Logika prezentacji III

wykład prowadzi
Mikołaj Morzy

Aplikacje WWW

Plan wykładu

- Szablony JSP
  - cykl życia
  - deklaracje
  - dyrektywy
  - skryptlety
  - język EL
- Inne technologie szablonów
  - Velocity
  - WebMacro
  - FreeMarker

Logika prezentacji III (2)

Celem wykładu jest przedstawienie metod tworzenia logiki prezentacji aplikacji internetowej przy wykorzystaniu technologii szablonów rozwijanych dla języka Java. Podstawową technologią szablonów dla języka Java jest technologia JSP (ang. Java Server Pages), w ramach wykładu przedstawione zostaną cykl życia aplikacji JSP oraz podstawowe składowe technologii: deklaracje, dyrektywy i skryptlety. Dodatkowo zaprezentowany zostanie język JSP EL (ang. Expression Language), który ułatwia tworzenie złożonych aplikacji JSP. W drugiej części wykładu zostaną przedstawione komplementarne technologie szablonów dla języka Java: Velocity, WebMacro i FreeMarker.

Aplikacje WWW

JSP - wprowadzenie

- Technologia umożliwiająca łącznie statycznego kodu HTML lub XML z dynamicznym kodem Java
- Rozszerzenie technologii serwletów
- Podstawowe narzędzie tworzenia warstwy prezentacji w architekturze Java EE
- Historia
  - wersja JSP 1.2 i Servlet 2.3 częścią J2EE 1.3 (1999)
  - wersja JSP 2.0 i Servlet 2.4 częścią J2EE 1.4
  - wersja JSP 2.1 i Servlet 2.5 częścią Java EE 5 Platform

Logika prezentacji III (3)

Technologia Java Server Pages (JSP) to technologia szablonów umożliwiających łatwe łączenie statycznego kodu HTML lub XML z dynamiczną zawartością generowaną przez kod Java. Szablony są konstruowane przy pomocy kilku dodatkowych znaczników XML zwanych akcjami JSP (ang. JSP actions). Znaczniki te umożliwiają osadzanie kodu Java bezpośrednio w dokumencie HTML i XML, deklarowanie zmiennych, wartościowanie wyrażeń, czy współpracę z innymi komponentami, np. komponentami JavaBean. W rzeczywistości JSP jest rozszerzeniem podstawowej technologii serwletów. Dokumenty JSP są "w locie" tłumaczone na serwlety i kompilowane do postaci pseudokodu Java, a następnie wykonywane tak samo, jak tradycyjne serwlety przez właściwy kontener. JSP, wraz z towarzyszącymi rozszerzeniami (specyfikacją JavaBean i bibliotekami znaczników), stanowi podstawowe narzędzie tworzenia warstwy prezentacji w architekturze Java EE.

Historycznie, zarówno serwlety jak i JSP były równolegle opracowane przez Sun Microsystems. Począwszy od wersji JSP 1.2 (odpowiadała jej wersja Servlet 2.3 API) obie technologie były rozwijane pod auspicjami Java Community Process. Największą popularność technologia JSP zdobyła wraz z opublikowaniem wersji JSP 2.0 (wraz z Servlet 2.4 API obie technologie weszły do specyfikacji J2EE 1.4). W maju 2006 roku opublikowano długo oczekiwaną wersję stanowiącą następcę poprzedniej specyfikacji. J2EE została przemianowana na Java EE 5 Platform, zaś specyfikacje serwletów i JSP otrzymały odpowiednio numery 2.5 i 2.1.

Przykład prostej strony

```
<%@ page language="java" %>
<html>
  <head>
    <title>EuroKonwerter v.0.1</title>
  </head>
  <body>
  <jsp:include page="header.jsp"/>
  <%! double priceEUR = 10, pricePLN = 0; %>
  <%! Double ratioPLN2EUR = 4.75; %>

  <% pricePLN = priceEUR * ratioPLN2EUR; %>

  <p> Kwocie <%= priceEUR %> euro odpowiada
      <%= pricePLN %> zlotych
  </body>
</html>
```

dyrektywa

akcja

deklaracja

skryptlet

statyczne dane

wyrażenie

Na stronę JSP składają się

•statyczne dane: tekst jest kopiowany i wysyłany do klienta HTTP dokładnie w takiej postaci, w jakiej jest umieszczony na stronie JSP, najczęściej statycznymi danymi są bloki HTML lub XML

•dyrektywy: polecenia kontrolujące sposób generowania serwletu wynikowego na podstawie JSP

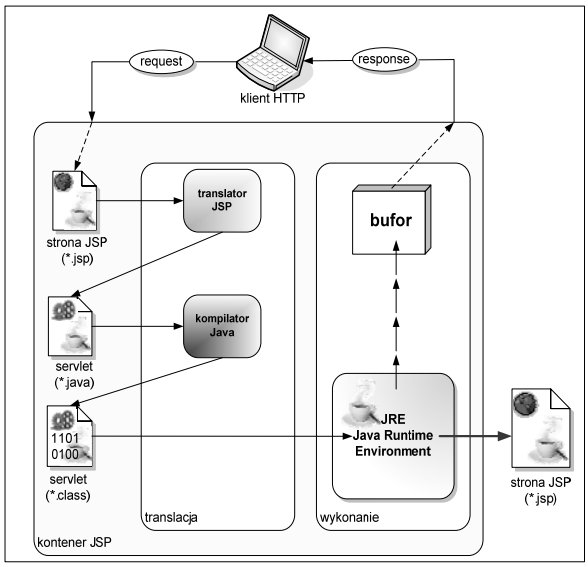
•deklaracje: znaczniki XML umożliwiające deklarowanie zmiennych globalnych

•skryptlety: znaczniki XML umożliwiające osadzenie kodu Java

•wyrażenia: znaczniki XML pozwalające wartościować wyrażenia Java i wyświetlać wartości w wynikowym kodzie HTML lub XML

