2.

Implementowanie API

Wiesz już mniej więcej, jak projektować zasoby RESTful i jakich czasowników HTTP używać do wykonywania różnych działań na tych zasobach, więc możemy przejść do kwestii implementowania API i budowania usługi typu RESTful. Głównym tematem tego podrozdziału jest:

- API Javy dla usług RESTful (JAX-RS).

API Javy dla usług RESTful (JAX-RS)

API Javy dla usług RESTful służy do budowania i rozwijania aplikacji wg zasad technologii REST. Przy użyciu JAX-RS można udostępniać obiekty Javy jako usługi sieciowe typu RESTful, które są niezależne od podstawowej technologii i używają prostego API opartego na adnotacjach.

Najnowsza wersja specyfikacji to JAX-RS 2.0. Od wersji JAX-RS 1.0 różni się przede wszystkim:

- narzędziami do sprawdzania poprawności ziaren,
- obsługą API klienta,
- możliwością wykonywania wywołań asynchronicznych.

Implementacja specyfikacji JAX-RS nazywa się Jersey.

Wszystkie wymienione tematy zostały szczegółowo opisane w kolejnych rozdziałach. Przedstawiam prosty przykład kawiarni, w którym można utworzyć zasób REST o nazwie `CoffeesResource` o następujących umiejętnościach:

- podanie szczegółów złożonych zamówień,
- tworzenie nowych zamówień,
- sprawdzenie informacji o wybranym zamówieniu.

Tworzenie zasobu RESTful zaczniemy od utworzenia obiektu Javy o nazwie `CoffeesResource`. Poniżej znajduje się przykład zasobu JAX-RS:

```
@Path("v1/coffees")
public class CoffeesResource {

    @GET
    @Path("orders")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Coffee> getCoffeeList( ){
        // implementacja
}
```

1. W powyższym kodzie został utworzony niewielki obiekt Javy o nazwie `CoffeesResource`. Klasę tę opatrzyłam adnotacją `@Path("v1/coffees")` określającą ścieżkę URI, dla której klasa ta obsługuje żądania.

2. Następnie definiujemy metodę o nazwie `getCoffeeList()`. Ma ona następujące adnotacje:
   - `@GET`: oznacza, że metoda reprezentuje żądanie HTTP `GET`.
   - `@PATH`: w tym przypadku żądania `GET` zasobu `v1/coffees/orders` będą obsługiwane przez metodę `getCoffeeList()`.
   - `@Produces`: definiuje typy mediów zwracane przez ten zasób. W omawianym przykładzie określono typ mediów `MediaType.APPLICATION_JSON`, którego wartość to `application/json`.
3. Inna metoda tworząca zamówienie wygląda tak:

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@ValidateOnExecution
public Response addCoffee(@Valid Coffee coffee) {
// implementacja
}
```

Jest to metoda o nazwie `addCoffee()` zawierająca następujące adnotacje:

- `@POST`: oznacza, że metoda reprezentuje żądanie HTTP `POST`.
- `@Consumes`: definiuje przyjmowane przez zasób typy mediów. W omawianym przykładzie określono typ mediów `MediaType.APPLICATION_JSON`, którego wartość to `application/json`.
- `@Produces`: definiuje typy mediów zwracane przez ten zasób. W omawianym przykładzie określono typ mediów `MediaType.APPLICATION_JSON`, którego wartość to `application/json`.
- `@ValidateOnExecution`: określa, dla których metod parametry i wartości zwrotne mają być sprawdzane. Szerzej o adnotacjach `@ValidateOnExecution` i `@Valid` piszę w rozdziale 3.

Jak widać, zmiana prostego obiektu Javy w usługę REST jest bardzo łatwa. Teraz obejrzymy podklasę klasy `Application`, która będzie zawierała definicje komponentów aplikacji JAX-RS włącznie z metadanymi.

Poniżej znajduje się kod źródłowy przykładowej podklasy klasy `Application` o nazwie `CoffeeApplication`:

