JavaServer Pages

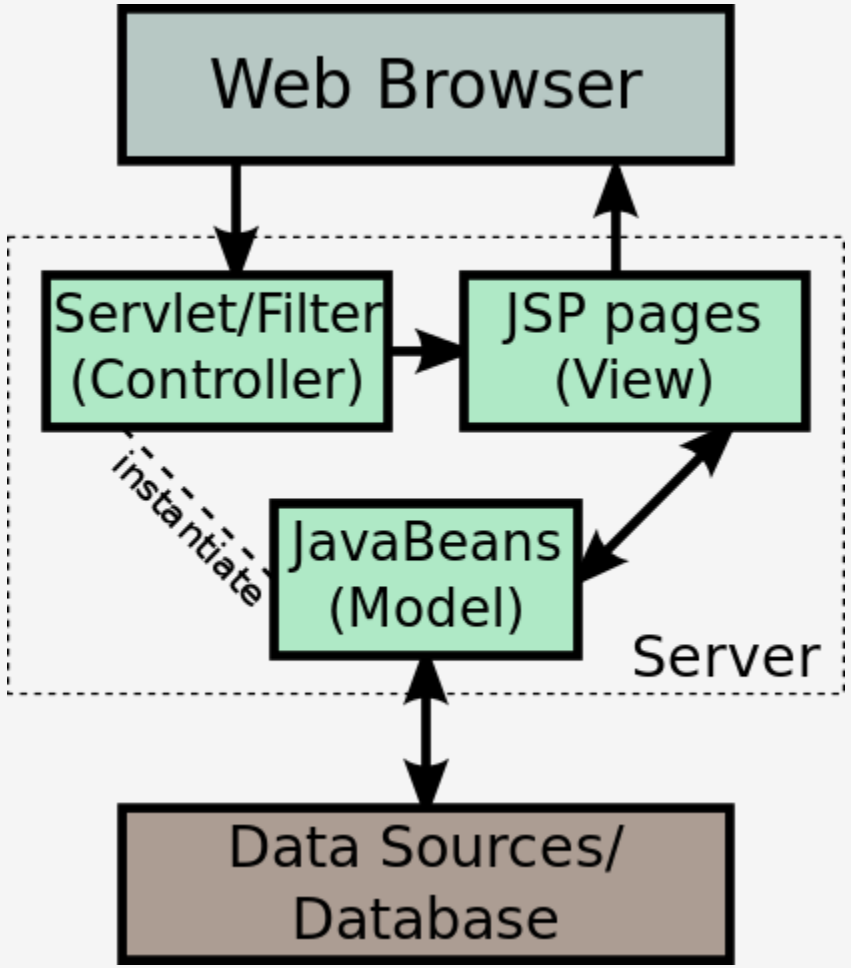
Jakub Szlenk

JSP - Wprowadzenie do technologii JSP

- Technologia JavaServer Pages jest integralną częścią J2EE - platformy aplikacji klasy enterprise wprowadzoną w 1999 roku
- Umożliwia generowanie dynamicznych stron internetowych
- Kod stron JSP jest w większości kodem HTML
- Dodatkowo posiada znaczniki JSP i JSTL oraz kod Javowy, które „ożywiają” stronę
- Dzięki temu (m.in.) separuje się warstwę prezentacji od warstwy logiki biznesowej, a kod HTML może być przygotowywany oddzielnie przez webmastera przy użyciu dedykowanych narzędzi
- Kod strony JSP konwertowany jest (automatycznie) do servletu
- Do uruchomienia strony JSP wymagają serwera webowego, np. Tomcata
- Zalecane rozszerzenia plików JSP, to: .jsp, .jspx, .jspf

JSP - Wprowadzenie do technologii JSP

Model



JSP jako metoda separacji widoku od serwletów

- Pojedyncze żądanie może być obsługiwane przez więcej, niż jeden serwlet

- W tym celu wykorzystuje się mechanizm **dyspozytora żądań** (ang. request dispatcher)

- Dyspozytor żądań może wykonać:

- przekierowanie (forward), jeśli chcemy, aby inny zasób zajął się realizacją żądania
- dołączenie (include), jeśli chcemy dołączyc efekt działania innego zasobu dla danego żądania

JSP jako metoda separacji widoku od serwletów

- Przekierowanie sprawia, że oryginalny serwlet przestaje zajmować się obsługą danego żądania
- Dołączenie pozwala na połączenie efektów działania dwóch lub więcej serwletów
- Zasobami, które mogą być dołączane przy użyciu dyspozytora, są inne serwlety, pliki HTML, a także **strony JSP**.
- Strony JSP stanowią receptę na podstawowy problem serwletów – dynamiczne generowanie treści do klienta
- Mimo że serwlet umożliwia wygodny dostęp do poszczególnych elementów żądania i odpowiedzi, generowanie w nim kodu HTML stanowi istną udrękę
- Strony JSP zawierają z reguły kod HTML połączony ze specjalnymi konstrukcjami JSP, które pozwalają na dynamiczne generowanie treści