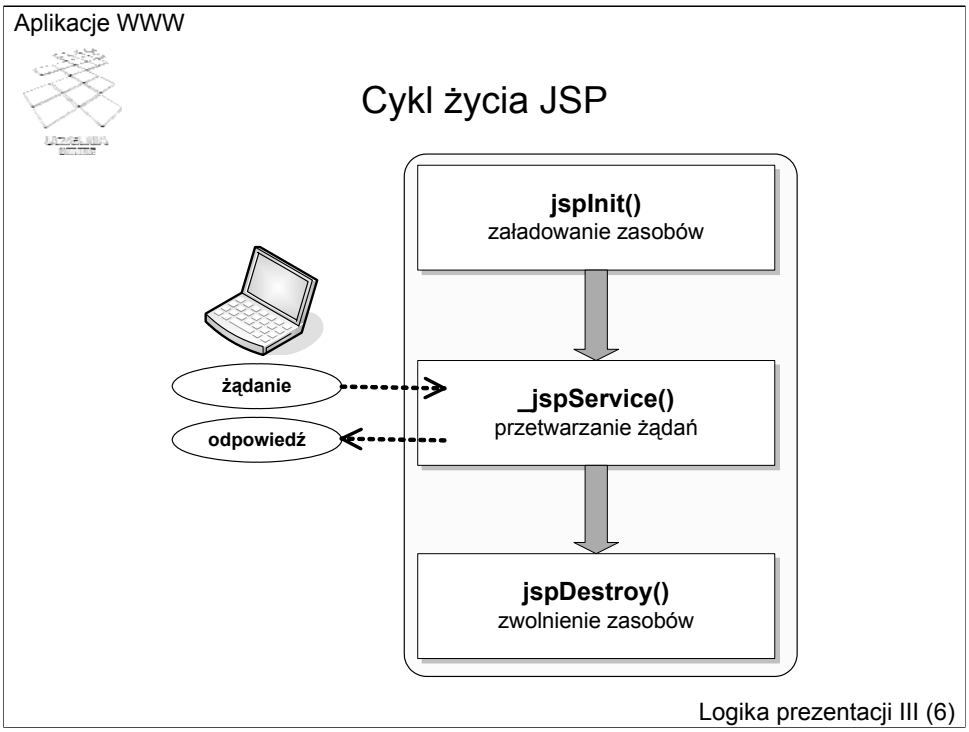
•akcje: znaczniki XML wywołujące funkcje serwera HTTP

Schemat działania JSP



Klient HTTP wysyła żądanie do serwera HTTP pobrania strony JSP. Żądanie zostaje przekierowane do właściwego kontenera JSP zlokalizowanego w serwerze aplikacji. Podczas pierwszego pobrania strona JSP zostaje wysłana do translatora JSP, który generuje wynikowy kod Java w postaci serwletu. Następnie serwlet w postaci źródłowej jest przesyłany do kompilatora Java, który przygotowuje pseudokod Java serwletu. Na tym kończy się faza translacji. Skompilowany serwlet jest przesyłany do maszyny wirtualnej Java, w której zostanie wykonany. Wynikiem działania serwletu jest strumień znaków składający się na wynikowy dokument HTML lub XML przesyłany do klienta HTTP. Strumień wyjściowy jest dodatkowo buforowany. W stronie JSP może się znaleźć adnotacja wskazująca, która inna strona JSP powinna zostać załadowana w przypadku wystąpienia błędu. Jeśli w trakcie wykonywania serwletu maszyna wirtualna Java napotka na jakiś błąd, sterowanie wraz z informacją o napotkanym błędzie zostanie przekazane do wskazanej strony obsługi błędu.

Powyższa procedura translacji ma miejsce tylko przy pierwszym odwołaniu do strony JSP. Skompilowane strony JSP pozostają załadowane do maszyny wirtualnej Java i kolejne odwołania do tej samej strony nie wymagają przejścia przez fazę translacji.



Cykl życia dokumentu JSP składa się z trzech faz:

•inicjalizacja: uruchomienie metody jspInit() w serwlecie wynikowym wygenerowanym na podstawie dokumentu JSP, inicjalizacja odbywa się w momencie załadowania serwletu wynikowego do kontenera, najczęściej podczas tej fazy następuje załadowanie potrzebnych zasobów, nawiązanie połączenia z bazą danych, itp.

•obsługa żądań: współbieżne wykonanie metody _jspService() z serwletu wynikowego, każde odwołanie do dokumentu JSP powoduje uruchomienie osobnego wątku w jednej instancji serwletu wynikowego

•zniszczenie: uruchomienie metody jspDestroy() z serwletu wynikowego, odbywa się w momencie usuwania serwletu wynikowego z kontenera, faza służy do zamknięcia i zwolnienia wszystkich zasobów alokowanych do serwletu wynikowego.

Poprawne wykorzystanie faz inicjalizacji, obsługi żądań i zniszczenia umożliwia efektywne zarządzanie zasobami, jakimi dysponuje aplikacja.

Dyrektywy

- Kontrolują sposób translacji JSP do serwletu
- **<%@ include %>**: włączenie zewnętrznego pliku
  - `file`
- **<%@ page %>**: ustawienia strony
  - `import`, `contentType`, `errorPage`, `isErrorPage`, `info`, `buffer`, `session`, `autoFlush`, `extends`, `isThreadSafe`, `isELEnabled`
- **<%@ taglib %>**: wskazanie na bibliotekę znaczników
  - `prefix`, `uri`

Dyrektywy kontrolują sposób translacji strony JSP do wynikowego serwletu. Dyrektywy umieszczone są w znacznikach <%@ %>. Dostępne są trzy dyrektywy:

1. <%@ include %>: umożliwia dołączenie zewnętrznego pliku, najczęściej zawierającego współdzielony kod, zaleca się, aby dołączane pliki miały rozszerzenie *.jspf (JSP Fragment), adres pliku podawany jest w atrybucie *file*.

2. <%@ page %>: kontroluje różne ustawienia strony za pomocą następujących atrybutów:

•import: oddzielone przecinkami nazwy klas do zaimportowania

•contentType: wartość nagłówka HTTP specyfikującego typ MIME odpowiedzi

•errorPage: nazwa strony JSP, która powinna zostać załadowana w przypadku wystąpienia błędu

•isErrorPage: flaga oznaczająca, że dana strona JSP służy do obsługi błędów

•info: łańcuch znaków dodatkowo opisujący stronę

•buffer: specyfikacja rozmiaru bufora (można podać wartość none)

•session: flaga informująca czy klient HTTP musi się znajdować w sesji w celu oglądania strony JSP

•autoFlush: flaga określająca czy bufor zostanie automatycznie wyczyszczony po wypełnieniu

•extends: nazwa klasy, z której dziedziczy serwlet wynikowy

•isThreadSafe: flaga określająca czy strona JSP może być bezpiecznie wykonana w środowisku wielodostępnym

•isELEnabled: flaga określająca czy strona JSP może korzystać z języka wyrażeń EL

3. <%@ taglib %>: wskazanie na użyte biblioteki znaczników, wyspecyfikowane za pomocą atrybutów prefix i uri

Deklaracje

- Pozwalają na deklarowanie metod i składowych serwletu wynikowego
- Mogą zawierać inicjalizację
- Wprowadzane przez znaczniki **<%! %>**

```
<%! int licznik = 0;  %>
<%! int a, b, c; %>
<%! Array mojaTablica = new Array(); %>
<%! int ktoraGodzina() {
  Calendar cal = new GregorianCalendar();
  return cal.get(Calendar.HOUR_OF_DAY);
  } %>
```


Deklaracje są umieszczane w stronie JSP za pomocą znaczników <%! %>. Deklaracje służą do wprowadzania składowych oraz metod, które zostaną dodane do wynikowego serwletu. Należy pamiętać, że każda deklaracja musi się kończyć średnikiem. Zmienne zadeklarowane wewnątrz znaczników <%! %> są tłumaczone na składowe serwletu, czyli są współdzielone przez wszystkich klientów HTTP korzystających z danej strony JSP. Dzieje się tak, ponieważ strony JSP są zazwyczaj wykonywane jako współbieżne wątki korzystające z tej samej instancji serwletu wynikowego. Taki model współbieżnego wykonania może skutkować nieprzewidzianymi błędami. Dostęp do zmiennych może być synchronizowany, lecz synchronizacja powoduje zmniejszenie stopnia współbieżności, a co za tym idzie, negatywnie wpływa na efektywność.