```
@ApplicationPath("/")
public class CoffeeApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(CoffeesResource.class);
        return classes;
    }
```

W podklasie tej została nadpisana metoda `getClasses()` oraz dodano `CoffesResource`. W pliku WAR podklasy klasy `Application` mogą znajdować się w katalogach *WEB-INF/classes* i *WEB-INF/lib*.

Wdrażanie usług typu RESTful

Następnym krokiem po utworzeniu zasobu i dodaniu metadanych do podklasy klasy `Application` jest utworzenie pliku WAR, który można wdrożyć w każdym kontenerze serwletów.

Kod źródłowy opisywanych przykładów znajduje się w plikach do pobrania z serwera FTP. Dodatkowo można w nich znaleźć szczegółowe instrukcje, jak uruchomić te przykłady.

Testowanie usług typu RESTful

Teraz możemy użyć funkcjonalności API klienta JAX-RS 2.0 w celu uzyskania dostępu do zasobów.

W tym podrozdziale opisane są następujące tematy:

- API klienta w JAX-RS 2.0,
- uzyskiwanie dostępu do zasobów RESTful przy użyciu narzędzia cURL lub rozszerzenia przeglądarki internetowej o nazwie Postman.

API klienta w JAX-RS 2.0

W JAX-RS 2.0 dodano nowe API klienckie służące do uzyskiwania dostępu do zasobów RESTful. Jego punkt początkowy to `javax.ws.rs.client.Client`.

Z tego nowego API można korzystać w następujący sposób:

```
Client client = ClientFactory.newClient();
WebTarget target = client.target("http://. . ./coffees/orders");
String response = target.request().get(String.class);
```

Jak widać w tym przykładzie, domyślny egzemplarz klienta tworzy się przy użyciu metody `ClientFactory.newClient()`. Za pomocą metody `target()` został utworzony obiekt `WebTarget`. Z wykorzystaniem obiektów tego typu przygotowuje się żądanie przez dodanie metody i parametrów zapytania.

Zanim nie pojawiły się nowe API, dostęp do zasobów REST uzyskiwało się w następujący sposób:

```
URL url = new URL("http://. . ./coffees/orders");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setDoInput(true);
conn.setDoOutput(false);
BufferedReader br = new BufferedReader(new InputStreamReader(conn.
  getInputStream()));
String line;
while ((line = br.readLine()) != null) {
  //...
}
```

Przykład ten wyraźnie pokazuje, jak dużego postępu dokonano w API klienckim JAX-RS 2.0 — wyeliminowano konieczność używania klasy `HttpURLConnection`, zamiast której można używać API `Client`.

Jeśli żądanie jest typu `POST`:

```
Client client = ClientBuilder.newClient();
Coffee coffee = new Coffee(...);
WebTarget myResource = client.target("http://foo.com/v1/coffees");
myResource.request(MediaType.APPLICATION_XML) .post(Entity.xml(coffee),
  Coffee.class);
```

metoda `WebTarget.request()` zwraca obiekt `javax.ws.rs.client.InvocationBuilder`, który za pomocą metody `post()`wywołuje żądanie HTTP `POST`. Metoda `post()` pobiera jednostkę z egzemplarza `Coffee` i określa typ mediów jako `APPLICATION_XML`.

W kliencie zostaje zarejestrowana implementacja klas `MessageBodyReader` i `MessageBodyWriter`. Szerzej na temat tych klas piszę w rozdziale 2.

W poniższej tabeli znajduje się zestawienie opisanych do tej pory najważniejszych klas i adnotacji JAX-RS.

Nazwa	Opis
`javax.ws.rs.Path`	Określa ścieżkę URI, dla której zasób serwuje metodę.
`javax.ws.rs.ApplicationPath`	Jest używana przez podklasę klasy `Application` jako bazowy URL wszystkich identyfikatorów URI dostarczanych przez zasoby w aplikacji.
`javax.ws.rs.Produces`	Definiuje typ mediów, jaki może zostać zwrócony przez dany zasób.
`javax.ws.rs.Consumes`	Definiuje typ mediów przyjmowany przez zasób.
`javax.ws.rs.client.Client`	Definiuje punkt wejściowy dla żądań klienta.
`javax.ws.rs.client.WebTarget`	Definiuje cel zasobu identyfikowany przez URI.

Klienty to ciężkie obiekty do obsługi infrastruktury komunikacyjnej po stronie klienta. Ponieważ ich tworzenie i usuwanie to dość czasochłonne operacje, powinno się tworzyć jak najmniej tych obiektów. Ponadto egzemplarz klienta zawsze trzeba poprawnie zamknąć, aby nie dopuścić do wycieku zasobów.

Uzyskiwanie dostępu do zasobów RESTful

W tym podrozdziale znajduje się opis różnych sposobów uzyskiwania dostępu do zasobów REST i testowania ich przez klienty.

cURL

cURL to popularne narzędzie wiersza poleceń do testowania API REST. Za jego pomocą użytkownik może tworzyć żądania, wysyłać je do API i analizować otrzymane odpowiedzi. Poniżej znajduje się parę przykładowych żądań `curl` wykonujących podstawowe czynności:

Żądanie curl	Opis
`curl http://api.foo.com/v1/coffees/1`	Proste żądanie `GET`.
`curl -H "foo:bar" http://api.foo.com/v1/coffees`	Żądanie z dodatkiem nagłówka za pomocą parametru `-H`.
`curl -i http://api.foo.com/v1/coffees/1`	Żądanie z wyświetleniem nagłówków odpowiedzi HTTP za pomocą parametru `-i`.
`curl –X POST -d'{"name":"JanKowalski", "username":"jkow","phone":"412-344-5644"} http://api.foo.com/v1/users`	Żądanie metodą `POST` utworzenia nowego użytkownika.

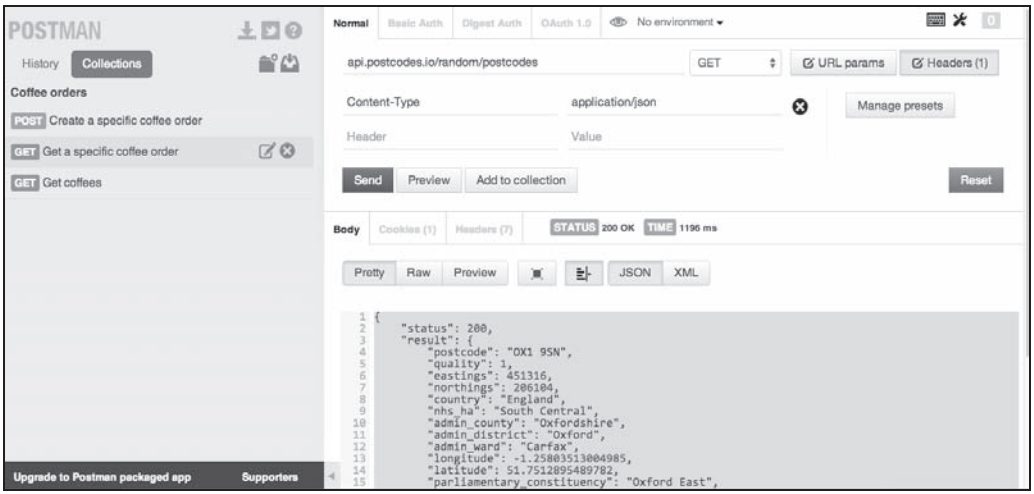
Choć narzędzie cURL jest bardzo pomocne, ma wiele opcji, które trzeba zapamiętać. Dlatego czasami lepszym rozwiązaniem jest użycie narzędzia przeglądarkowego, np. Postman albo Advanced REST client.

Postman

Postman dla przeglądarki Chrome to doskonałe narzędzie do testowania i rozwijania API REST. Zawiera przeglądarkę danych w formatach JSON i XML oraz umożliwia podglądanie żądań HTTP 1.1, jak również ich wielokrotne wysyłanie i zapisywanie na przyszłość. Postman działa w środowisku przeglądarki internetowej i umożliwia też przeglądanie danych cookie.

Zaletą narzędzia Postman w porównaniu z cURL jest przyjazny interfejs użytkownika do wprowadzania parametrów, dzięki czemu nie trzeba wpisywać całych poleceń ani skryptów. Ponadto program ten obsługuje różnego rodzaju metody uwierzytelniania, takie jak uwierzytelnianie podstawowe czy przy użyciu skrótów (tzw. *digest access authentication*).

Poniżej znajduje się zrzut ekranu przedstawiający sposób wysyłania zapytań w narzędziu Postman.



Na powyższym zrzucie ekranu przedstawiono okno aplikacji Postman. Najprostszym sposobem na przetestowanie tego programu jest uruchomienie go w przeglądarce Chrome.

Następnie należy wybrać metodę HTTP `GET` i wpisać adres URL `api.postcodes.io/random/postcodes`. (PostCodes to darmowa otwarta usługa, której działanie opiera się na danych geograficznych).

Otrzymasz odpowiedź JSON podobną do poniższej:

```
{
    "status": 200,
    "result": {
        "postcode": "OX1 9SN",
        "quality": 1,
        "eastings": 451316,
        "northings": 206104,
        "country": "England",
        "nhs_ha": "South Central",
        "admin_county": "Oxfordshire",
        "admin_district": "Oxford",
        "admin_ward": "Carfax",
...}
}
```

Po lewej stronie okna znajdują się różne zapytania, które zostały dodane do kolekcji, np. pobranie wszystkich zamówień kawy, pobranie jednego konkretnego zamówienia, utworzenie zamówień itd. na podstawie różnych przykładów z tej książki. Możesz też tworzyć własne kolekcje zapytań.

Pobieranie przykładów kodu

Pliki z przykładowym kodem źródłowym można pobrać z serwera FTP wydawnictwa Helion, pod adresem *ftp://ftp.helion.pl/przyklady/restja.zip*.

Więcej informacji na temat narzędzia Postman znajduje się na stronie *http://www.getpostman.com/*.

Inne narzędzia

Oto parę innych narzędzi, które również mogą być przydatne w pracy z zasobami REST.

Advanced REST client

Advanced REST client to kolejne rozszerzenie przeglądarki Chrome oparte na Google Web-Toolkit i służące do testowania oraz tworzenia API REST.

JSONLint

JSONLint to proste internetowe narzędzie do sprawdzania poprawności danych w formacie JSON. Gdy wysyła się dane w tym formacie, dobrze jest sprawdzić, czy są sformatowane zgodnie ze specyfikacją. Można to zrobić właśnie za pomocą narzędzia JSONLint. Więcej informacji na jego temat znajduje się na stronie *http://jsonlint.com/*.

Najlepsze praktyki projektowania zasobów

W tym podrozdziale znajduje się opis niektórych najlepszych praktyk projektowania zasobów RESTful:

- Programista API powinien używać rzeczowników, aby ułatwić użytkownikowi poruszanie się po zasobach, a czasowników tylko jako metod HTTP. Na przykład URI `/user/1234/books` jest lepszy niż `/user/1234/getBook`.
- Do identyfikacji podzasobów używaj asocjacji. Na przykład aby pobrać autorów książki 5678 dla użytkownika 1234, powinno się użyć URI `/user/1234/books/5678/authors`.
- Do pobierania specyficznych wariacji używaj parametrów zapytań. Na przykład aby pobrać wszystkie książki mające 10 recenzji, użyj URI `/user/1234/books?reviews_counts=10`.
- W ramach parametrów zapytań w razie możliwości zezwalaj na częściowe odpowiedzi. Przykładem może być pobranie tylko nazwy i wieku użytkownika. Klient może wysłać w URI parametr zapytania `?fields` zawierający listę pól, które chce otrzymać od serwera, np. `/users/1234?fields=name,age`.

- Zdefiniuj domyślny format odpowiedzi na wypadek, gdyby klient nie podał, jaki format go interesuje. Większość programistów jako domyślnego formatu używa JSON.
- W nazwach atrybutów stosuj `notacjęWielbłądzią` lub ze znakami podkreślenia _.
- Dla kolekcji zapewnij standardowe API liczące, np. `users/1234/books/count`, aby klient mógł sprawdzić, ilu obiektów może się spodziewać w odpowiedzi.

  Będzie to też pomocne dla klientów używających stronicowania. Szerzej na temat stronicowania piszę w rozdziale 5.
- Zapewnij opcję eleganckiego drukowania — `users/1234?pretty_print`. Ponadto nie powinno się buforować zapytań z parametrem drukowania.
