Zagadnienia Programowania Obiektowego

Usługi sieciowe w Javie EE 5

Adam Maciejewski

21 maja 2007

Usługi sieciowe w Javie EE 5

1. Wprowadzenie
   - Co to i po co
   - Jak było dawniej
2. Java EE 5: JAX–WS 2.0
   - Podstawy
   - Co można skonfigurować
3. Co jeszcze można zrobić
   - Po stronie serwera
   - Po stronie klienta
4. JAX–WS od środka
   - Architektura JAX–WS
5. Ułatwienia
   - Narzędzia i IDE
6. Zakończenie

Co to jest usługa sieciowa?

Web service

Dostępny zdalnie (poprzez sieć) komponent programowy, realizujący funkcje wyszczególnione w opisanym w języku WSDL interfejsie, niezależny od wykorzystywanej platformy. Wymiana danych pomiędzy klientem a serwerem odbywa się zazwyczaj za pośrednictwem protokołu SOAP.

Dlaczego ludzie chcą to mieć?

- Wygodny, elastyczny sposób tworzenia aplikacji rozproszonych
- Dobre rozdzielenie abstrakcyjnego interfejsu aplikacji od jego konkretnej implementacji
- Niezależność od wykorzystywanej platformy, zarówno po stronie serwera jak i klienta
- Łatwość integrowania ze sobą aplikacji pochodzących od różnych dostawców
- Usługi sieciowe to jeden z podstawowych elementów tzw. Service–Oriented Architecture

Na samym początku...

- Najpierw trzeba było napisać w WSDL–u definicję usługi
- Potem jakoś wygenerować jej implementację w Javie
- Producenci serwerów aplikacji dostarczali do tego swoje własne narzędzia (każdy inne) ...
- ...mogliśmy również skorzystać z Apache Axis (http://ws.apache.org/axis/)

J2EE 1.4: JAX–RPC 1.1

- Piszemy w Javie interfejs dla naszej usługi (musi dziedziczyć po `java.rmi.Remote`) oraz klasę, która go implementuje
- W XML–owym deskryptorze podajemy nazwę usługi, docelową przestrzeń nazw oraz wskazujemy interfejs i pakiet zawierający jego implementację
- Narzędzie `wscompile` wygeneruje nam odpowiedni plik WSDL oraz mapowania typów
- Można też w drugą stronę (WSDL → Java)

JAX–WS 2.0: najprostsza usługa

Kodujemy klasę implementującą naszą usługę, opatrujemy ją adnotacją `javax.jws.WebService` ... i gotowe

```
package foo;

import javax.jws.WebService;

@WebService
public class Foo {

    @WebMethod
    public String foo(String arg) {
        return "Foo (" + arg + ")!";
    }
}
```

JAX–WS 2.0: publikowanie usługi

Możemy skorzystać ze statycznej metody `Endpoint.publish()`, by umieścić naszą usługę pod konkretnym adresem ...

```
import javax.xml.ws.Endpoint;

public static void main(String[] args) {
    Endpoint.publish("http://localhost:8080/foo",
        new Foo());
}
```

JAX–WS 2.0: publikowanie usługi

Możemy skorzystać ze statycznej metody `Endpoint.publish()`, by umieścić naszą usługę pod konkretnym adresem . . .

```
import javax.xml.ws.Endpoint;

public static void main(String[] args) {
    Endpoint.publish("http://localhost:8080/foo",
         new Foo());
}
```

. . . ale jest to konieczne tylko wtedy, gdy pracujemy poza serwerem aplikacji (inaczej wyręczy nas kontener EJB albo serwletów — docelowy adres podamy w fazie umieszczania aplikacji na serwerze)

JAX–WS 2.0: najprostszy klient

Korzystając z mechanizmu wstrzykiwania zależności, pobieramy odniesienie do naszej usługi ...

```
@WebServiceRef(wsdlLocation =
    "http://localhost:8080/foo")
static FooService service;
```

JAX–WS 2.0: najprostszy klient

Korzystając z mechanizmu wstrzykiwania zależności, pobieramy odniesienie do naszej usługi . . .

```
@WebServiceRef(wsdlLocation =
    "http://localhost:8080/foo")
static FooService service;
```

. . . wydobywamy z niego *port* — reprezentuje on po stronie klienta interfejs usługi

```
Foo port = service.getFooPort();
```