Skryptlety

- Znaczniki XML umożliwiające osadzanie kodu Java
- Mogą generować kod HTML lub XML za pomocą predefiniowanego obiektu **`out`**
- Wprowadzane przez znaczniki **<% %>**

```
<% Calendar dzis = new GregorianCalendar();
   int godzina = dzis.get(Calendar.HOUR_OF_DAY);
   out.println("<B>jest godzina " + godzina + "</B>");
%>
```


Skryptlety umożliwiają osadzenie dowolnego kodu języka Java bezpośrednio w dokumencie HTML lub XML. Skryptlety umieszcza się w znacznikach <% %>. Należy pamiętać, że każde polecenie wewnątrz skryptletu musi być zakończone średnikiem. Cały kod umieszczony w skryptlecie jest przenoszony bez żadnych modyfikacji do wnętrza metody _jspService() serwletu wynikowego. Zmienne zadeklarowane w skryptlecie są lokalne i nie są współdzielone przez współbieżne wątki tej samej instancji serwletu. Skryptlet może wygenerować kod, który znajdzie się w dokumencie wynikowym. Do wygenerowania kodu HTML lub XML do dokumentu wynikowego należy posłużyć się predefiniowanym obiektem out z klasy javax.servlet.jsp.JspWriter (obiekt ten jest domyślnie dostępny).

Skryptlety i HTML

- Instrukcje warunkowe: uwaga na nawiasy!

```
<html>
<body>
  <% Calendar today = new GregorianCalendar();
     int now = today.get(Calendar.HOUR_OF_DAY);

       if (now <= 15) { %>
         <b> jest jeszcze przed piętnastą </b>
  <% } else { %>
         <b> jest już po piętnastej </b>
  <% } %>
</body>
</html>
```


W przypadku dłuższego kodu HTML posługiwanie się obiektem out może być kłopotliwe, można wówczas wykorzystywać jednocześnie skryptlet i statyczny kod HTML. Należy jednak pamiętać o poprawnym zagnieżdżaniu i zamykaniu nawiasów. Ma to szczególne znaczenie w przypadku instrukcji warunkowych. W przypadku przedstawionym na slajdzie tylko jeden blok HTML zostanie wyświetlony. W zależności od wartości warunku logicznego wykona się część objęta kodem if () { ... } lub else { ... }. Wewnątrz obu bloków umieszczono kod HTML. Każdorazowe przejście między skryptletem i kodem HTML wymaga zamknięcia i otwarcia znaczników <% i %>.

Wyrażenia

- Znaczniki XML umożliwiające wartościowanie wyrażenia
- Wartość konwertowana do łańcucha znaków i włączana do wynikowego kodu HTML lub XML
- Wprowadzane przez znaczniki **<%= %>**

```
<%! Calendar today = new GregorianCalendar(); %>
<P>Dzisiejsza data to
  <%= today.get(Calendar.DAY_OF_MONTH) %>,
  <%= today.get(Calendar.MONTH) %>, roku
  <%= today.get(Calendar.YEAR) %>
</P>
```


Wyrażenia to znaczniki XML umożliwiające wartościowanie wyrażenia i włączenie wyniku wartościowania bezpośrednio do wynikowego kodu HTML lub XML. Wyrażenia są ujęte w znaczniki <%= %>. W przeciwieństwie do deklaracji i skryptletów wyrażenia nie mają kończącego średnika. Wynika to z techniki ich translacji: wyrażenia są translowane bezpośrednio na wywołania metody out.println() umieszczonej w metodzie _jspService() wynikowego serwletu

Znaczniki JSP (akcje)

- Znaczniki XML wywołujące akcje serwera aplikacji
- Dostępne akcje
  - **<jsp:include>**: włączenie zewnętrznego kodu
  - **<jsp:forward>**: przekazanie sterowania
  - **<jsp:param>**: zdefiniowanie parametru
  - **<jsp:plugin>**: obsługa apletów Java
  - **<jsp:fallback>**: gdy klient nie obsługuje apletów
  - i znaczniki do obsługi komponentów JavaBean

Znaczniki JSP, zwane też akcjami JSP, to znaczniki XML powodujące uruchomienie akcji wykonywanej przez serwer aplikacji. Dostępne są następujące znaczniki JSP:

•<jsp:include>: powoduje włączenie do bieżącej strony JSP zawartości innej strony JSP. Technicznie sterowanie jest tymczasowo przekazane do strony wskazanej przez atrybut page, a po zakończeniu przetwarzania wskazanej strony sterowanie powraca do oryginalnej strony. Dzięki temu mechanizmowi zawartość danej strony JSP może być współdzielona przez wiele różnych stron.

•<jsp:forward>: akcja powoduje natychmiastowe przekazanie sterowania do strony wskazanej w atrybucie page znacznika. Przekierowanie jest transparentne dla klienta HTTP.

•<jsp:param>: znacznik umożliwia zdefiniowanie parametrów podczas przekazywania sterowania za pomocą znaczników <jsp:include> lub <jsp:forward>. Każdy parametr ma nazwę określoną w atrybucie name i wartość określoną w atrybucie value.

•<jsp:plugin>: znacznik powoduje wygenerowanie poprawnego osadzenia apletu Java w kodzie HTML. Osadzenie odbywa się poprzez właściwą kombinację znaczników <object> i <embed>, dzięki czemu aplet uruchamia się poprawnie we wszystkich przeglądarkach.

•<jsp:fallback>: służy do wprowadzenia fragmentu kodu wyświetlanego w przypadku, gdy klient HTTP nie obsługuje apletów

Dodatkowo występuje zbiór znaczników do obsługi komponentów JavaBean, zostaną one omówione w trakcie innego wykładu.

Obiekty predefiniowane

- Bogaty zestaw predefiniowanych obiektów
  - **`config`**: dostęp do parametrów inicjalizacyjnych
  - **`request`**: żądanie otrzymane od klienta HTTP
  - response: odpowiedź wysłana klientowi HTTP
  - **`session`**: sesja HTTP
  - **`application`**: kontekst aplikacji
  - **`pageContext`**: kontekst strony wraz ze zmiennymi
  - **`out`**: strumień stanowiący odpowiedź dla klienta HTTP
  - **`page`**: serwlet wynikowy dla dokumentu JSP

JSP dostarcza wielu predefiniowanych obiektów, które ułatwiają pisanie skryptletów. Obiekty predefiniowane są tworzone i inicjalizowane automatycznie.