- Staraj się minimalizować komunikację przez dostarczenie jak najpełniejszych informacji w pierwszej odpowiedzi. Chodzi o to, że jeśli serwer nie dostarczy wystarczającej ilości danych w odpowiedzi, klient będzie musiał wysłać kolejne żądania, aby zdobyć potrzebne mu informacje. W ten sposób marnuje się zasoby i wyczerpuje limit żądań klienta. Szerzej na temat ograniczania liczby żądań klienta piszę w rozdziale 5.

Zalecana lektura

- RFC 2616: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html*.
- Model dojrzałości Richardsona: *http://www.crummy.com/writing/speaking/2008-QCon/act3.html*.
- Implementacja JAX-RS Jersey: *https://jersey.java.net/*.
- InspectB.in: *http://inspectb.in/*.
- Postman: *http://www.getpostman.com/*.
- Advanced REST Client: *https://code.google.com/p/chrome-rest-client/*.

Podsumowanie

W tym rozdziale przedstawiłam podstawowe założenia technologii REST, opisałam API CRUD oraz pokazałam, jak projektować zasoby RESTful. W przykładach użyte zostały adnotacje JAX-RS 2.0, za pomocą których można reprezentować metody HTTP, i API klienckie, przy użyciu których można odnosić się do zasobów. Ponadto zrobiłam przegląd najlepszych praktyk projektowania usług typu RESTful.

W następnym rozdziale znajduje się rozszerzenie tych wiadomości. Bardziej szczegółowo poznasz zasady negocjowania treści, dostawców jednostek w JAX-RS 2.0, techniki obsługi błędów, sposoby kontrolowania wersji oraz kody odpowiedzi REST. Ponadto dowiesz się, w jaki sposób serwer może wysyłać do klienta odpowiedzi przy użyciu strumieniowania i kawałkowania.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

W

X

Z

Ż

REST
Najlepsze praktyki i wzorce w języku Java

Postępująca cyfryzacja współczesnego świata wymaga coraz większej integracji przeróżnych systemów informatycznych. Nierzadko są to systemy znajdujące się w odległych lokalizacjach, napisane z wykorzystaniem różnych języków programowania i technologii. Jak zapewnić wymianę informacji między nimi? Usługi typu REST to jeden z najwygodniejszych sposobów!

Co to jest REST? Jak przygotować usługę tego typu? Jak ustrzec się przed błędami? Na te i wiele innych pytań odpowiada ta książka. Znajdziesz w niej najlepsze praktyki tworzenia usług REST z wykorzystaniem języka Java. Sięgnij po nią i dowiedz się, jak projektować zasoby i zapewniać im bezpieczeństwo oraz w jaki sposób przygotowywać usługi REST dla różnych wersji językowych. Odkryj, jak testować udostępnione zasoby i zagwarantować ich najwyższą wydajność oraz jaka przyszłość czeka usługi tego typu. Sprawdź też, jak wygląda API takich usług, jak GitHub, Twitter i Facebook (API Graph). Ta książka jest doskonałą lekturą dla wszystkich programistów chcących bezproblemowo tworzyć wydajne usługi typu REST.

Wydajne usługi REST w Twoim zasięgu!

Dzięki tej książce:

- poznasz zasady projektowania usług typu REST
- wykorzystasz zaawansowane mechanizmy uwierzytelnienia i autoryzacji
- zapewnisz Twoim usługom najwyższą wydajność
- poznasz API popularnych serwisów

Bhakti Mehta — od ponad 13 lat projektuje rozwiązania oparte na platformie Java EE. Pracuje jako starszy inżynier oprogramowania w firmie Blue Jeans Network. Jest odpowiedzialna za tworzenie usług REST, ich wydajność oraz skalowalność. Często występuje w roli prelegentki, jest autorką licznych artykułów publikowanych w serwisach branżowych.

[PACKT] open source PUBLISHING community experience distilled

Helion

32960 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-0644-8

Informatyka w najlepszym wydaniu

cena: 34,90 zł