I Seminarium PLOUG
Warszawa
Marzec 2001

Java Server Pages: koncepcje i zastosowania

Marek Wojciechowski
Marek.Wojciechowski@cs.put.poznan.pl
Maciej Zakrzewicz
Maciej.Zakrzewicz@cs.put.poznan.pl

Politechnika Poznańska, Instytut Informatyki

Autorzy

Są pracownikami Instytutu Informatyki Politechniki Poznańskiej.

Streszczenie

Java Server Pages (JSP), najnowsze wcielenie popularnej technologii serwletów Java, jest mechanizmem dynamicznego generowania dokumentów WWW, umożliwiającym tworzenie przenaszalnych aplikacji internetowych. JSP rozwiązuje zasadniczy problem serwletów Java: pozwala odseparować kod proceduralny (JavaBeans) od definicji szaty graficznej generowanego dokumentu WWW. Technologia ta stanowi interesującą alternatywę dla innych metod dynamicznego generowania dokumentów WWW po stronie serwera: CGI, ASP, PL/SQL Cartridge, PERL Cartridge.
W artykule przedstawiono podstawowe zasady konstruowania bazodanowych aplikacji JSP i ich wykorzystania na platformie Oracle8i. Wiele omawianych rozwiązań zostało zilustrowanych przykładami.

1 Wstęp

W ciągu kilku ostatnich lat obserwuje się radykalną zmianę znaczenia słowa „Internet": początkowo rozumiany jako zbiór prostych usług sieciowych, umożliwiających głównie przesyłanie wiadomości, plików i stron HTML, dziś odbierany jako zaawansowana platforma aplikacyjna, wywłaszczająca dotychczas tak popularne systemy klient-serwer. Technologie wykorzystywane do budowy aplikacji Internetowych można podzielić na (1) technologie klienta i (2) technologie serwera. Technologie klienta sa w pełni implementowane w oparciu o funkcje przeglądarki WWW (HTML, XML, applety Java, CSS, JavaScript, DHTML, itp.). Technologie serwera są głównie wykorzystywane do dynamicznego generowania dokumentów i innych obiektów, wysyłanych następnie do klienta. Najpopularniejsze technologie serwera to CGI, ASP, PL/SQL Cartridge, serwlety Java, JSP.

W artykule przedstawiono główne koncepcje i zastosowania technologii Java Server Pages (JSP). Struktura artykułu jest następująca. Rozdział 1 stanowi wprowadzenie do programowania serwletów Java i wykorzystywania JDBC do współpracy z bazami danych. Rozdział 2 prezentuje architekturę i techniki implementacji dokumentów JSP oraz wykorzystywania komponentów JavaBeans. Rozdział 3 zawiera uwagi końcowe.

1.1 Serwlety Java

Serwlet jest programem w języku Java, składowanym w postaci skompilowanej w systemie plików serwera WWW. Każdy serwlet jest implementowany w formie klasy dziedziczącej z `HttpServlet` (`javax.servlet.http.HttpServlet`), zawierającej definicję metody o nazwie `doGet()` (w niektórych przypadkach `doPost()`, `doPut()` lub `doDelete()`). Zadaniem tej metody, a zarazem zadaniem całego serwletu, jest wygenerowanie kompletnego dokumentu HTML (XML, WML, itp.) i przekazanie go serwerowi WWW. Na żądanie użytkownika, wyrażone w formie adresu URL, serwer WWW wykonuje kod serwletu i przesyła użytkownikowi wygenerowany dokument HTML. Przykład prostego serwletu oraz wynik jego działania zostały przedstawione na rys. 1.

```
(1) import java.io.*;
(2) import javax.servlet.*;
(3) import javax.servlet.http.*;
(4) public class MyServlet extends HttpServlet {
(5)   public void doGet(HttpServletRequest request,
(6)                     HttpServletResponse response)
(7)      throws ServletException, IOException {

(8)   response.setContentType("text/html");
(9)   PrintWriter out = response.getWriter();
(10)  out.println("<H1>Moj serwlet!</H1>");
(11)  }
(12) }
```