JAX–WS 2.0: najprostszy klient

Korzystając z mechanizmu wstrzykiwania zależności, pobieramy odniesienie do naszej usługi ...

```
@WebServiceRef(wsdlLocation =
    "http://localhost:8080/foo")
static FooService service;
```

... wydobywamy z niego *port* — reprezentuje on po stronie klienta interfejs usługi

```
Foo port = service.getFooPort();
```

... za jego pośrednictwem możemy wywoływać metody dostarczane przez usługę

```
String response = port.foo("bar");
```

Adnotacje i ich parametry

@WebService

name
: typ portu (`wsdl:portType`), domyślnie nazwa klasy

targetNamespace
: przestrzeń nazw pliku WSDL, domyślnie generowana przez JAXB z nazwy pakietu

serviceName
: nazwa usługi, domyślnie nazwa klasy + „Service”

endpointInterface
: wskazuje na Service Endpoint Interface usługi, jeśli puste, JAX–WS sam go wygeneruje

portName
: wartość `wsdl:portName`, domyślnie `WebService.name` + „Port”

Adnotacje i ich parametry, c.d.

@WebMethod

operationName
: wartość `wsdl:operationName` dla tej metody, domyślnie nazwa metody w klasie

exclude
: czy wykluczyć daną metodę z usługi, domyślnie `false`

action
: wartość SOAP Action

Adnotacje i ich parametry, c.d.

@WebMethod

operationName wartość `wsdl:operationName` dla tej metody, domyślnie nazwa metody w klasie

exclude czy wykluczyć daną metodę z usługi, domyślnie `false`

action wartość SOAP Action

@WebServiceRef

wsdlLocation adres pliku WSDL opisującego usługę

name, type nazwa JNDI i typ wstrzykiwanego zasobu, domyślnie takie jak dla danego pola/własności

Adnotacje i ich parametry, c.d.

@BindingType

value
: napis będący URL–em określającym protokół przesyłu komunikatów, np. (z javax.xml.ws.soap) SOAPBinding.SOAP11HTTP_BINDING [domyślnie], SOAPBinding.SOAP12HTTP_BINDING czy javax.xml.ws.http.HTTPBinding.HTTP_BINDING

Oprócz tego możemy do klasy implementującej naszą usługę dodawać obserwatorów — służą do tego adnotacje `@PostConstruct` i `@PreDestroy`

Usługi w kontenerze webowym lub EJB

- Usługi sieciowe można implementować jako ziarenka EJB: klasy takie muszą spełniać wszystkie wymagania dla Stateless Session Beans, a oprócz tego mieć adnotację `@WebService` lub `@WebServiceProvider`

Usługi w kontenerze webowym lub EJB

- Usługi sieciowe można implementować jako ziarenka EJB: klasy takie muszą spełniać wszystkie wymagania dla Stateless Session Beans, a oprócz tego mieć adnotację `@WebService` lub `@WebServiceProvider`
- ... albo jako serwlety; jeśli chcemy zezwolić na dostęp przez tylko jeden wątek naraz, musimy zaimplementować interfejs `javax.servlet.SingleThreadModel` (uznany w ostatniej specyfikacji serwletów za przestarzały...)
- Również w serwlecie nie wolno przechowywać żadnego stanu

Usługi w kontenerze webowym lub EJB

- Usługi sieciowe można implementować jako ziarenka EJB: klasy takie muszą spełniać wszystkie wymagania dla Stateless Session Beans, a oprócz tego mieć adnotację `@WebService` lub `@WebServiceProvider`
- ... albo jako serwlety; jeśli chcemy zezwolić na dostęp przez tylko jeden wątek naraz, musimy zaimplementować interfejs `javax.servlet.SingleThreadModel` (uznany w ostatniej specyfikacji serwletów za przestarzały...)
- Również w serwlecie nie wolno przechowywać żadnego stanu
- W obu przypadkach publikacją usługi zajmuje się kontener; specyfikacja wymaga, by próba użycia `Endpoint.publish()` z wewnątrz kontenera kończyła się odmową dostępu

Podejście niskopoziomowe

Interfejs javax.xml.ws.Provider

- Czasamy możemy chcieć zejść aż do poziomu wiadomości XML–owych, które odbiera i wysyła nasza usługa
- JAX–WS daje nam taką możliwość: wystarczy zaimplementować interfejs `Provider<T>` (zawsze mamy przynajmniej `Provider<Source>` oraz `Provider<SOAPMessage>`)
- Zwawiera on jedną metodę: `T invoke(T request)` — w niej możemy robić wszystko
- Musimy tylko pamiętać, by przy naszej klasie była adnotacja `@WebServiceProvider`

Podejście niskopoziomowe, c.d.