JSP jako metoda separacji widoku od serwletów

- JSP można używać bezpośrednio, podając ścieżkę do pliku strony
- Bardziej funkcjonalne jest połączenie możliwości serwletów i stron JSP:

- Serwlet zajmuje się przetworzeniem danych i logiką biznesową, a następnie przekierowuje żądanie do strony JSP

- Strona JSP zajmuje się jedynie wyświetleniem uprzednio przygotowanych danych

•Plik JSP jest w momencie kompilacji przekształcany na serwlet – tak więc koniec końców wszystkie strony JSP stają się serwletami

•Zapis przypominający plik HTML jest po prostu ukłonem w stronę twórców warstwy interfejsu użytkownika

Przekazywanie informacji do stron JSP

- Tym, co odróżnia strony JSP od plików HTML (tudzież innych statycznych zasobów) jest możliwość korzystania z danych aplikacji
- Kluczową funkcjonalność stanowi więc wyświetlanie i ogólnie pojęte wykorzystanie informacji zawartych w żądaniu, sesji i kontekście aplikacji

Cykl życia strony JSP

- Kompilacja
- parsowanie
- konwersja do servletu
- kompilacja servletu
- Inicjalizacja
- wywołanie metody jsp**Init**()
- Przetwarzanie żądań
- wywoływanie metody _jsp**Service**(request, response)
- Zwalnianie zasobów
- wywołanie metody jsp**Destroy**()

Przetwarzanie żądania

1. Przeglądarka wysyła żądanie (*request*) HTTP do serwera
2. Serwer rozpoznaje żądanie strony JSP i przekazuje sterowanie do silnika JSP
3. Silnik JSP zaczytuje stronę JSP z dysku i konwertuje ją do servletu
4. Kod servletu jest kompilowany i uruchamiany
5. Servlet przygotowuje odpowiedź w formacie HTML
6. Kod HTML przekazywany jest do serwera
7. Serwer odsyła odpowiedź (*response*) HTTP do przeglądarki

Konwersja i kompilacja z punktów 3-4 najczęściej zachodzi podczas instalacji aplikacji na serwerze, tak aby pierwsze żądanie nie było obarczone koniecznością oczekiwania na te operacje

Zmienne zdefiniowane (1)

- **request** (ServletRequest) - używany do pobierania informacji o żądaniu (parametrów, atrybutów)
- **response** (ServletResponse) - reprezentuje odpowiedź HTTP; umożliwia wysłanie nagłówka, ustawienie ciastka
- **session** (HttpSession) - trzyma informacje o sesji użytkownika
- **out** (JspWriter) - służy do wypisywania tekstu na stronie

Zmienne zdefiniowane (2)

- **application** (ServletContext) - przechowuje atrybuty związane z servletem dostępne dla wszystkich użytkowników
- **config** (ServletConfig) - przechowuje parametry inicjalizujące
- **pageContext** (PageContext) - to samo co jspContext oraz dodatkowe metody dedykowane servletom
- **page** (Object) - zwraca referencje do aktualnej strony JSP, synonim this

Elementy strony - wyrażenia

- Wyrażenia **<%= expression %>**

- ich wartość zostanie obliczona i wstawiona do danych wyjściowych servletu

- kontener pobiera zawartość pomiędzy <%= a %> i umieszcza jako argument metody print() klasy JspWriter (PrintWriter): out.print()

```
<html>
  <head>
    <title>Dzisiaj jest...</title>
  </head>
  <body>
    <p>Dzisiaj jest: <%= java.time.LocalDate.now().toString()%></p>
  </body>
</html>
```

Elementy strony - skryptlety

- Skryptlety **<% code %>**

- mogą zawierać dowolną ilość kodu Javy

- oferują dostęp do zmiennych zdefiniowanych dla stron JSP

- mogą służyć np, do ustawiania nagłówków odpowiedzi oraz kodów stanu, komunikacji z bazą danych i innych złożonych instrukcji

```
<html>
  <head>
    <title>Dzisiaj jest...</title>
  </head>
  <body>
    <p>Dzisiaj jest: <%= java.time.LocalDate.now().toString()%></p>
  </body>
</html>
```

Elementy strony – dyrektywy (1)

- Dyrektywy **<%@ directive attribute=„value”… %>**

- kontrolują proces translacji strony

- **page** - atrybuty związane ze stroną

. **import** - pozwala na określenie klas i pakietów, które zostaną zaimportowane do servletu wygenerowanego ze strony JSP

. **info** - definiuje, co zwróci metoda getServletInfo()