Rys. 1 Przykład serwletu Java

W wierszach 1-3 serwlet deklaruje klasy biblioteczne wykorzystywane w dalszej części implementacji. W wierszu 4 rozpoczyna się definicja klasy serwletu, dziedziczącej z HttpServlet. Wiersze 5 i 6 zawierają deklarację nagłówka metody doGet(). Metoda doGet() nadpisuje oryginalną metodę klasy HttpServlet (javax.servlet.http. HttpServlet) o takiej samej nazwie i w związku z tym musi stosować identyczny nagłówek. Metoda otrzymuje dwa argumenty wywołania: request (typu javax.servlet.http. HttpServletRequest) i response (typu javax.servlet.http. HttpServletResponse). Są to argumenty obiektowe, reprezentujące jednokierunkowe kanały komunikacyjne do serwera WWW. Obiekt request to kanał od serwera WWW do serwletu (m.in. parametry wywołania), natomiast response to kanał od serwletu do serwera WWW (m.in. generowany dokument HTML). W wierszu 7 zostały zadeklarowane dwa wyjątki, jakie mogą być wygenerowane podczas

pracy serwletu – w omawianym rozwiązaniu wyjątki te nie są obsługiwane lokalnie lecz są propagowane do środowiska wywołującego (serwer WWW). Wiersz 8 powoduje wysłanie do serwera WWW, a tym samym do przeglądarki użytkownika, deklaracji typu generowanego dokumentu (dokument HTML). Gdyby serwlet generował kod XML, WML, GIF, JPEG, itd., wtedy należałoby użyć stosownego kodu typu dokumentu (kody MIME). Wiersz 9 to pobranie uchwytu połączenia tekstowego z przeglądarką użytkownika. Uchwyt ten jest w dalszej części implementacji dostępny w postaci obiektu out klasy PrintWriter (java.io.PrintWriter). Dopiero w wierszu 10 rozpoczyna się faktyczny (dotyczący użytkownika) kod serwletu. Wywołanie metody println obiektu out powoduje wysłanie podanego jako argument tekstu (kod HTML) do przeglądarki użytkownika, która z kolei wyświetla otrzymany dokument na ekranie.

Podstawową zaletą serwletów Java jest ich przenaszalność. Zaimplementowany i skompilowany serwlet może być wykorzystywany przez dowolny serwer WWW, pod warunkiem, że serwer ten wspiera technologię Java. Ponadto, w przeciwieństwie do rozwiązań konkurencyjnych (Perl CGI, ASP, PL/SQL Cartridge), serwlety wykorzystują bardzo popularny język programowania (Java), dzięki czemu programiści mogą bazować na swoich dotychczasowych doświadczeniach.

Technologia serwletów nie jest jednocześnie pozbawiona wad. Język Java słynie z niskiej wydajności, co po części odbija się na efektywności serwletów. Pewnym rozwiązaniem jest możliwość skorzystania z kompilatorów JIT (Just-in-time Compilers). Technika ta polega na automatycznej kompilacji serwletu do kodu maszynowego w chwili jego pierwszego uruchomienia. Odbywa sie to w sposób niezauważalny dla użytkownika (jeśli pominąć „zauważalne” opóźnienie podczas pierwszego odwołania). Drugą wadą technologii serwletów jest konieczność zanurzania kodu HTML wewnątrz kodu Java. Gdy serwlet jest używany do generowania skomplikowanych graficznie dokumentów, to w kodzie źródłowym umieszcza się setki wierszy `out.println(...)`. Pociąga to za sobą istotne trudności w utrzymywaniu szaty graficznej dokumentów. Każda modyfikacja formy prezentacji dokumentu oznacza rekompilację kodu źródłowego serwletu.

1.2 Java Database Connectivity (JDBC)

JDBC jest standardową biblioteką Java, umożliwiającą dowolnym programom Java współpracę z dowolnymi systemami baz danych. JDBC jest powszechnie wykorzystywane przez serwlety Java do generowania dokumentów HTML w oparciu o zawartość bazy danych. W szczególności, JDBC pozwala programiście na realizację następujących operacji:

1. Nawiązanie połączenia z bazą danych
2. Wysyłanie treści poleceń SQL do serwera bazy danych
3. Przetwarzanie wyników zapytań SQL

Biblioteka JDBC zawiera kilka klas, służących do implementacji wyżej wymienionych operacji (pakiet `java.sql`). Ogólna procedura, jaką stosuje programista w celu wykorzystania obiektów JDBC do współpracy z bazą danych jest nastepująca:

1. Zarejestrowanie wybranego sterownika JDBC w środowisku maszyny wirtualnej Java

   Ponieważ JDBC jest interfejsem o charakterze uniwersalnym, dlatego dodatkowo program Java musi korzystać z tzw. sterownika JDBC. Sterownik JDBC jest modułem dostarczanym przez producenta, dołączanym do środowiska maszyny wirtualnej Java. Oracle zaleca stosowanie dwóch podtypów sterowników: JDBC Thin Java Driver (napisany w pełni w języku Java, samodzielnie obsługujący Net8) oraz JDBC OCI Java Driver (napisany częściowo w języku Java, wykorzystujący lokalną bibliotekę Net8). W celu zarejestrowania pakietu sterowników Oracle, należy wykonac poniższy kod:

```
DriverManager.registerDriver(neworacle.jdbc.driver.OracleDriver());
```

2. Utworzenie obiektu klasy `Connection`, reprezentującego połączenie z bazą danych. Fizyczne nawiązanie połączenia.

   Gdy wykorzystywany jest sterownik JDBC Thin Java Driver:

```
Connection conn;
conn = DriverManager.getConnection(
   "jdbc:oracle:thin:@150.254.31.47:1521:orcl",
      "scott","tiger");
```

   gdzie:
   - `150.254.31.47` - adres IP serwera bazy danych
   - `1521` - numer portu procesu listenera
   - `orcl` – nazwa instancji bazy danych
   - `scott` – nazwa użytkownika
   - `tiger` – hasło dostępu

   Gdy korzystamy ze sterownika JDBC OCI Java Driver:

```
Connection conn;
conn = DriverManager.getConnection(
   "jdbc:oracle:oci8:@baza1",
   "scott","tiger");
```

   gdzie:
   - `baza1` – nazwa bazy danych w Net8
   - `scott` – nazwa użytkownika
   - `tiger` – hasło dostępu

3. Utworzenie obiektu klasy `Statement`, reprezentującego polecenie SQL.
   Utworzony obiekt reprezentuje „pusty" bufor, który posłuży do zapisania treści polecenia SQL.

```
Statement stmt;
      stmt = conn.createStatement();
```

   gdzie:
   - `conn` – obiekt klasy `Connection`, reprezentujący połączenie z bazą danych

5. Wykonanie polecenia SQL. W przypadku zapytań, utworzenie obiektu klasy `ResultSet`, reprezentującego wynikowy zbiór rekordów.

   Dla zapytań:

```
ResultSet rset;
      rset = stmt.executeQuery("select ename,sal from emp");
```

   gdzie:
   - `stmt` – obiekt klasy `Statement`, reprezentujący bufor polecenia SQL

   Dla pozostałych poleceń:

```
stmt.executeUpdate("delete emp where ename='Smith'");
```

gdzie:
- `stmt` – obiekt klasy `Statement`, reprezentujący bufor polecenia SQL

6. W przypadku zapytań – przetwarzanie wynikowego zbioru rekordów.

```
while (rset.next()) {
  System.out.println(rset.getString("ename"));
  System.out.println(rset.getString("sal"));
}
```

gdzie:
- `rset` – obiekt klasy ResultSet, reprezentujący wynikowy zbiór rekordów
- `ename` – nazwa kolumny
- `sal` – nazwa kolumny
- `rset.getString()` – zwraca wartość podanej kolumny z bieżącego rekordu
- `rset.next()` – przesuwa kursor na kolejny rekord wyniku zapytania

7. Zamknięcie wykorzystywanych obiektów i rozłączenie sesji.

```
rset.close();
stmt.close();
conn.close();
```

W przypadku niepowodzenia ktoregoś z powyższych kroków, zgłaszany jest wyjątek `SQLException`. Poniżej przedstawiono przykładowy kod kompletnej aplikacji Java, ktora wykorzystuje JDBC w celu pobrania i wyświetlenia zawartości tabeli `EMP`.

```
import java.sql.*;

public class MyDemo {
   public static void main(String[] args)
   throws SQLException {
     Connection conn;
     Statement stmt;
     ResultSet rset;

     DriverManager.registerDriver(
      new oracle.jdbc.driver.OracleDriver());
     conn = DriverManager.getConnection(
        "jdbc:oracle:thin:@srv1:1521:orcl","scott","tiger");
     stmt = conn.createStatement();
     rset = stmt.executeQuery("select ename,sal from emp");
     while (rset.next()) {
        System.out.print(rset.getString("ename")+" ");
        System.out.println(rset.getString("sal"));
     }
     rset.close(); stmt.close(); conn.close();
}}
```