•config: obiekt jest instancją klasy javax.servlet.ServletConfig i umożliwia przede wszystkim odczytywanie parametrów inicjalizacyjnych zapisanych w pliku deskryptora instalacji web.xml za pomocą metody config.getInitParameter()

•request: obiekt jest instancją klasy javax.servlet.http.HttpServletRequest i zawiera całe żądanie HTTP otrzymane od klienta, obiekt umożliwia m.in. odczytywanie parametrów przesyłanych do dokumentu JSP lub odczytywanie nagłówków żądania

•response: obiekt jest instancją klasy javax.servlet.http.HttpServletResponse i reprezentuje całą odpowiedź przesyłaną do klienta HTTP w odpowiedzi na otrzymane żądanie, obiekt umożliwia m.in. dodawanie i modyfikację nagłówków HTTP, dodawanie ciasteczek, itp.

•session: obiekt jest instancją klasy javax.servlet.http.HttpSession i reprezentuje sesję HTTP tworzoną automatycznie przy każdym odwołaniu do dokumentu JSP, obiekt umożliwia odczytywanie i zapisywanie zmiennych widocznych w kontekście bieżącej sesji

•application: obiekt jest instancją klasy javax.servlet.ServletContext i reprezentuje cały kontekst aplikacji internetowej postrzegany przez serwlet wynikowy, odwołania do obiektu application odpowiadają wołaniu ServletConfig.getServletContext() w serwlecie wynikowym

•pageContext: obiekt jest instancją klasy javax.servlet.jsp.PageContext i reprezentuje cały kontekst pojedynczej strony JSP, w tym wszystkie obiekty predefiniowane dostępne na stronie, metody do przekierowywania i włączania innych zasobów aplikacyjnych, oraz metody do odczytywania i zapisywania zmiennych widocznych w zasięgu bieżącej strony

•out: obiekt jest instancją klasy javax.servlet.jsp.JspWriter i reprezentuje strumień wyjściowy będący odpowiedzią wysyłaną do klienta HTTP, obiekt out łączy funkcjonalność klas java.io.PrintWriter i java.io.BufferedWriter i umożliwia generowanie odpowiedzi zarówno z wykorzystaniem buforowania jak i w sposób bezpośredni

•page: obiekt reprezentuje serwlet wynikowy dla bieżącej strony JSP (odpowiednik referencji this w języku Java)

JSP w XML – dokumenty JSP

- Strony JSP nie są poprawnymi dokumentami XML
- Standard JSP 1.2 wprowadza tzw. **dokumenty JSP**
  - znacznik **`<jsp:root xmlns:jsp=adres>`**
  - skryptlety, deklaracje i wyrażenia posiadają odpowiadające im znaczniki XML
    - **`<jsp:directive>`**
    - **`<jsp:declaration>`**
    - **`<jsp:expression>`**
    - **`<jsp:scriptlet>`**
    - **`<jsp:text>`**

Tradycyjne strony JSP nie są poprawnymi dokumentami XML, ponieważ nie posiadają poprawnie zagnieżdżonych i zamkniętych znaczników. Wynika to w dużej mierze z tego, że większość dokumentów HTML nie spełnia wymagań XML. W wersji JSP 1.2 wprowadzono możliwość definiowania dokumentów JSP, które są poprawnymi dokumentami XML.

Każdy dokument JSP zaczyna się od znacznika <jsp:root> identyfikującego plik jako dokument JSP i podającego właściwą przestrzeń nazw. Każdemu elementowi JSP odpowiada właściwy znacznik XML. Standard XML nie przewiduje swobodnego tekstu, stąd wszystkie znaczniki HTML umieszczone na stronie JSP muszą być ograniczone znacznikami <jsp:text>. Dokumenty JSP są dłuższe niż odpowiadające im strony JSP. Początkowo możliwość przygotowywania dokumentów JSP zgodnych ze standardem XML nie budziła istotnego zainteresowania osób projektujących i programujących strony JSP. W ostatnim czasie obserwuje się jednak rosnące zainteresowanie dokumentami JSP. Ma to związek z coraz powszechniejszym wykorzystaniu języka XML w tworzeniu aplikacji internetowych.

Przykład dokumentu JSP

```
<?xml version="1.0" encoding="UTF-8"?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">

    <jsp:directive.page contentType="text/xml;charset=UTF-8"/>
    <jsp:directive.page import="java.util.Date"/>

    <jsp:declaration>
        Date today = new Date();
    </jsp:declaration>

    <jsp:text><![CDATA[ <html><body> <h1>Hello World!</h1>
        Today is : ]]>
    </jsp:text>

    <jsp:expression>today</jsp:expression>

    <jsp:text><![CDATA[ </body></html> ]]></jsp:text>

</jsp:root>
```


Slajd przedstawia przykładowy dokument JSP. W pierwszym wierszu mieści się tradycyjny nagłówek XML. Drugi wiersz zawiera znacznik <jsp:root> będący jednocześnie korzeniem dokumentu XML i wskazaniem na właściwą przestrzeń nazw. Kolejno umieszczono dwa znaczniki <jsp:directive.page> odpowiadające znacznikowi dyrektywy <%@ page …>. Deklaracja zmiennej umieszczona została w znaczniku <jsp:declaration>. Znacznik <jsp:text> zawiera fragment kodu HTML, ponieważ kod ten zawiera znaczniki, które nie powinny być interpretowane przez silnik XML, fragment ten jest dodatkowo otoczony deklaracjami XML <![CDATA[…]]>. Wreszcie wyrażenie (określenie wartości zmiennej) jest ujęte w znaczniki <jsp:expression>.

Obsługa wyjątków

- Strona obsługująca wyjątki
  - wskazywana przez dyrektywę
    <%@ page errorPage="errorHandler.jsp" %>
  - musi zawierać dyrektywę
    <%@ page isErrorPage="true" %>
  - najczęściej importuje zawartość pakietu java.io
    <%@ page import="java.io.*" %>
- Dostęp do wyjątku przez obiekt **`exception`**
- Specjalny wyjątek **`JspException`**

Poza tradycyjną obsługą wyjątków wewnątrz skryptletów za pomocą bloków try {} catch () technologia JSP umożliwia osobom programującym definiowanie specjalnych stron obsługi wyjątków. Strona JSP będąca stroną obsługi wyjątku musi posiadać w dyrektywie <%@ page %> atrybut isErrorPage ustawiony na wartość true. Dana strona JSP wskazuje na swoją stronę obsługi wyjątku przez atrybut errorPage dyrektywy <%@ page %>. Najczęściej programista chce mieć dostęp do szczegółów zgłoszonego wyjątku, taką możliwość oferuje predefiniowany obiekt exception będący instancją klasy java.lang.Throwable i oferujący metody do pobierania szczegółowych informacji o błędach (np. exception.printStackTrace() lub exception.toString()). Wykorzystanie obiektu exception najczęściej wiąże się z koniecznością zaimportowania zawartości pakietu java.io.* (np. klas StringWriter i PrintWriter), które są niezbędne m.in. do zapisania wyjątku w pliku dziennika. JSP oferuje również mechanizm zgłaszania własnych wyjątków z komunikatami przyjaznymi dla użytkownika. Zajmuje się tym specjalna klasa JspException będąca podklasą klasy java.lang.Exception.