. **isELIgnored** - określa, czy ignorować wyrażenia EL

. **isScriptingEnabled** - określa, czy ignorować elementy skryptowe

Elementy strony – dyrektywy (2)

. **pageEncoding** - specyfikuje kodowanie strony

. **contentType** - określa nagłówek odpowiedzi o nazwie Content-Type

. **isThreadSafe** - wskazuje, czy wygenerowany servlet ma implementować interfejs SingleThreadModel

. **session** - definiuje, czy podłączać stronę pod sesję HTTP

. **extends** - określa, po której klasie servlet ma dziedziczyć

Elementy strony – dyrektywy (3)

. **buffer** - definiuje rozmiar bufora wykorzystywanego przez zmienną out klasy JspWriter

. **autoflush** - kontroluje, czy bufor przechowujący dane wyjściowe powinien być automatycznie wysyłany w przypadku przepełnienia, czy generowany ma być wyjątek

. **errorPage** - wskazuje stronę JSP, która powinna obsłużyć wyjątek rzucony (i nie obsłużony) na bieżącej stronie

. **isErrorPage** - określa, że bieżąca strona jest stroną obsługi błędów występujących na innych stronach

Elementy strony – dyrektywy (4)

- include

. <%@include file=„embedded.jspf”%>

. pozwala dołączyć plik, np. fragment (segment) strony JSP

. dołączana strona nie musi być *well-formed*, ponieważ nie będzie przetwarzana przed załączeniem

- taglib

. <%@taglib uri=„http://foo.com/ftags” prefix=„f”%>

. deklaruje użycie biblioteki tagów

Elementy strony – deklaracje

- Deklaracje **<%! code %>**

- pozwalają na zdefiniowanie metod oraz pól, które zostaną wstawione do głównego ciała klasy servletu

- używane zazwyczaj z wyrażeniami lub skryptletami

```
<%@page language="java" contentType="text/html; charset=UTF-8" %>
<%@page pageEncoding="UTF-8" %>

<%! private long visitCount = 0; %>

<h2>Ilość odwiedzin strony: <%= ++visitCount%></h2>
```

Elementy strony – deklaracje

- Deklaracje **<%! code %>**

- pozwalają na zdefiniowanie metod oraz pól, które zostaną wstawione do głównego ciała klasy servletu

- używane zazwyczaj z wyrażeniami lub skryptletami

```
<%@page language="java" contentType="text/html; charset=UTF-8" %>
<%@page pageEncoding="UTF-8" %>

<%! private long visitCount = 0; %>

<h2>Ilość odwiedzin strony: <%= ++visitCount%></h2>
```

Elementy strony – komentarze (1)

- Komentarze **<%— text —%>**

- służą do umieszczania dodatkowych informacji na temat kodu pisanego w JSP

- tekst komentarza nie będzie dostępny w kodzie źródłowym servletu wygenerowanego ze strony JSP ani w kodzie HTML w przeglądarce - w odróżnieniu od komentarza HTML

```
<%@page language="java" contentType="text/html; charset=UTF-8" %>
<%@page pageEncoding="UTF-8" %>

<html>
<head>
    <title>Komentarze JSP i HTML</title>
</head>
<body>
<h1>Komentarze JSP i HTML</h1>
<%-- Komentarz JSP --%>
<!-- komentarz HTML -->
</body>
</html>
```

Elementy strony – komentarze (2)

- komentarze

```
out.write("    <html>\n");
out.write("    <head>\n");
out.write("        <title>Komentarze JSP i HTML</title>\n");
out.write("    </head>\n");
out.write("    <body>\n");
out.write("    <h1>Komentarze JSP i HTML</h1>\n");
out.write("    ");
out.write("\n");
out.write("    <!-- komentarz HTML -->\n");
out.write("    </body>\n");
out.write("    </html>\n");
```

```
<html>
<head>
    <title>Komentarze JSP i HTML</title>
</head>
<body>
<h1>Komentarze JSP i HTML</h1>

<!-- komentarz HTML -->
</body>
</html>
```

Elementy strony – język wyrażeń (1)

- Język wyrażeń

- ${wyrażenie}

. natychmiastowa ewaluacja

. tylko do odczytu

. obliczane po pierwszym wyświetleniu strony

- #{wyrażenie}

. do odczytu i zapisu

. obliczane w dowolnym etapie cyklu życia strony

- Można ich używać w statycznym tekście oraz w atrybutach znaczników, które to umożliwiają
- Wyrażenie jest automatyczne konwertowane do potrzebnego typu

Elementy strony – język wyrażeń (2)

- Dostęp do predefiniowanych zmiennych

applicationScope	Map<String, Object>
cookie	Map<String, Cookie>
header	Map<String, String>
headerValues	Map<String, String[]>
initParam	Map<String, String>
pageContext	PageContext
pageScope	Map<String, Object>
param	Map<String, String>
paramValues	Map<String, String[]>
requestScope	Map<String, Object>
sessionScope	Map<String, Object>