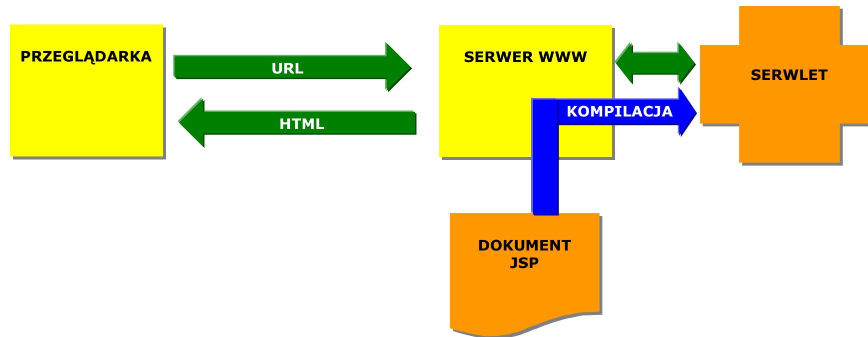
2 Dokumenty Java Server Pages (JSP)

2.1 Wprowadzenie

Technologia JSP jest odpowiedzią na podstawowy problem związany z wykorzystywaniem serwletów Java: brak izolacji pomiędzy kodem wykonywalnym a definicją szaty graficznej generowanego dokumentu. JSP nie jest całkowicie nowym rozwiązaniem – jest to rodzaj nakładki na popularną technologię serwletów Java. Tworzenie aplikacji JSP polega na przygotowywaniu tekstowych dokumentów HTML, wewnątrz których zanurzany jest kod Java, służący do konstruowania części dynamicznych. Dokumenty takie nazywane są dokumentami, lub stronami, JSP (Java

Server Pages). Gdy przeglądarka użytkownika żąda przesłania dokumentu JSP, wtedy serwer WWW wykonuje zanurzony kod Java i w odpowiedzi wysyła wynikowy dokument HTML.

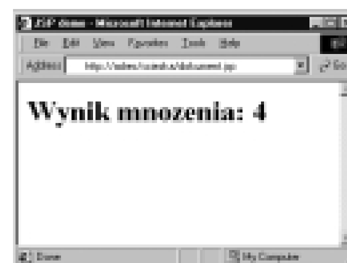
Proces przetwarzania żądania pobrania dokumentu JSP został przedstawiony na rys.2. Podczas pierwszego odwołania użytkownika do dokumentu JSP, serwer WWW przepisuje dokument do postaci serwletu, a nastepnie go kompiluje (wynik kompilacji zapisywany w systemie plików). Tak skompilowany serwlet generuje wynikowy dokument HTML. Kolejne odwołania są obsługiwane już przy wykorzystaniu serwletu, pod warunkiem, że oryginalny dokument JSP nie uległ zmianie.



Rys. 2 Architektura Java Server Pages

Przykład prostego dokumentu JSP oraz wynik jego działania zostały przedstawione na rys. 3. Elementy JSP najczęściej występują w postaci znaczników <%...%>. W wierszu 6 znajduje się deklaracja języka programowania wykorzystywanego wewnątrz dokumentu. Aktualnie Java jest jedynym obsługiwanym językiem, w związku z czym dyrektywa ta jest w większości implementacji opcjonalna. Wiersz 7 ilustruje sposób deklaracji zmiennych globalnych dla dokumentu. Zadeklarowana zmienna `wynik` posłuży do zapisania wyniku operacji arytmetycznej, a nastepnie jej wartość będzie umieszczona w wynikowym dokumencie HTML. Wiersz 8 zawiera zanurzony kod Java - każdy fragment kodu Java nazywany jest *scriptletem*. Przykładowy fragment kodu wykonuje prostą operację arytmetyczną. Scriptlety mogą przyjmować znacznie większe rozmiary – wówczas odpowiedni znacznik rozciąga się na wiele wierszy tekstu. W wierszu 10 znajduje się wyrażenie, którego wartość zostanie umieszczona w wygenerowanym dokumencie HTML dokładnie w tym miejscu, gdzie został umieszczony znacznik.

```
(1) <HTML>
(2) <HEAD>
(3) <TITLE>JSP demo</TITLE>
(4) </HEAD>
(5) <BODY>
(6) <%@ page language="java" %>
(7) <%! int wynik; %>
(8) <% wynik = 2*2; %>
(9) <H1> Wynik mnozenia:
(10)<%= wynik %>
(11)</H1>
(12)</BODY>
(13)</HTML>
```