Adnotacja javax.xml.ws.WebServiceProvider

@WebServiceProvider

portName	wartość `wsdl:portName`
serviceName	nazwa usługi
targetNamespace	przestrzeń nazw pliku WSDL
wsdlLocation	adres pliku WSDL opisującego usługę

(wszystkie atrybuty opcjonalne, jeśli ich nie podamy, zostaną użyte wartości z deskryptora)

Generowanie kodu w oparciu o WSDL

- Podobnie jak JAX–RPC, także JAX–WS daje nam do tego odpowiednie narzędzie — tym razem nazywa się ono `wsimport`
- Za jego pomocą automatycznie wygenerujemy z pliku WSDL interfejs(y) opisujące naszą usługę; pozostaje nam tylko napisać implementację i pamiętać o adnotacji `@WebService(endpointInterface=...)`
- Nie potrzebujemy (jak to było w JAX–RPC) żadnych dodatkowych plików konfiguracyjnych

Dostęp do usługi poprzez JNDI

Zamiast korzystać z wstrzykiwania zależności, możemy wyjąć interfejs naszej usługi z kontekstu:

```
InitialContext ic = new InitialContext ();
Service foo = (Service) ic.lookup(
        "java:comp/env/service/FooService");
```

albo precyzyjniej:

```
InitialContext ic = new InitialContext ();
FooService foo = (FooService) ic.lookup(
        "java:comp/env/service/FooService");
```

Z JAX–WS usunięto klasę `ServiceFactory`; zamiast niej należy korzystać właśnie z JNDI albo adnotacji `@WebServiceRef`

Dynamiczny klient
Dispatch API

Do wywołania metody getPort() z klasy Service potrzebna jest nam znajomość opisu WSDL usługi (musi być wskazany w deskryptorze klienta). W przeciwnym razie musimy użyć działającego na poziomie wiadomości XML Dispatch API

```
Service serv = Service.create(serviceName);
serv.addPort(portName, bindingType, endpointUrl);
Dispatch disp = serv.createDispatch(portName,
        msgClass, serviceMode);
```

gdzie msgClass to klasa obiektów wiadomości (np. Source lub SOAPMessage), a serviceMode to albo Service.Mode.MESSAGE, albo Service.Mode.PAYLOAD

Dynamiczny klient, c.d.

Mając obiekt typu Dispatch, musimy teraz sami przygotować wiadomość z żądaniem, a następnie wywołać usługę:

```
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage request = mf.createMessage();
 ...
SOAPMessage response = disp.invoke(request);
// disp.invokeOneWay(request);
```

Dynamiczny klient, c.d.

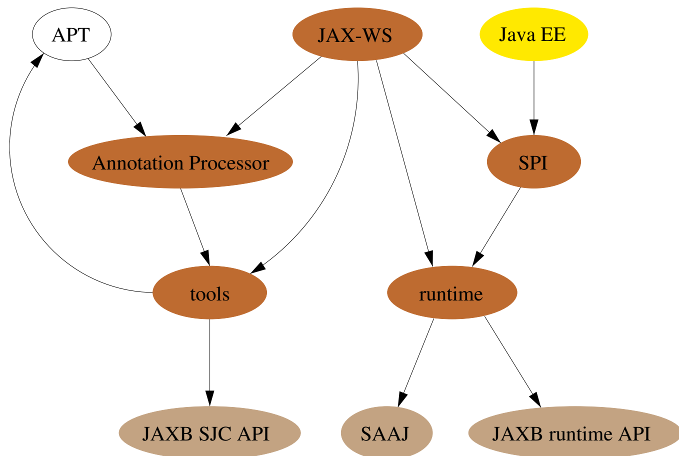
Mając obiekt typu Dispatch, musimy teraz sami przygotować wiadomość z żądaniem, a następnie wywołać usługę:

```
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage request = mf.createMessage();
 ...
SOAPMessage response = disp.invoke(request);
// disp.invokeOneWay(request);
```

Możemy też wywołać naszą usługę w sposób asynchroniczny, w trybie *polling* albo *callback*:

```
Response<T> response = dispatch.invokeAsync(T);
Future<?> response =
        dispatch.invokeAsync(T, AsyncHandler);
```