Elementy strony – znaczniki akcji (1)

- Znaczniki akcji **<jsp:action attribute=„value”…></jsp:action>**
- Mogą być zagnieżdżone
- Są przetwarzane podczas żądania strony i umożliwiają wykonanie poniższych operacji:

•forward

- <jsp forward page=„other.jsp”/>
- przerywa przetwarzanie bieżącej strony i przekazuje request do wskazanej strony

•include

- <jsp:include page=„footer.jsp”/>
- dołącza stronę JSP do bieżącej strony; dołączana strona musi być *well-formed*, ponieważ dołączany jest wygenerowany przez nią kod HTML

•param

- <jsp:param name=„id” value=„5”/>
- pozwalają dodać parametry dla strony (servletu) wywoływanych przy użyciu jsp:forward lub jsp:include

Elementy strony – znaczniki akcji (2)

- **useBean**

- <jsp:useBean id=„name” class=„package.class” type=„object type” scope=„page|request|session|application”/>
- umożliwia korzystanie ze standardowych JavaBeans

- **setProperty**

- <jsp:setProperty name=„beanId” property=„beanPropertyName” value=„value to set”/>
- umożliwia ustawienie wartości pola beana; bean musi być wcześniej zdefiniowany poprzez <jsp:useBean…/>

- **getProperty**

- <jsp:getProperty name=„beanId” property=„beanPropertyName”/>
- umożliwia pobranie wartości pola beana

Język wyrażeń EL

- Głównym narzędziem do interakcji z danymi aplikacji jest język wyrażeń (Expression Language)
- Język ten pozwala na dostęp do komponentów JavaBean, map i list
- Podstawowe wyrażenie ma następującą składnię:

${uzytkownik.adres.ulica}

- Aby skorzystac z danych zawartych w różnych zasięgach, w wyrażeniach EL należy odwołać się do następujących obiektów:

pageScope, requestScope, sessionScope, applicationScope – pozwala na dostęp do atrybutów zawartych w zasięgach (odpowiednio) strony, żądania, sesji i aplikacji (kontekstu)

Język wyrażeń EL

Najważniejsze obiekty dostępne w ramach wyrażeń języka EL:

param – mapa (słownik) parametrów żądania GET/POST

header – mapa nagłówków

paramValues, headerValues – wersje tablicowe, zwracają wiele wartości dla danego klucza (o ile dany klucz wystąpił w nagłówkach lub parametrach wielokrotnie)

cookie – mapa ciasteczek

initParam – mapa parametrów inicjalizacji kontekstu aplikacji

pageContext – kontekst strony, udostępnia:

- request – obiekt żądania (nie mylić z samodzielnym request – mapą atrybutów o zasięgu żądania!)
- session – obiekt sesji
- response – obiekt odpowiedzi
- servletContext – kontekst aplikacji webowej

Język wyrażeń EL

• Jeśli wyrażenie zawiera obiekt inny, niż jeden z wymienionych, nastąpi próba odnalezienia podanego obiektu w poszczególnych zasięgach, w kolejności:

page, request, session, application

Klucz obiektu w mapie może być zapisany bezpośrednio w kodzie wyrażenia, np.:

${koszyk["liczba.produktow"]}

Język wyrażeń EL

• Poza bezpośrednim odwołaniem do map i kolekcji, możliwe jest również wykorzystywanie obiektów, spełniających założenia konwencji JavaBeans

• Aby spełnić wymagania JavaBean, klasa musi:

- zawierać bezparametrowy konstruktor publiczny
- udostępniać swoje dane w formie metod getXXX()/setXXX() (ewentualnie isXXX())

Język wyrażeń EL

- Poza prostymi odwołaniami do konkretnych obiektów, istnieje możliwość tworzenia wyrażeń przy użyciu operatorów:

o+, -, *, /, mod, div

and, &&, or, ||, not, !

==, !=, <>, >, <, >=, <=, eq, ne, gt, lt, ge, le

empty, ?:

Średnia cena produktu: ${koszyk.suma div koszyk.liczbaProduktow}

Akcje, czyli specjalne znaczniki JSP

- Akcje umożliwiają wykonanie pewnych czynności przy użyciu znaczników JSP
- Znaczniki te są zapisywane tak, jak wszystkie inne znaczniki, ale należy poprzedzac je standardowym, domyślnym prefiksem jsp
- jsp:include, jsp:forward – pozwalają na zachowanie identyczne, jak w przypadku metod include() i forward() dyspozytora żądań w serwletach
- jsp:param – pozwala na przekazywanie parametrów podczas wykonywania akcji include i forward

```
<jsp:include page="naglowek.jsp">
<jsp:param name="tytul" value="Strona logowania" />
</jsp:include>
```