Rys.3 Przykład dokumentu JSP

Podsumowując powyższy przykład, zauważamy, że dokument JSP jest w istocie dokumentem HTML, zawierającym dodatkowe znaczniki:

1. *Dyrektywy*, specyfikujące globalne cechy dokumentu (np. <%@ page language="java" %>)
2. *Deklaracje* zmiennych i metod globalnych dla dokumentu (np. <%! int wynik; %>)
3. *Scriptlety* w postaci zanurzonego kodu wykonywalnego Java (np. <% wynik = 2*2; %>)
4. *Wyrażenia*, umożliwiające włączanie dodatkowych treści do dokumentu wynikowego (np. <%= wynik %>); zwróćmy uwagę na to, że wyrażenia nie są kończone znakiem średnika

2.2 Przekazywanie parametrów

Ponieważ dokumenty JSP często pełnią funkcje tradycyjnych aplikacji, dlatego konieczne jest umożliwienie użytkownikowi parametryzowania ich działania. Technologia JSP oferuje identyczny mechanizm przekazywania parametrów wywołania jak w przypadku serwletów Java. Użytkownik specyfikuje wartości parametrów wywołania bądź wewnątrz adresu URL, bądź przy pomocy dodatkowego formularza HTML. Wewnątrz dokumentu JSP dostępny jest predefiniowany obiekt o nazwie `request`, który przechowuje wartości wszystkich otrzymanych parametrów. Odczyt wartości parametru odbywa się przy użyciu metody `getParameter()` obiektu `request`. Przykład parametryzowanego dokumentu JSP został przedstawiony na rys. 4. Wartości dwóch parametrów wywołania, nazwanych `par1` i `par2` są dodawane do siebie, a wynik wyświetlany jest na ekranie przeglądarki użytkownika.

```
(1) <HTML>
(2) <HEAD>
(3) <TITLE>JSP demo</TITLE>
(4) </HEAD>
(5) <BODY>
(6) <%@ page language=”java” %>
(7) <%! int wynik;
(8)     int skladnik1, skladnik2; %>
(9) <% skladnik1 = Integer.parseInt(
          request.getParameter(”par1”));
(10)   skladnik2 = Integer.parseInt(
          request.getParameter(”par2”));
(11)   wynik = skladnik1 * skladnik2; %>
(12)<H1> Wynik mnozenia:
(13)<%= wynik %>
(14)</H1>
(15)</BODY>
(16)</HTML>
```

Rys.4 Parametryzowany dokument JSP

W przykładzie z rys. 4, w wierszach 9-10 odczytywane są parametry wywołania dokumentu JSP – ich wartości zapisywane są w zmiennych `skladnik1` i `skladnik2`. Znaczenie pozostałych elementów jest identyczne jak w poprzednim przykładzie.

W celu uruchomienia przedstawionego dokumentu JSP, użytkownik musi przekazać wartości parametrów wykorzystując jedną z poniższych metod:

- umieścić definicje parametrów w adresie URL, np:
  `http://adres/sciezka/dokument.jsp?par1=2&par2=4`
- zbudować formularz HTML, skojarzony z dokumentem JSP; przykład takiego formularza przedstawiono na rys.5; nazwy pól formularza muszą być identyczne jak nazwy parametrów wywołania dokumentu JSP

```
(1) <HTML>
(2) <HEAD>
(3) <TITLE>Wprowadź parametry</TITLE>
(4) </HEAD>
(5) <BODY>
(6) <H1>Wprowadź parametry</H1>
(6) <FORM ACTION=’http://adres/sciezka/dokument.jsp’>
(7) Składnik1:
(8) <INPUT TYPE=’text’ NAME=’par1’><BR>
(9) Składnik2:
(10)<INPUT TYPE=’text’ NAME=’par2’><BR>
(11)<INPUT TYPE=’submit’ VALUE=’OK’>
(12)</FORM>
(15)</BODY>
(16)</HTML>
```