Wywołanie strony JSP z serwletu

- Wywołanie strony JSP z serwletu
  - pobranie kontekstu serwletu
  - pobranie zarządcy żądań
  - przekierowanie przez **`include()`** lub **`forward()`**
- Przekazanie danych z serwletu do strony JSP
  - umieszczenie parametrów w adresie URL
  - wykorzystanie obiektu **`request`**

Aby wywołać stronę z serwletu należy wykonać następujące kroki:

1. Pobrać kontekst serwletu: ServletContext ctx = this.getServletContext();
2. Pobrać zarządcę żądań: RequestDispatcher dispatcher = ctx.getRequestDispatcher("/somepage.jsp");
3. Przekierować żądanie HTTP: dispatcher.forward(request,response);

Przekierowanie może mieć charakter trwały (metoda forward(), sterowanie pozostaje w stronie JSP) lub tymczasowy (metoda include(), po wykonaniu strony JSP sterowanie wraca do wołającego serwletu). W obu przypadkach należy do strony JSP przekazać obiekty request i response reprezentujące, odpowiednio, oryginalne żądanie HTTP otrzymane od klienta i odpowiedź przesyłaną klientowi. Dodatkowe dane mogą być przekazane do strony JSP poprzez umieszczenie parametrów bezpośrednio w adresie URL, do którego następuje przekierowanie (np. RequestDispatcher dispatcher = ctx.getRequestDispatcher("/somepage.jsp?country=Poland&city=Poznan");) albo poprzez ustawienie wartości atrybutów obiektu request za pomocą metod setAttribute() i getAttribute().

Wywołanie strony JSP z serwletu - przykład

simpleServlet.java

```
ServletContext ctx = this.getServletContext();
RequestDispatcher dispatcher =
  ctx.getRequestDispatcher("/simplePage.jsp");

request.setAttribute("name","James Bond");
dispatcher.forward(request,response);
```

simplePage.jsp

```
<%@page contentType="text/html"%>
<html>
<body>
    <h1>JSP Page</h1>
    Hello <%= request.getAttribute("name") %>!
</body>
</html>
```


Slajd przedstawia fragment kodu serwletu simpleServlet.java wołającego stronę simplePage.jsp. W pierwszym kroku następuje pobranie kontekstu serwletu, następnie na podstawie kontekstu zostaje pobrany zarządca żądań związany z podanym adresem URL. W kolejnym kroku obiekt request zostaje zmodyfikowany poprzez dopisanie dodatkowego (a zatem nie pochodzącego od klienta HTTP) atrybutu o nazwie "name" i wartości "James Bond". W ostatnim kroku sterowanie zostaje przekazane do strony simplePage.jsp. Ponieważ metodą przekazania sterowania jest forward(), a zatem po zakończeniu wykonywania wskazanej strony JSP sterowanie nie powróci do wołającego serwletu. Warto napomknąć, że całe przekierowanie jest całkowicie transparentne dla klienta HTTP, który nie jest świadomy, że jego żądanie nie jest teraz obsługiwane przez oryginalny zasób (serwlet), ale przez stronę JSP. W praktyce oznacza to, że jeśli klientem HTTP jest przeglądarka internetowa, to po stronie klienta będzie widać wciąż oryginalny adres URL, natomiast w przeglądarce zostanie wyświetlony kod HTML wygenerowany przez stronę JSP. Drugi listing pokazuje kod strony JSP. Wartość atrybutu dodanego przez serwlet zostaje odczytana za pomocą metody getAttribute().

Język wyrażeń EL - wprowadzenie

- Niewygodna składnia wyrażeń JSP
  `<tag attribute="<%= request.getAttribute("name") %>">`
- Język wyrażeń (JSP Expression Language, JSP EL) wprowadzony w JSP 2.0
  - uproszczenie kodu: **`<tag attribute="${name}">`**
  - opcjonalność: **`<%@ page isELIgnored="true" %>`**
  - dodatkowe obiekty predefiniowane
  - zmienne: tablice, mapy, listy, własności
  - ewaluacja wyrażeń, warunków logicznych

Technologia JSP była wielokrotnie krytykowana ze względu na nieprzejrzystość kodu i kłopoty związane z łączeniem wyrażeń JSP z tradycyjnymi znacznikami. Przykładowo, utworzenie znacznika z atrybutem odczytanym przez wyrażenie JSP z obiektu request może mieć postać: <tag attribute="<%= request.getAttribute("name") %>">. W wersji 2.0 standardu JSP wprowadzono język wyrażeń (ang. Expression Languagae, JSP EL), który stanowi bardzo istotne usprawnienie technologii JSP. JSP EL umożliwia bardzo daleko idące uproszczenie kodu poprzez spójną i prostą notację. Powyższy znacznik przy wykorzystaniu JSP EL przyjmuje postać <tag attribute="${name}">. Ponieważ składnia ${} nie była zarezerwowana dla języka wyrażeń w wersjach JSP 1.2 i wcześniejszych, JSP EL nie jest kompatybilny w dół. Stąd, aplikacje J2EE w wersji 1.3 domyślnie mają wyłączoną obsługę języka wyrażeń, natomiast aplikacje przygotowane na platformie J2EE 1.4 mają obsługę JSP EL włączoną. Interpretacja JSP EL może być włączana i wyłączana dla indywidualnych stron za pomocą dyrektywy <%@ page %> przez atrybut isELIgnored (przyjmuje wartości true i false). JSP EL oferuje wiele dodatkowych obiektów predefiniowanych, które ułatwiają tworzenie stron JSP (m.in. obsługa parametrów żądania, ciasteczek, czy nagłówków). JSP EL umożliwia definiowanie i wykorzystywanie zmiennych będących nie tylko prostymi typami, ale także tablicami, mapami, listami, czy własnościami innych obiektów. Co istotne, obsługa prostych i złożonych zmiennych odbywa się dokładnie w ten sam sposób za pomocą uproszczonej notacji. Wreszcie JSP EL oferuje możliwość ewaluacji wyrażeń arytmetycznych i warunków logicznych.

JSP EL - wyrażenia

- Wyrażenia JSP EL mogą się znaleźć:
  - w statycznym tekście
  - w atrybucie znacznika standardowego lub znacznika zdefiniowanego przez użytkownika
- Wyrażenia w notacji **`${zmienna.pole}`**
- Automatyczna konwersja typu wynikowego wyrażenia do oczekiwanego typu

Wyrażenia JSP EL mogą się znaleźć bezpośrednio w statycznym tekście lub jako wartości atrybutów znaczników, zarówno standardowych, jak i znaczników pochodzących z bibliotek znaczników i znaczników definiowanych przez użytkownika. Najczęściej wartość wyrażenia JSP EL jest zwracana jako łańcuch znaków. Wyrażenia JSP EL przyjmują postać ${zmienna.pole} (poza wyrażeniami arytmetycznymi i logicznymi). Ewaluacja wyrażenia ${produkt.cena} przebiega na podstawie zachowania metody PageContext.findAttribute() i polega na przeszukaniu zasięgów page, request, session i application w poszukiwaniu obiektu o nazwie "produkt" i cesze "cena". Wyrażenie ${produkt.cena} jest traktowane przez JSP EL w taki sam sposób jak wyrażenie ${produkt["cena"]} i obie notacje są zunifikowane. W zależności od faktycznego typu obiektu "produkt" wynik ewaluacji może zwrócić:

•jeśli "produkt" jest typu java.util.Map, to ${produkt.cena} zwraca produkt.get("cena") lub null jeśli !produkt.containsKey("cena")

•jeśli "produkt" jest typu java.util.List lub Array, to ${produkt.cena} zwraca produkt.get((int)cena) lub null (np. jeśli metoda get zwróci wyjątek IndexOutOfBoundsException)

•jeśli "produkt" jest komponentem JavaBean, to ${produkt.cena} zwraca produkt.getCena() lub null (jeśli komponent zwróci wyjątek)

Wyrażenia JSP EL są ewaluowane od lewej strony do prawej strony, typ wynikowy jest automatycznie konwertowany do typu oczekiwanego.