Akcje, czyli specjalne znaczniki JSP

- jsp:useBean – służy do pobrania ziarna (komponentu) o określonym id z pewnego zasięgu

- jsp:getProperty – pozwala na wyświetlenie dowolnej właściwości dowolnego ziarna użytego za pomocą znacznika jsp:useBean

Przykład:

<jsp:useBean id=*"uzytkownik" scope="session" class="encje.Uzytkownik" />*

<jsp:getProperty name=*"uzytkownik" property="nazwisko" />*

Dyrektywy JSP

- Dyrektywy JSP stanowią przykład szerokiej grupy konstrukcji, używanych od początku istnienia JSP, umieszczanych wewnątrz znaczników <%...%>
- Większość z tych znaczników jest obecnie uważana za przestarzałe i nie należy z nich korzystać!

- Istnieją trzy dyrektywy JSP:

page – służy do określania różnych atrybutów strony, m.in. kodowania znaków, adresu strony błędu, importu określonych pakietów

taglib – służy do zaimportowania biblioteki znaczników, a także pozwala na określenie prefiksu, z jakim będą używane znaczniki

include – pozwala na dołączenie treści pliku – statycznego, jak i dynamicznego

Dyrektywy JSP

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>

<%@ include file="/WEB-INF/naglowek.html" %>

Legacy code

Skryptlety (*scriptlets*) pozwalały na wywołanie dowolnego kodu:

(out – strumień tekstowy odpowiedzi):

```
<body>
<%
out.println("W skryptletach można umieszczać kod w języku Java!");
%>
</body>
```

Legacy code

• Wyrażenia (*expressions*) to fragmenty kodu zwracające wartość do natychmiastowego wyświetlenia:

<%=request.getAttribute("nazwaUzytkownika") %>

• Treść wyrażenia stanowi argument dla metody out.println()

Legacy code

• Wyrażenia (*expressions*) to fragmenty kodu zwracające wartość do natychmiastowego wyświetlenia:

<%=request.getAttribute("nazwaUzytkownika") %>

• Treść wyrażenia stanowi argument dla metody out.println()

•Deklaracje pozwalały na dodanie do danej strony (a co za tym idzie – serwletu) metod i pól:

<%! **int liczbaUzytkownikow = 3; %>**

<%=liczbaUzytkownikow%>

JSTL – krótka charakterystyka

- JavaServer Pages Standard Tag Library – biblioteka znaczników rozszerzająca możliwości standardu JSP
- JavaServer Pages Standard Tag Library (JSTL), to zbiór użytecznych tagów JSP, które udostępniają wiele funkcjonalności

wykorzystywanych przy pisaniu stron JSP

- Aby móc korzystać z biblioteki tagów, należy zainkludować ją przy użyciu dyrektywy taglib
- Popularne biblioteki
- Core tags
- Formatting tags
- SQL tags
- XML tags
- JSTL functions

JSTL – krótka charakterystyka

Core tags

```
<c:
```

c:catch	http://java.sun.com/jsp/jstl/core
c:choose	http://java.sun.com/jsp/jstl/core
c:forEach	http://java.sun.com/jsp/jstl/core
c:forTokens	http://java.sun.com/jsp/jstl/core
c:if	http://java.sun.com/jsp/jstl/core
c:import	http://java.sun.com/jsp/jstl/core
c:otherwise	http://java.sun.com/jsp/jstl/core
c:out	http://java.sun.com/jsp/jstl/core
c:param	http://java.sun.com/jsp/jstl/core
c:redirect	http://java.sun.com/jsp/jstl/core
c:remove	http://java.sun.com/jsp/jstl/core
c:set	http://java.sun.com/jsp/jstl/core
c:url	http://java.sun.com/jsp/jstl/core
c:when	http://java.sun.com/jsp/jstl/core

JSTL – krótka charakterystyka

Formatting tags

```
<fmt:
```

fmt:bundle	http://java.sun.com/jsp/jstl/fmt
fmt:formatDate	http://java.sun.com/jsp/jstl/fmt
fmt:formatNumber	http://java.sun.com/jsp/jstl/fmt
fmt:message	http://java.sun.com/jsp/jstl/fmt
fmt:param	http://java.sun.com/jsp/jstl/fmt
fmt:parseDate	http://java.sun.com/jsp/jstl/fmt
fmt:parseNumber	http://java.sun.com/jsp/jstl/fmt
fmt:requestEncoding	http://java.sun.com/jsp/jstl/fmt
fmt:setBundle	http://java.sun.com/jsp/jstl/fmt
fmt:setLocale	http://java.sun.com/jsp/jstl/fmt
fmt:setTimeZone	http://java.sun.com/jsp/jstl/fmt
fmt:timeZone	http://java.sun.com/jsp/jstl/fmt