Rys.5 Przykład formularza HTML

2.3 Komponenty JavaBeans

Kod Java zanurzany wewnątrz dokumentu JSP może osiągnąć znaczne rozmiary i istotnie utrudnić dalsze modyfikacje i rozwój. W pewnym sensie przedstawiane dotychczas podejście do konstrukcji dokumentów przeczy podstawowej zalecie technologii JSP, jaką jest izolacja kodu wykonywalnego od formy prezentacji. Istnieje jednak możliwość wyraźniejszego rozdzielenia kodu od prezentacji dzięki zastosowaniu tzw. komponentów JavaBeans.

JavaBeans to standard tworzenia komponentów programowych Java, wykorzystywany w wielu środowiskach programistycznych (np. aplikacje JDeveloper). Podstawowa idea komponentów JavaBeans polega na hermetycznym zamknięciu kodu wykonywalnego w oddzielnej klasie, która może być nastepnie współdzielona przez wiele aplikacji. W przypadku technologii JSP oznacza to, że kod wykonywalny będzie mógł znajdować się w odrębnym pliku, związanym przy pomocy specjalnego znacznika. Dzięki temu wewnątrz właściwego dokumentu JSP nie będzie znajdować się zbyt wiele elementów Java.

Formalnie, komponent JavaBeans to pojedyncza klasa Java, zawierająca prywatne atrybuty (byc może wirtualne), specjalnie nazwane metody dostepu do atrybutów `getNazwaAtrybutu()` i `setNazwaAtrybutu()`, bezargumentowy konstruktor oraz dowolne inne metody. W celu zilustrowania zastosowań komponentów JavaBeans w technologii JSP rozważmy następujący przykład. Na rys. 6 przedstawiono kod źródłowy sparametryzowanego dokumentu JSP, wyświetlającego kwotę podatku dochodowego dla podanego dochodu, oraz kod źródłowy wykorzystanego komponentu JavaBeans. Zakładamy, że dokument ten wywoływany jest z parametrem o nazwie `Dochod`, ktorego wartością jest roczny dochód osoby fizycznej.

```
(1) <HTML>
(2) <HEAD>
(3) <TITLE>JSP demo</TITLE>
(4) </HEAD>
(5) <BODY>
(6) <%@ page language="java" %>
(7) <jsp:useBean id="podatki"
    class="taxCalc"/>
(8) <jsp:setProperty name="podatki"
     property="Dochod" />
(9) <H1> Kwota podatku dochodowego:
(10)<jsp:getProperty name="podatki"
     property="Podatek" />
(11)</H1>
(12)</BODY>
(13)</HTML>
```

```
public class taxCalc {
float kwota_dochodu;

public setDochod(float dochod) {
  kwota_dochodu = dochod;
 }

public float getPodatek() {
 float kwota_podatku;

 if (kwota_dochodu < 2295.79)
   kwota_podatku = 0;
 else if (kwota_dochodu < 32736)
   kwota_podatku =
   (float) (0.19 * (kwota_dochodu) – 436.20);
 else if (kwota_dochodu < 65472)
   kwota_podatku =
   (float) (0.3 * (kwota_dochodu - 32736) + 5783.64);
 else
   kwota_podatku =
   (float) (0.4 * (kwota_dochodu - 65472) + 15604.44);
 return kwota_podatku;
 }
}
```

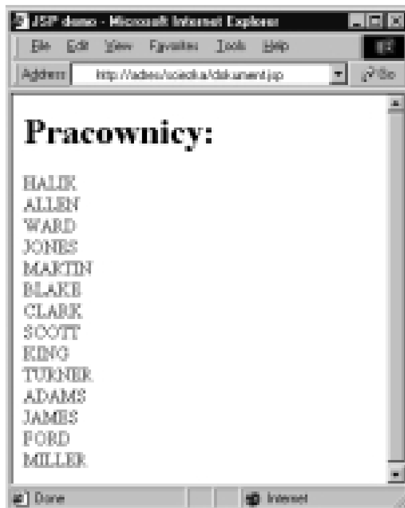
Rys. 6 Dokument JSP oraz wykorzystywany komponent JavaBeans