JSP EL – obiekty predefiniowane

- Dodatkowe obiekty predefiniowane:
  - **`pageContext:`** pełen kontekst strony JSP
  - **`param`**: wartość parametru
  - **`paramValues`**: tablica wartości parametru
  - **`header`**: wartość nagłówka HTTP
  - **`headerValues`**: tablica wartości nagłówka HTTP
  - **`cookie`**: wartość ciasteczka
  - **`initParam`**: wartość parametru inicjalizacyjnego
  - **`pageScope`**, **`requestScope`**, **`sessionScope`**, **`applicationScope`**

Język wyrażeń JSP EL oferuje, poza standardowymi obiektami predefiniowanymi JSP, następujące obiekty predefiniowane:

•pageContext: kontekst strony JSP zawierający m.in. obiekty takie jak ServletContext, session, request, response

•param: wartość parametru o podanej nazwie, np. ${param.city} zwraca wartość parametru o nazwie "city"

•paramValues: tablica wartości parametru o podanej nazwie, analogicznie do obiektu param

•header: wartość nagłówka HTTP o podanej nazwie, np. ${header.user-agent} zwraca opis klienta HTTP

•headerValues: tablica wartości nagłówka HTTP o podanej nazwie, analogicznie do obiektu header

•cookie: wartość ciasteczka o podanej nazwie

•initParam: wartość parametru inicjalizacyjnego zapisanego w pliku deskryptora aplikacji webowej web.xml

•obiekty reprezentujące różne zasięgi widzialności zmiennych i komponentów JavaBean: zasięg strony, żądania, sesji i aplikacji

JSP EL – literały i operatory

- Literały
  - logiczne: **`true`** i **`false`**
  - liczby całkowite i zmiennoprzecinkowe
  - łańcuchy znaków
- Operatory
  - arytmetyczne: **`+ - * / % mod div`**
  - logiczne: **`and or not && || !`**
  - porównania: **`== != => > <= < lt gt le ge eq ne`**
  - **`empty`**, warunkowy **`A ? B : C`**

Język wyrażeń JSP EL umożliwia wykorzystanie literałów logicznych, literałów reprezentujących liczby całkowite i zmiennoprzecinkowe (identycznie jak w języku Java), łańcuchów znaków (podobnie jak w języku Java, literały mogą być otoczone cudzysłowami lub apostrofami). Język wyrażeń JSP EL wykorzystuje typowe operatory arytmetyczne, logiczne (jako słowa kluczowe i symbole) oraz operatory porównania (jako słowa kluczowe i symbole). Dodatkowo, JSP EL definiuje operator empty służący do sprawdzenia, czy wartością zmiennej nie jest wartość pusta null. Możliwe jest także wykorzystanie operatora warunkowego A ? B : C, którego wynik (B lub C) zależy od logicznej ewaluacji A.

JSP EL - przykład

simpleServlet.java

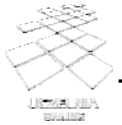
```
ServletContext ctx = this.getServletContext();
RequestDispatcher dispatcher =
  ctx.getRequestDispatcher("/simplePage.jsp?code=007");

request.setAttribute("name","James Bond");
dispatcher.forward(request,response);
```

simplePage.jsp

```
<%@page contentType="text/html"%>
<html>
<body>
    Hello ${name}!
    Your browser is ${header["user-agent"]}.
    Your secret code is ${param.code}.
</body>
</html>
```


Slajd przedstawia analogiczny przykład wołania strony JSP z serwletu. W przykładzie strona JSP wykorzystuje język wyrażeń JSP EL do odczytania wartości atrybutu przekazanego w obiekcie request. Atrybut ten jest widoczny w zasięgu widoczności żądania, ale ewaluacja wyrażenia ${name} automatycznie przeszukuje wszystkie zasięgi, od najwęższego (pageScope) do najszerszego (applicationScope). Przykład demonstruje także sposób wykorzystania JSP EL do odczytania wartości parametru HTTP oraz odczytania wartości parametru przesłanego w adresie URL.



Technologie szablonów dla serwletów Java

- Służą do generowania dokumentów tekstowych na podstawie wcześniej przygotowanych szablonów
- Najczęściej stosowane w serwletach jako alternatywa dla JSP
  - powstały, gdy JSP było technologią niedojrzałą
- Zalety:
  - pełna separacja kodu HTML i Java
  - czytelny podział zadań między twórców aplikacji
  - duża czytelność kodu

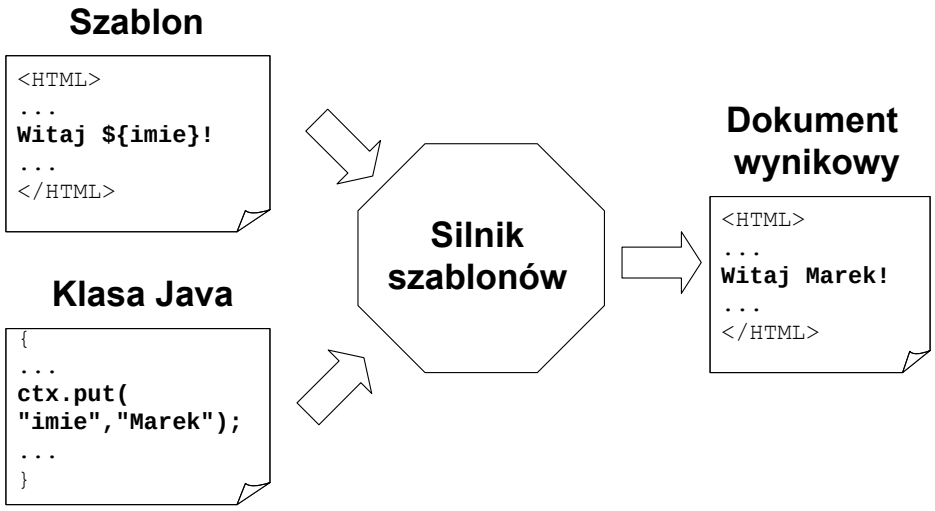
Technologie szablonów służą do generowania wszelkiego rodzaju dokumentów tekstowych z wcześniej przygotowanych szablonów. Najczęściej stosuje się je w serwletach do generowania dokumentów HTML, ale można je stosować również do generowania dokumentów XML, RTF, WML, wiadomości email, sms, kodów źródłowych programów lub plików konfiguracyjnych.