JAX–WS a inne elementy Javy EE 5



Części składowe JAX–WS

runtime
: główny moduł zapewniający całą funkcjonalność JAX–WS potrzebną w czasie działania naszej aplikacji

tools
: narzędzia (`wsgen`, `wsimport`) służące do przekształcania plików WSDL i klas Javy w gotowe do uruchomienia usługi sieciowe

Annotation Processing
: klasy do przetwarzania adnotacji zdefiniowanych w pakiecie `javax.jws`

SPI
: (Service Provider Interface) zbiór interfejsów i klas abstrakcyjnych określających zasady współpracy pomiędzy JAX–WS a pozostałymi składnikami Javy EE 5

Wykorzystywane biblioteki

SAAJ (SOAP with Attachments API for Java) — do tworzenia, wysyłania, odbierania i dekodowania wiadomości SOAP (wiadomości odbierane są jako strumienie w obiektach HttpServletRequest)

JAXB (Java Architecture for XML Binding) — do serializacji i deserializacji obiektów Javy do/z XML; w JAX–RPC stosowane były do tego własne mechanizmy

IDE, wtyczki, itp.

Eclipse

- W Eclipse Web Tools nie ma i jeszcze przez jakiś czas nie będzie; w wersji 2.0 obsługa JAX–WS na pewno już się nie pojawi (z nowych rzeczy ma być tylko Apache Axis2)
- Wsparcie dla JAX–WS ma być w SOA Tools Platform (`http://www.eclipse.org/stp/`), na razie w wersji 0.4.0

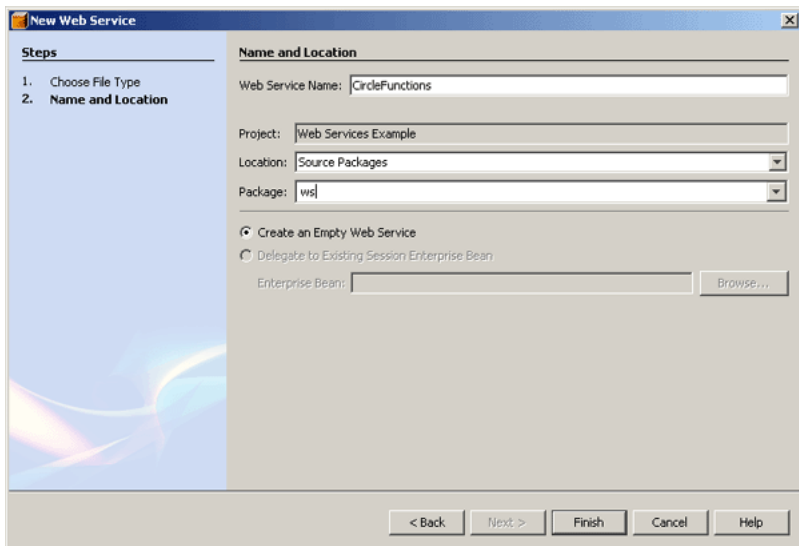
Maven

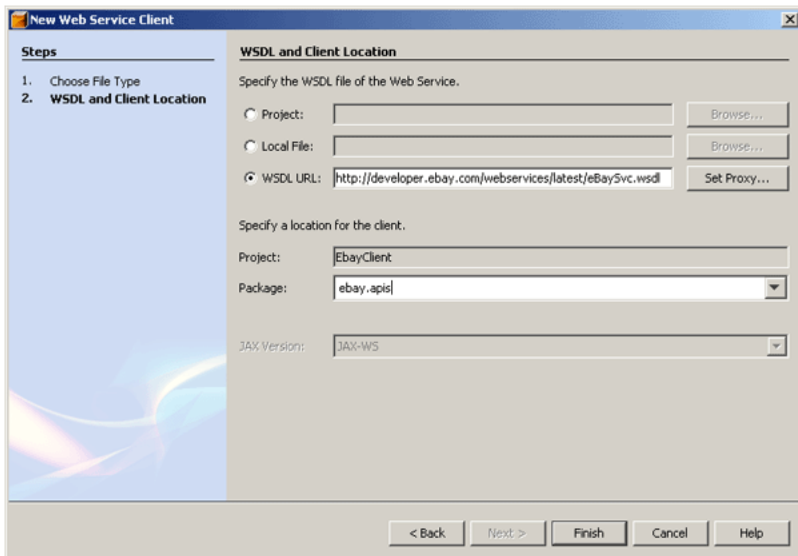
- Pod adresem `https://jax-ws-commons.dev.java.net/` dostępna jest wtyczka umożliwiająca korzystanie z narzędzi `wsgen` i `wsimport`