JSTL – krótka charakterystyka

SQL tags

```
<sql:
```

```
sql:dateParam          http://java.sun.com/jsp/jstl/sql
sql:param              http://java.sun.com/jsp/jstl/sql
sql:query              http://java.sun.com/jsp/jstl/sql
sql:setDataSource      http://java.sun.com/jsp/jstl/sql
sql:transaction        http://java.sun.com/jsp/jstl/sql
sql:update             http://java.sun.com/jsp/jstl/sql
```

JSTL – krótka charakterystyka

XML tags

```
<x:
    choose      http://java.sun.com/jsp/jstl/xml
    forEach     http://java.sun.com/jsp/jstl/xml
    if          http://java.sun.com/jsp/jstl/xml
    otherwise   http://java.sun.com/jsp/jstl/xml
    out         http://java.sun.com/jsp/jstl/xml
    param       http://java.sun.com/jsp/jstl/xml
    parse       http://java.sun.com/jsp/jstl/xml
    set         http://java.sun.com/jsp/jstl/xml
    transform   http://java.sun.com/jsp/jstl/xml
    when        http://java.sun.com/jsp/jstl/xml
```

JSTL – krótka charakterystyka

Functions

```
<${fn}
λ fn:contains(String input, String substring)                    boolean
λ fn:containsIgnoreCase(String input, String substring)          boolean
λ fn:endsWith(String input, String substring)                    boolean
λ fn:escapeXml(String input)                                     java.lang.String
λ fn:indexOf(String input, String substring)                     int
λ fn:join(String[] array, String separator)                      java.lang.String
λ fn:length(Object obj)                                          int
λ fn:replace(String input, String substringBefore,…              java.lang.String
λ fn:split(String input, String delimiters)                      java.lang.String[]
λ fn:startsWith(String input, String substring)                  boolean
λ fn:substring(String input, int beginIndex, int e…              java.lang.String
λ fn:substringAfter(String input, String substring)              java.lang.String
λ fn:substringBefore(String input, String substrin…              java.lang.String
λ fn:toLowerCase(String input)                                   java.lang.String
λ fn:toUpperCase(String input)                                   java.lang.String
λ fn:trim(String input)                                          java.lang.String
```

JSP – Client Request

- Obiekt requestu zawiera wszystkie informacje związane z żądaniemużytkownika
- W protokole HTTP dane te przesyłane są poprzez nagłówki (headers) i treść wiadomości (message body)
- HttpServletRequest

- treść wiadomości
- Parametry
- atrybuty
- nagłówki
- ścieżki
- ciasteczka
- kodowanie

```
iMac:~ rafos$ telnet localhost 8080
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.43.0
Accept: */*
```

JSP – Client Response

- Obiekt response zawiera wszystkie informacje związane z odpowiedzią servera do klienta
- W protokole HTTP dane te przesyłane są poprzez nagłówki (headers) i treść wiadomości (message body)
- HttpServletResponse
  - treść odpowiedzi
  - nagłówki
  - ciasteczka
  - kodowanie

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=D8825E65DC9681319B83935EBB05BAB1; Path=/; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Length: 54
Date: Fri, 12 Feb 2016 23:38:23 GMT



<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

JSP – Metoda GET (1)

- http://localhost:8080/search.jsp?query=jsp&page=3&sort=desc
- Domyślna metoda wysyłania informacji z przeglądarki do serwera
- Przesyłane parametry dołączane do żądania po znaku ?
- Poszczególne serwery wprowadzają (konfigurowalny) limit na długość żądania
- Parametry GET żądania dostępne są po stronie serwera w zmiennej QUERY_STRING
- Poszczególne parametry dostępne są za pomocą metod

- request.getParameter(„nazwaParametru”)
- request.getParameterNames()
- request.getParameterMap()

JSP– Metoda GET (1)

GET


```
<body>
<h2>Parametry szukania</h2>
<p>
    Szukane słowo: <b><%=request.getParameter("query")%></b><br/>
    Strona: <b><%=request.getParameter("page")%></b><br/>
    Sortowanie: <b><%=("desc".equals(request.getParameter("sort")) ? "malejąco" : "rosnąco")%></b><br/>
</p>
</body>
```

JSP – Metoda POST (1)

- Podobnie jak metoda POST służy do przesyłania danych z przeglądarki do serwera
- Aby wysłać dane metodą POST wykorzystuje się formularze HTML, gdzie atrybut method formularza ustawiamy na post
- Metody POST używa się, gdy odwołanie jest formą interakcji z użytkownikiem; operacja wywołuje zmiany na serwerze; operacja nie jest idempotentna; uploaduje się plik na serwerze
- Parametry przesyłane są w treści żądania, a nie po znaku ?
- Poszczególne parametry dostępne są za pomocą metod