Technologie szablonów w połączeniu z serwletami Java stanowią ważną alternatywę dla JSP w zakresie implementacji warstwy prezentacji na platformie Java EE. Technologie szablonów pojawiły się w odpowiedzi na wady pierwszej odsłony technologii JavaServer Pages (JSP).

Podstawową zaletą stosowania szablonów jest możliwość pełnego odseparowania kodu Java zawierającego logikę biznesową i logikę prezentacji od kodu HTML definiującego ogólny wygląd strony WWW. Umożliwia to czytelny podział zadań między twórców aplikacji odpowiedzialnych za logikę aplikacji i jej stronę graficzną, a ponadto prowadzi do czytelnego kodu aplikacji.

Należy zwrócić uwagę, że mimo iż separacja kodu Java i znaczników opisujących wygląd strony była również celem JSP, to w początkach swego istnienia technologia JSP tej separacji w pełni nie umożliwiała. Dopiero wprowadzenie do technologii możliwości definiowania własnych znaczników i pojawienie się standardowej biblioteki znaczników JSTL, umożliwiło implementację złożonej logiki prezentacji w JSP bez konieczności zagnieżdżania w stronach fragmentów kodu Java. Obecnie JSP jest już wolne od większości wad, które spowodowały powstanie technologii szablonów. JSP ma status oficjalnej i podstawowej technologii do implementacji warstwy prezentacji na platformie Java EE, a technologie szablonów są używane przez twórców aplikacji, którym składnia JSP po prostu nie odpowiada.

Zasada działania technologii szablonów



Technologie szablonów są dostępne w postaci tzw. silników szablonów (ang. template engine). Silniki szablonów są implementowane w postaci bibliotek klas Java i mogą być wykorzystane zarówno z poziomu serwletów jak i innych rodzajów aplikacji Java. Aplikacja składa się z klas Java realizujących logikę biznesową i przygotowujących dane do prezentacji na stronach WWW oraz szablonów opisujących wygląd poszczególnych stron. Zadaniem silnika szablonów jest połączenie szablonu z udostępnionymi przez kod Java danymi. Dane wstawiane są do szablonu w miejscach wskazanych za pomocą konstrukcji języka szablonów. Wynikiem pracy silnika szablonów jest kompletny dokument, który może zostać zaprezentowany użytkownikowi.

Przegląd technologii szablonów

- Podstawowe technologie szablonów to:
  - Velocity
  - FreeMarker
  - WebMacro
- Wszystkie trzy rozwijane jako Open Source
- Liderami są Velocity i FreeMarker
- Zaletą Velocity jest prostota
- Zaletą FreeMarker jest elastyczność

Podstawowe technologie szablonów dla serwletów Java to:

•Velocity (http://jakarta.apache.org/velocity), rozwijana w ramach Apache Jakarta Project;

•FreeMarker (http://freemarker.sourceforge.net), rozwijana przez Visigoth Software Society;

•WebMacro (http://www.webmacro.org), rozwijana przez grupę Justina Wellsa.

Wszystkie trzy wymienione technologie szablonów dla serwletów Java rozwijane są na zasadach open source. Liderami jeśli chodzi o popularność i wsparcie ze strony narzędzi i szkieletów aplikacji są Velocity i FreeMarker. Zaletą Velocity jest prostota, a FreeMarker – elastyczność.

Velocity – Przykład (1/2)

serwlet

1
2
3
4
5

```
public class VelServlet extends VelocityServlet {

protected Template handleRequest (..., Context ctx )
  throws Exception
{
 String s1 = "Dariusz"; String s2 = "Marek";
 String tytul = "Lista imion";
 ctx.put("tytul", tytul);
 Vector listaImion = new Vector();
 listaImion.addElement(s1); listaImion.addElement(s2);
 ctx.put("lista", listaImion);
 Template t = null;
 try {
  t=getTemplate("lista_imion.vm");
 } catch (Exception e) {}
 return t;
}
}
```


Ten i kolejny slajd przedstawiają przykład prostej aplikacji wykorzystującej najpopularniejszą z technologii szablonów – Velocity.

Niniejszy slajd przedstawia kod serwletu, na kolejnym pokazany będzie kod wykorzystywanego przez niego szablonu i efekt działania serwletu widziany w przeglądarce. Znaczenie poszczególnych fragmentów jest następujące:

1. Serwlet tworzony jest jako klasa dziedzicząca z klasy bibliotecznej Velocity o nazwie VelocityServlet
2. Osoba programująca nie implementuje metody doGet(), gdyż wzorcowa klasa VelocityServlet, dostarcza jej odpowiednią implementację. Zadaniem osoby programującej jest implementacja metody handleRequest(), która jest wywoływana z doGet() i ma za zadanie przygotowanie danych do zagnieżdżenia w szablonie i wskazanie szablonu
3. Dane do zagnieżdżenia w szablonie są pod wybraną nazwą rejestrowane w obiekcie kontekstu Velocity. Można powiedzieć, że kontekst stanowi nośnik danych między kodem Java a szablonem. W tym wypadku w kontekście umieszczany jest tytuł strony
4. W kontekście umieszczona jest kolekcja imion
5. Szablon jest wskazywany za pomocą metody getTemplate(). Szablony w Velocity są zwyczajowo składowane w pliku z rozszerzeniem *.vm. Lokalizacja, z której będą pobierane szablony, zależy od konfiguracji silnika Velocity

Velocity – Przykład (2/2)

szablon

```
<html><head><title>
Lista imion w Velocity
</title></head>
<body>
<h2>$tytul</h2>
<table border="1">
<tr>
<th>Lp.</th><th>Imię</th>
</tr>
#foreach ($imie in $lista)
  <tr>
  <td>$velocityCount</td>
  <td>$imie</td>
  </tr>
#end
</table>
</body></html>
```

1

2

3

wynik działania serwletu

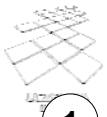
Niniejszy slajd prezentuje przykładowy szablon, wykorzystywany przez serwlet przedstawiony na poprzednim slajdzie oraz końcowy efekt działania serwletu, zaobserwowany w przeglądarce. Szablon ma postać dokumentu HTML z zagnieżdżonymi referencjami i dyrektywami języka szablonów Velocity. Znaczenie poszczególnych fragmentów kodu jest następujące:

1. Referencja wskazująca miejsce w szablonie, w którym ma być umieszczony tekst zarejestrowany w kontekście pod nazwą "tytul".
2. Pętla przeglądająca kolekcję imion dostępną w kontekście pod nazwą "lista". Znaczniki zawarte w ciele pętli będą powtórzone w wynikowym dokumencie tyle razy, ile będzie przebiegów pętli. W każdym przebiegu pętli kolejny element kolekcji jest przypisywany do zmiennej "imie"
3. Referencje wskazujące miejsca w szablonie, w których mają być umieszczone: wartość wewnętrznego licznika pętli ("velocityCount") i bieżąca wartość zmiennej "imie".