W wierszu 7 tworzony jest obiekt klasy `taxCalc` o nazwie podatki. Wiersz 8 wywołuje metodę `setDochod()`, której argumentem wejściowym niejawnie staje się wartość parametru wywołania dokumentu JSP o takiej samej nazwie (`Dochod`). Zwróćmy uwagę na to, że nazwa metody `setDochod()` jest wskazana niejawnie: przedrostek „`set`” jest automatycznie dołączany do nazwy atrybutu komponentu JavaBeans. W wierszu 10 wywoływana jest metoda `getPodatek()`, a jej wynik umieszczany jest w generowanym dokumencie HTML. Ponownie, przedrostek „`get`” jest dołączany automatycznie do nazwy atrybutu. Postać serwletu, wygenerowanego przez serwer Oracle8.1.7 na podstawie powyższego dokumentu JSP została przedstawiona w Dodatku A.

2.4 Wykorzystywanie JDBC

Komponenty JavaBeans mogą ułatwić konstrukcję bazodanowych dokumentów JSP, gdyż kod obsługi JDBC z reguły przybiera znaczne rozmiary. Przykład tego typu zastosowania przedstawiono na rys. 7. Komponent JavaBeans w postaci klasy `DbBean` odpowiada za: nawiązanie połączenia z bazą danych (metoda `getConnection()`), wydruk wyniku zapytania (metoda `getTable()`) i zakończenie sesji (metoda `closeConnection()`) .

```
(1) <HTML>
(2) <HEAD>
(3) <TITLE>JSP demo</TITLE>
(4) </HEAD>
(5) <BODY>
(6) <%@ page language="java" %>
(7) <jsp:useBean id="db" class="DbBean"/>
(8) <% db.openConnection(
     "jdbc:oracle:thin:@srv1:1521:orcl");
    %>
(9) <H1> Pracownicy:</H1>
(10)<%= db.displayTable() %>
(11)<% db.closeConnection(); %>
(12)</BODY>
(13)</HTML>
```




```
import java.sql.*;

public class DbBean {
   Connection conn;

   public void openConnection(String connect_string)
   throws SQLException {

     DriverManager.registerDriver(
 new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(
            connect_string,"scott","tiger");
    }

   public String displayTable()
   throws SQLException {
     String result = "";
     Statement stmt;
     ResultSet rset;

     stmt = conn.createStatement();
     rset = stmt.executeQuery(
            "select ename,sal from emp");

     while (rset.next())
        result += rset.getString("ename")+"<BR>";

     rset.close(); stmt.close();

     return result;
   }

   public void closeConnection()
   throws SQLException
   {
     conn.close();
   }
}
```

Rys. 7 Bazodanowy dokument JSP z wykorzystaniem JavaBeans

W praktyce, należałoby tak zmodyfikować przedstawiony przykład, aby połączenie z bazą danych nie było rozłączane po każdym odwołaniu do dokumentu (co powoduje pogorszenie wydajności). Fakt ten nie zmienia jednak ogólności naszych rozważań.

2.5 Obsługa JSP w środowisku Oracle

System zarządzania bazą danych Oracle, począwszy od wersji 8.1.7, rozpowszechniany jest wraz z serwerem WWW Apache, wyposażonym m.in. w moduły obsługi serwletów i JSP. Wcześniejsze wersje systemu Oracle wymagają uzupełnienia o serwer WWW obsługujący technologię JSP – może to być np. Apache. JSP jest również obsługiwane przez Oracle Application Server, począwszy od wersji 9i.

3 Podsumowanie

W artykule omówiono podstawowe własności technologii JSP. Z punktu widzenia programisty, najciekawszymi cechami JSP są: mozliwość odizolowania kodu wykonywalnego od formy prezentacji, korzystanie z bogatej funkcjonalności języka Java oraz dostęp do interfejsu JDBC. Głównym obszarem zastosowań JSP wydają się być aplikacje, w których dominująca rolę odgrywa często modyfikowana szata graficzna.