- request.getParameter(„nazwaParametru”)
- request.getParameterNames()
- request.getParameterMap()

JSP – Metoda GET (1)

POST

```
<form action="search.jsp" method="POST">
    Słowo: <input type="text" name="query"/><br/>
    Strona: <input type="text" name="page"/><br/>
    Sortowanie:
    <select name="sort">
        <option value="asc">rosnąco</option>
        <option value="desc">malejąco</option>
    </select><br />
    <input type="submit" value="Submit"/>
</form>
```

JSP – Ciasteczka (1)

- Ciasteczka to małe tekstowe informacje trzymane po stronie klienta
- Działanie mechanizmu ciasteczek można podzielić na trzy fazy

- serwer (w odpowiedzi) wysyła ciasteczko do przeglądarki klienta
- przeglądarka zapisuje ciasteczko (na dysku)
- podczas kolejnego żądania do serwera, przeglądarka odsyła ciasteczko wraz z żądaniem

```
Cookie cookie = new Cookie("searchId", String.valueOf(2138902773));
cookie.setMaxAge(60*60*24);
response.addCookie(cookie);
```

JSP – Ciasteczka (2)

cook

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=C97D0F33B6CFFF25B0400618FAD89106; Path=/; HttpOnly
Set-Cookie: searchId=2138902773; Expires=Mon, 15-Feb-2016 14:54:00 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 450
Date: Sun, 14 Feb 2016 14:54:00 GMT
```

```
request.getCookies(); // ... and next get searchId cookie
%>CookieName: <b><%=cookie.getName()%></b>, CookieValue: <b><%=cookie.getValue()%></b><%
```

JSP – Mechanizm sesji (1)

- Protokół HTTP jest bezstanowy
- Bez dodatkowych mechanizmów, serwer nie jest w stanie zidentyfikować (pamiętać) użytkownika podczas kilku jego żądań
- W celu identyfikacji użytkownika powstał mechanizm sesji
- W JSP realizuje go interfejs HttpSession
- Domyślnie strony JSP wspierają sesję
- Aby wyłączyć obsługę sesji na stronie należy użyć dyrektywy page: <%@ page session=„false” %>
- Identyfikator sesji jest nadawany przez server i ustawiany w ciasteczku o nazwie JSESSIONID

JSP – Mechanizm sesji (2)

- **getId**() - zwraca identyfikator sesji
- **getCreationTime**() - zwraca timestamp utworzenia sesji
- **isNew**() - zwraca true, jeśli użytkownikowi została przydzielona właśnie sesja
- **getLastAccessedTime**() - zwraca timestamp ostatniego żądania użytkownika z zalogowaną sesją
- **getAttribute**(String name) - zwraca atrybut sesji o danej nazwie
- **setAttribute**(String name, Object value) - ustawia atrybut w sesji
- **removeAttribute**(String name) - usuwa atrybut z sesji
- **getMaxInactiveInterval**() - zwraca maksymalny czas utrzymywania sesji bez żądań użytkownika
- **invalidate**() - inwaliduje sesję i usuwa wszystkie obiekty z nią związane

Znaczniki bazowe (rdzenia)

- Instrukcja importująca (JSP):

<%@ taglib uri="*http://java.sun.com/jsp/jstl/core*" *prefix="c"* %>

- Zbiór znaczników, które pozwalają na zmianę przepływu sterowania w ramach widoku

Znaczniki bazowe (rdzenia)

- c:set – umożliwia ustawienie wartości dla całego obiektu lub jego wybranej właściwości:

<c:set value=*"12345" var="identyfikator" scope="request" />* <c:set value=*"Kowalski" target="${uzytkownik}" property="nazwisko" />*

Znaczniki bazowe (rdzenia)

- c:catch – przechwytuje wyjątki rzucone w kodzie umieszczonym wewnątrz tego znacznika:

```
<c:catch var="wyjatek"> <c:set value="Kowalski" target="{uzytkownik}" property="nazisko" />
</c:catch> ${wyjatek}
```

Znaczniki bazowe (rdzenia)

• c:if – pozwala na wykonanie fragmentu kodu, o ile wartośd atrybutu test zwraca true (opcjonalnie można zapisad wynik wyrażenia logicznego, korzystając z atrybutów var i scope:

<c:set value=*"12345" var="identyfikator" scope="request" />* <c:if test=*"${identyfikator* ***eq '12345'}">*** Przyznano dostęp </c:if>

Znaczniki bazowe (rdzenia)

• c:choose – pozwala na wykonanie jednego z wielu fragmentów kodu c:when lub gdy nie pasuje żaden, wykonanie klauzuli c:otherwise

<c:choose>

<c:when test="${wiek **gt 18**}"> Jesteś dorosły </c:when>

<c:when test="${wiek **lt 18**}"> Jesteś dzieckiem </c:when>

c:otherwise> Masz 18 lat, czyli też jesteś dorosły.

</c:otherwise> </c:choose>

Znaczniki bazowe (rdzenia)

- c:forEach – pozwala na iterowanie po danej kolekcji przy użyciu danego fragmentu kodu, udostępniając przy każdej iteracji informacje o aktualnie przetworzonym obiekcie
- Może byd stosowany do kolekcji Collection, Map, a także do iteratorów (Iterator) i typów wyliczeniowych (Enumeration)