WebMacro

- Szkielet tworzenia serwletów
  - pełna separacja kodu programu i kodu wynikowego (HTML, XML, txt)
  - wykorzystanie szablonów
  - wymuszenie architektury MVC
  - mechanizm introspekcji

WebMacro to technologia szkieletowa tworzenia serwletów. Podstawowe cechy tej technologii to przede wszystkim całkowita separacja kodu serwletu od kodu wynikowego. Narzędziem zapewniającym separację są szablony przygotowywane za pomocą prostego języka skryptowego. Serwlet jest odpowiedzialny za utworzenie kontekstu i przygotowanie danych do prezentacji oraz umieszczenie danych w kontekście. Danymi mogą być zarówno zmienne typów bazowych i proste obiekty Java, jak i struktury złożone, np. kolekcje. W kolejnym kroku serwlet ładuje przygotowany wcześniej szablon, definiujący wygląd kodu wynikowego. Kodem wynikowym może być kod HTML, XML, czysty tekst, a nawet kod źródłowy (WebMacro z powodzeniem wykorzystuje się jako preprocesor kodu źródłowego). Prosty język skryptowy umożliwia umieszczenie danych pobranych z kontekstu we właściwych fragmentach szablonu. Język skryptowy zapewnia podstawowe konstrukcje, takie jak przypisania, instrukcje warunkowe czy iteracje po zawartości kolekcji. Ostatni etap to utworzenie kodu wynikowego na podstawie szablonu i dostępnych danych. Taka architektura zapewnia pełną separację warstw odpowiedzialnych za kontroler i model od warstwy prezentacji, a co za tym idzie, w naturalny sposób wymusza wzorzec projektowy Model-View-Controller (MVC). Łączenie zmiennych z szablonu z obiektami języka Java odbywa się poprzez introspekcję, której zasady przypominają zasady introspekcji w języku wyrażeń JSP EL.



WebMacro – przykład

WMServlet.java

```
1  WebMacro wm = new WebMacro();
   FastWriter out = wm.getFastWriter();
2  Context ctx = wm.getContext();
   ctx.put("SQLQuery","SELECT * FROM products");
   Result[] results = ... // zapytanie do bazy danych
3  ctx.put("SQLResult",results);
4  Template tmpl = wm.getTemplate("search.view");
   tmpl.write(out,ctx);
   out.flush();
```

search.view

```
<h1>Here are the results for $query:</h1>
  <table>
  #foreach $result in $results {
    <tr><td>$result.Number</td>
    <td><a href="$result.Link">$result.Name</a></td></tr> }
  </table>
```


Slajd przedstawia przykładową prostą aplikację WebMacro. Serwlet WMServlet.java tworzy nowy obiekt WebMacro i obiekt FastWriter reprezentujący strumień wyjściowy (1). Następnie pobierany jest kontekst (2). W kroku (3) w kontekście umieszczane są różne dane. Może to być prosty łańcuch znaków zawierający treść polecenia SQL (dana "SQLQuery"), lub kolekcja obiektów zawierająca wynik zapytania (dana "SQLResult"). Ostatni etap pracy serwletu (4) to załadowanie szablonu z pliku zewnętrznego i wywołanie metody write() z przekazaniem kontekstu (i znajdujących się w kontekście danych) oraz strumienia wyjściowego. Ostatnia linia jest obowiązkowa i powoduje wyczyszczenie bufora i przesłanie wyniku do klienta HTTP. Szablon search.view zawiera praktycznie czysty kod HTML z kilkoma prostymi konstrukcjami. Instrukcja #foreach powoduje iterację po wszystkich elementach podanej kolekcji. W każdej iteracji do zmiennej $result przypisywany jest kolejny obiekt z kolekcji $results, której zawartość mieści się w kontekście.

FreeMarker

- Technologia bardzo podobna do WebMacro i Velocity
  - biblioteka Java do tworzenia tekstowych plików wynikowych (HTML, XML, RTF, kod źródłowy, itp.)
- Cechy
  - uniwersalność
  - zaawansowane szablony
  - wsparcie dla technologii webowych
  - internacjonalizacja i lokalizacja
  - przetwarzanie XML

FreeMarker to technologia szablonów bardzo podobna do WebMacro i Velocity. FreeMarker nie jest osobnym narzędziem, jest to biblioteka języka Java umożliwiające tworzenie tekstowych plików wynikowych o dowolnym formacie (HTML, XML, RTF, txt, kod źródłowy języka programowania). FreeMarker nie jest ograniczony tylko do środowiska serwletów i może być wykorzystywany w dowolnej aplikacji języka Java. FreeMarker jest narzędziem uniwersalnym, może być wykorzystywany w wielu typach aplikacji, może generować wiele typów plików wynikowych, wreszcie szablony mogą być ładowane z wielu lokalizacji (plik lokalny, plik zdalny, baza danych). FreeMarker oferuje zaawansowany mechanizm tworzenia szablonów i rozbudowany język skryptowy obejmujący dyrektywy, zmienne, wyrażenia, makra, przestrzenie nazw, instrukcje iteracji i instrukcje warunkowe. FreeMarker posiada wbudowane mechanizmy obsługi typowych zadań związanych z aplikacjami internetowymi: obsługa znaków niestandardowych w HTML, wsparcie architektury MVC, obsługa bibliotek znaczników JSP. Aplikacje przygotowane w FreeMarker poddają się łatwo internacjonalizacji i lokalizacji (formatowanie dat i liczb). FreeMarker posiada specjalny zestaw znaczników umożliwiający zaawansowane operacje na danych źródłowych przechowywanych w plikach XML.

FreeMarker – przykład szablonu

```
<html>
  <body>
  <#-- komentarz FreeMarker -->  (1)
  <h1>Tutaj ${interpolacja}</h1>  (2)

  <b>Lista miast</b>
  <ul>
    <#list cities as city>  (3)
      <li>${city.name}: population #{city.population; m1}  (4)
    </#list>
  </ul>
  </body>
</html>
```


Slajd przedstawia przykład prostego szablonu FreeMarker. Dodatkowe znaczniki mają postać <#znacznik> </#znacznik>. Komentarz umieszczony jest w znacznikach <#-- --> (1). Interpolacja to ewaluacja wartości wyrażenia, przykład zastosowania interpolacji (czyli znalezienia i przekazania do szablonu wartości zmiennej interpolacja. Przykład interpolacji w kroku (2). FreeMarker umożliwia także łatwą nawigację po kolekcji za pomocą znaczników <#list></#list> (3). W przeciwieństwie do zwyczajnej interpolacji (oznaczonej przez ${}) interpolacja numeryczna (oznaczona przez #{}) umożliwia podanie formatu danych źródłowych (4). Poza elementami przedstawionymi na slajdzie FreeMarker oferuje także zestaw funkcji do operowania na liczbach i łańcuchach znaków, zestaw operatorów (w tym operator przypisania <#assign>)

Materiały dodatkowe

- JSP, http://java.sun.com/products/jsp
- Velocity, http://jakarta.apache.org/velocity/index.html
- WebMacro, http://www.webmacro.org
- FreeMarker, http://freemarker.sourceforge.net