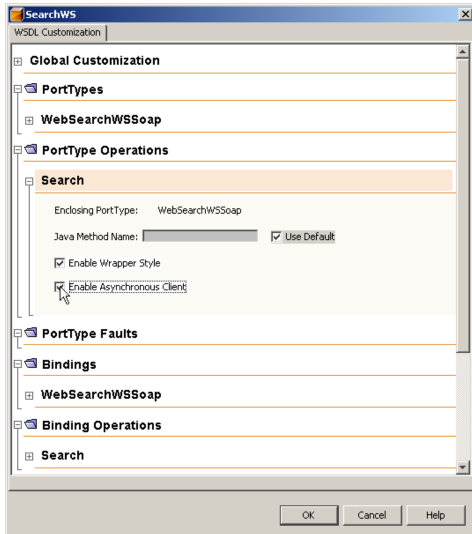
IDE, wtyczki, itp.

NetBeans

- Obsługa JAX–WS pojawiła się w wersji 5.0, w 5.5 ulepszona i uzupełniona; dalsze rozszerzenia w Enterprise Packu
- Kreatory do tworzenia projektów aplikacji implementujących i korzystających z usług sieciowych, automatyczne generowanie kodu (w tym dla wywołań asynchronicznych!), narzędzia ułatwiające konfigurowanie aplikacji, . . .


New Web Service
Steps
1. Choose File Type
2. Name and Location
Name and Location
Web Service Name: CircleFunctions
Project: Web Services Example
Location: Source Packages
Package: ws
Create an Empty Web Service
Delegate to Existing Session Enterprise Bean
Enterprise Bean:
Browse...
< Back
Next >
Finish
Cancel
Help


New Web Service Client
Steps
1. Choose File Type
2. WSDL and Client Location
WSDL and Client Location
Specify the WSDL file of the Web Service.
Project:
Browse...
Local File:
Browse...
WSDL URL:
http://developer.ebay.com/webservices/latest/eBaySvc.wsdl
Set Proxy...
Specify a location for the client.
Project:
EbayClient
Package:
ebay.apis
JAX Version:
JAX-WS
< Back
Next >
Finish
Cancel
Help


SearchWS
WSDL Customization
Global Customization
PortTypes
WebSearchWSSoap
PortType Operations
Search
Enclosing PortType: WebSearchWSSoap
Java Method Name:
Use Default
Enable Wrapper Style
Enable Asynchronous Client
PortType Faults
Bindings
WebSearchWSSoap
Binding Operations
Search
OK
Cancel
Help

Powiązane technologie

JAXR (Java API for XML Registries) — ułatwia publikowanie informacji o usługach i ich wyszukiwanie w tzw. *rejestrach*, np. w standardzie ebXML czy UDDI (Universal Description, Discovery, and Integration)

WSIT (Web Services Integration Tools) — API mające na celu ułatwienie współpracy klientów i serwerów usług sieciowych pomiędzy platformami Java EE oraz .NET

BPEL (Business Process Execution Language) — język do orkiestracji usług sieciowych (komponowania ich w bardziej złożone procesy biznesowe). Referencyjna implementacja Javy EE 5 (Glassfish) zawiera wbudowane środowisko do wykonywania takich procesów

Więcej informacji

JSR 109: Implementing Enterprise Web Services
http://www.jcp.org/en/jsr/detail?id=109

JSR 181: Web Services Metadata for the Java Platform
http://www.jcp.org/en/jsr/detail?id=181

Strona referencyjnej implementacji JAX–WS
https://jax-ws.dev.java.net/

Opis architektury JAX–WS
https://jax-ws-architecture-document.dev.java.net/

Introducing JAX–WS with Java SE
http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/index.html

Więcej informacji, c.d.

JAX–WS Dispatch and Provider APIs
http://java.sun.com/mailers/techtips/enterprise/2006/TechTips_July06.html

Writing a Handler in JAX–WS
http://java.sun.com/mailers/techtips/enterprise/2006/TechTips_June06.html

Implementing SOA with Java EE 5
http://java.sun.com/developer/technicalArticles/WebServices/soa3/

Web Services Interoperability Technology
http://java.sun.com/webservices/interop/