4 Bibliografia

[1] http://www.javasoft.com/
[2] http://www.servlet.com/
[3] http://www.webdevelopersjournal.com/
[4] http://jserv.java.sun.com/
[5] http://technet.oracle.com/
[6] Dokumentacja systemu zarządzania bazą danych Oracle 8.1.7
[7] "Database Access from JavaServer Pages", Julie Basu, Oracle Open World, November 1999
[8] "JavaServer Pages 1.0 Specification", Sun Microsystems

Dodatek A

Kod źródłowy serwletu automatycznie wygenerowanego dla dokumentu JSP z rys. 6. (komentarze dodane przez autorów).

```
import oracle.jsp.runtime.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import java.beans.*;
// deklaracja użycia klasy taxCalc
import taxCalc;

// nazwa klasy serwletu pochodzi od nazwy dokumentu JSP
// klasa oracle.jsp.runtime.HttpJsp to implementacja klasy HttpServlet
public class _testmz extends oracle.jsp.runtime.HttpJsp {
  public final String _globalsClassName = null;

// metoda _jspService w tym przykładzie jest logicznie rownoważna metodzie doGet()
// klasy HttpServlet
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
         throws IOException, ServletException {

    // kod dodany dla obsługi sesji - niewykorzystywany w tym przykładzie
    JspFactory factory = JspFactory.getDefaultFactory();
    PageContext pageContext = factory.getPageContext( this, request, response,
                          null, true, JspWriter.DEFAULT_BUFFER, true);
    HttpSession session = pageContext.getSession();
    if (pageContext.getAttribute(OracleJspRuntime.JSP_REQUEST_REDIRECTED,
   PageContext.REQUEST_SCOPE) != null) {
                pageContext.setAttribute(OracleJspRuntime.JSP_PAGE_DONTNOTIFY, "true",
                                   PageContext.PAGE_SCOPE);
         factory.releasePageContext(pageContext);
         return;
   }
    ServletContext application = pageContext.getServletContext();
    JspWriter out = pageContext.getOut();
    _testmz page = this;
    ServletConfig config = pageContext.getServletConfig();
    // ------------------------------------------------------------------
```

```
    try {
      // wyświetla "<HTML>\n <HEAD>\n<TITLE>JSP demo</TITLE>\n</HEAD>\n<BODY>\n"
      out.print(__jsp_StaticText.text[0]);

      // tworzy obiekt klasy taxCalc (JavaBeans) o nazwie podatki;
      // wynik translacji <jsp:useBean id="podatki" class="taxCalc"/>
      taxCalc podatki;
      if ((podatki = (taxCalc) pageContext.getAttribute( "podatki",
                           PageContext.PAGE_SCOPE)) == null) {
        podatki = (taxCalc) new taxCalc();
        pageContext.setAttribute( "podatki", podatki, PageContext.PAGE_SCOPE);
      }
      {
   // odczytuje parametry wywołania JSP/serwletu;
   // wynik translacji <jsp:setProperty name="podatki" property="Dochod" />
        String[] __paramList = request.getParameterValues( "Dochod");
        if ((__paramList != null) && (__paramList[0] != null) &&
            (__paramList[0].length() > 0)) {

          // jeżeli parametr Dochod jest obecny, wołana jest
          // metoda setDochod obiektu podatki
          podatki.setDochod(Float.valueOf( __paramList[0]).floatValue());
        }
      }

      // wyświetla "\n<H1> Kwota podatku dochodowego:\n"
      out.print(__jsp_StaticText.text[1]);

      // wykonuje metodę getPodatek obiektu podatki i wyświetla jej rezultat;
      // wynik translacji <jsp:getProperty name="podatki" property="Podatek" />
      out.print( podatki.getPodatek());

      // wyświetla "\n</H1></BODY></HTML>\n".
      out.print(__jsp_StaticText.text[2]);

      out.flush();
    }
    catch( Exception e) {
      try {
        if (out != null) out.clear();
      }
      catch( Exception clearException) {
      }
      pageContext.handlePageException( e);
    }
    finally {
      if (out != null) out.close();
      factory.releasePageContext(pageContext);
    }

  }

  // przygotowanie struktur, przechowujących statyczny tekst HTML z dokumentu JSP
  private static class __jsp_StaticText {
    private static final char text[][]=new char[3][];
    static {
      text[0] =
      "<HTML>\n <HEAD>\n<TITLE>JSP demo</TITLE>\n</HEAD>\n<BODY>\n".toCharArray();
      text[1] =
      "\n<H1> Kwota podatku dochodowego:\n".toCharArray();
      text[2] =
      "\n</H1></BODY></HTML>\n".toCharArray();
    }
  }
}
```