```
<c:forEach var="identyfikator" items="${listaIdentyfikatorow}">
${identyfikator}
</c:forEach>
```

Znaczniki bazowe (rdzenia)

- c:url – pozwala na wygenerowanie adresu URL i umieszczenie go w wybranym atrybucie:

```
<c:url var="hiperlacze" value="http://serwer/wyszukiwarka.jsp">
<c:param name="q" value="${zapytanie}" />
</c:url>

Adres URL: ${hiperlacze}
```

Znaczniki bazowe (rdzenia)

- c:import – wstawia treśc podanego zasobu (tak jak jsp:include), umożliwiając przy tym pobranie zasobu z zewnętrznego serwera i zapisanie zawartości zasobu w formie atrybutu

<c:import url=*"http://www.google.pl/" var="tresc" scope="request" />*

Znaczniki funkcyjne

- Znaczniki te są wywoływane jak funkcje (metody) języka Java
- Poza znacznikiem fn:length, wszystkie inne dotyczą manipulacji łańcuchami znaków
- Znacznik fn:length zwraca liczbę elementów kolekcji. Kolekcją może być każdy obiekt akceptowany przez znacznik c:forEach, a dodatkowo można też podać łańcuch znaków

- Pozostałe znaczniki dotyczą łańcuchów znaków:

fn:contains(string, substring) – zwraca true, jeśli łańcuch substring występuje w łańcuchu string

fn:endsWith(string, sufiks) – zwraca true, jeśli łańcuch string kończy się łańcuchem sufiks

fn:escapeXml(string) – zwraca łańcuch znaków, w którym niektóre znaki zostały zamienione na swoje encje XML (efekt jak przy użyciu znacznika c:out)

Znaczniki funkcyjne

- Pozostałe znaczniki dotyczą łańcuchów znaków:

fn:indexOf(string, substring) – zwraca indeks, od którego łańcuch substring występuje w łańcuchu string

fn:join(tablica, separator) – zwraca łańcuch znaków złożony z elementów tablicy, przedzielanych znakami separatora

fn:replace(string, stary, nowy) – zamienia wystąpienia łańcucha stary na łańcuch nowy w łańcuchu string

fn:split(string, separator) – zwraca tablicę łańcuchów, otrzymaną w wyniku podziału łańcucha string na elementy zgodnie z podanym separatorem

Znaczniki funkcyjne

- Pozostałe znaczniki dotyczą łańcuchów znaków:

fn:substring(string, początek, koniec) – zwraca fragment łańcucha string, począwszy od elementu o indeksie początek aż do elementu o indeksie koniec (bez tego elementu)

fn:startsWith(string, prefiks) – zwraca true, jeśli łańcuch string rozpoczyna się od łańcucha prefiks

fn:toLowerCase(string)/fn:toUpperCase(string) – zwraca łańcuch string zawierający wyłącznie małe/wielkie litery

fn:trim(string) – usuwa białe znaki z początku i końca łańcucha string (zwraca kopię bez białych znaków

Znaczniki formatujące

- Znaczniki te są oznaczane prefiksem fmt i służą do przekształcania danych (np. liczb) do łańcuchów znaków, a także do czynności odwrotnych
- Pozwalają one także na łatwą lokalizację aplikacji (przystosowanie do danego języka/kraju), czyli na internacjonalizację
- fmt:formatNumber – pozwala na sformatowanie liczby, zgodnie z określonym typem (type – currency, number, percent), a także ustawieniami separatorów dziesiętnych i tysięcy:

<fmt:formatNumber type=*"currency" value="123.1" />*

Znaczniki formatujące

• fmt:parseNumber – parsuje liczbę określoną za pomocą atrybutu value, z możliwością utrwalenia w atrybucie o wybranym zasięgu

<fmt:parseNumber type=*"number" value=",3" var="kwota" scope="session" />*

• fmt:formatDate, parseDate – umożliwia na wyświetlanie sformatowanej daty lub jej przeparsowanie

• Można skorzystać z jednego z domyślnych stylów daty i czasu (dateStyle/timeStyle) lub zdefiniować własny przy użyciu atrybutu pattern

<fmt:formatDate dateStyle=*"long" value="${data}" />*