PaaS – technologie i standardy

Paulina Adamska

tiia@pjwstk.edu.pl

Plan przedmiotu

- Technologie webowe
- Standardy i protokoły wykorzystywane w aplikacjach webowych
- Wprowadzenie do tematyki chmur
  - Przegląd platform cloudowych (PaaS)
- Google App Engine
- Symulacje chmur
- Środowiska do symulacji ogólnie

Co się przyda?

- Programowanie
  - Java

  i środowisko Eclipse…
- Podstawowa wiedza na temat technologii webowych

  Przydatna w projektach ale nie niezbędna

Literatura

- "Cloud Computing Bible", Barrie Sosinsky, Wiley 2010
- „Beginning Java Google App Engine”, Kyle Roche, Jeffrey J. Douglas, Springer 2010
- „Google App Engine Java and GWT Application Development”, Daniel Guermeur, Amy Unruh, Packt Publishing 2009
- Prezentacje z Google I/O
- https://developers.google.com/appengine/

Zasady zaliczenia ćwiczeń

- Projekt
- Prezentacja dotycząca wybranego zagadnienia związanego z chmurami lub zadanie programistyczne
- Wszystkie projekty są jednoosobowe
- Ocena z ćwiczeń przeliczana jest na punkty, które doliczają się do oceny z wykładu

Punkty za ćwiczenia

Ocena	Punkty doliczane do wykładu
bdb	30
db+	25
db	20
dst+	15
dst	10

Projekty: Prezentacje

- Prezentacje na tematy związane z chmurami:
  - Technologie/frameworki webowe
  - Platformy cloudowe PaaS (różne prezentacje mogą dotyczyć tej samej platformy, ale różnych jej aspektów)
  - Prywatność w chmurze
  - Cloud testing
  - Cloud application benchmarking
  - ...

Prezentacje: Zasady

- Prezentacje należy przesłać na adres tiia@pjwstk.edu.pl najpóźniej na tydzień przed terminem prezentacji
- Prezentacja powinna zawierać kompletną bibliografię z listą źródeł do wglądu
- Wyniki pracy należy zaprezentować na ostatnich zajęciach (bez tego brak możliwości uzyskania oceny bdb)

Prezentacje: Zasady

- Prezentacje dotyczące platform PaaS mogą mieć formę tutoriali
- Prezentacje powinny być
  - Solidne (~30 minutowe)
  - Ciekawe

  nie powinny być zwykłym tłumaczeniem jednej publikacji lub tekstu z wikipedii

Prezentacje: Zasady

- Szczegółowy temat prezentacji należy wcześniej zgłosić (bezpośrednio lub drogą mailową)
- W przypadku zainteresowania tematem większej grupy osób decyduje kolejność zgłoszeń
- Istnieje możliwość wyboru jednego tematu z listy przez większą liczbę osób pod warunkiem, że każda z nich będzie zajmowała się innym jego aspektem

Projekty: Programistyczne

- Symulacje wykorzystujące CloudSim lub środowisko Repast
- Aplikacje
  - Zastrzegam sobie tydzień na podjęcie decyzji o akceptacji pomysłu na aplikację jako projektu
- Kompletny kod źródłowy programu wraz z dokumentacją należy udostępnić najpóźniej na tydzień przed terminem prezentacji

Zasady zaliczenia wykładu

- Punkty z ćwiczeń (max 30 punktów)
- Egzamin z wykładów (max 20 punktów)

Punkty	Ocena
<40, 50)	bdb
<35, 40)	db+
<30, 35)	db
<25, 30)	dst+
<20, 25)	dst

Ważne daty

- Zajęcia w całości przeznaczone na konsultacje projektowe
  - 29.10
  - 10.12

  - Prośba o uprzedzanie o temacie projektu, którego dotyczą konsultacje

    Wcześniejszy termin szczególnie dla osób, które planują projekt dotyczący symulacji, ponieważ ten temat poruszany będzie na zajęciach dopiero pod koniec semestru

Ważne daty

- Zajęcia w całości przeznaczone na prezentacje projektów
  - 17.12
  - 21.01

- W każdym terminie może prezentować maksymalnie 5 osób
- Rezerwacje konkretnych terminów drogą mailową (lista będzie dostępna na ftp)

Plan wykładu

- Aplikacje webowe
  - Rodzaje
  - Komponenty
  - Technologie
    - Java Servlet
    - Java Server Faces
    - Facelets
  - Cykl życia aplikacji
- Serwery aplikacji

Technologie webowe

WPROWADZENIE

Aplikacje webowe

- Dynamiczne rozszerzenie funkcjonalności sieci lub serwera aplikacji

- Dwa rodzaje:
  - **Presentation-oriented**
  - **Service-oriented**

Presentation-oriented

- Generowanie interaktywnych stron bazujących na HTML, XHTML, XML, itp.
- Dynamiczne generowanie treści w odpowiedzi na żądania HTTP
- Technologie/frameworki
  - Java Server Faces
  - GWT
  - Seam
  - ...

Service-oriented

- Implementują endpoint usługi
- Poprzednia grupa to często ich frontendy
- Technologie i standardy
  - JAX
  - REST

Elementy aplikacji

- Komponenty aplikacji (web components)

  O nich bardziej szczegółowo za chwilę…

- Statyczne pliki z zasobami (np. obrazki)

- Klasy pomocnicze, zewnętrzne biblioteki

Komponenty aplikacji

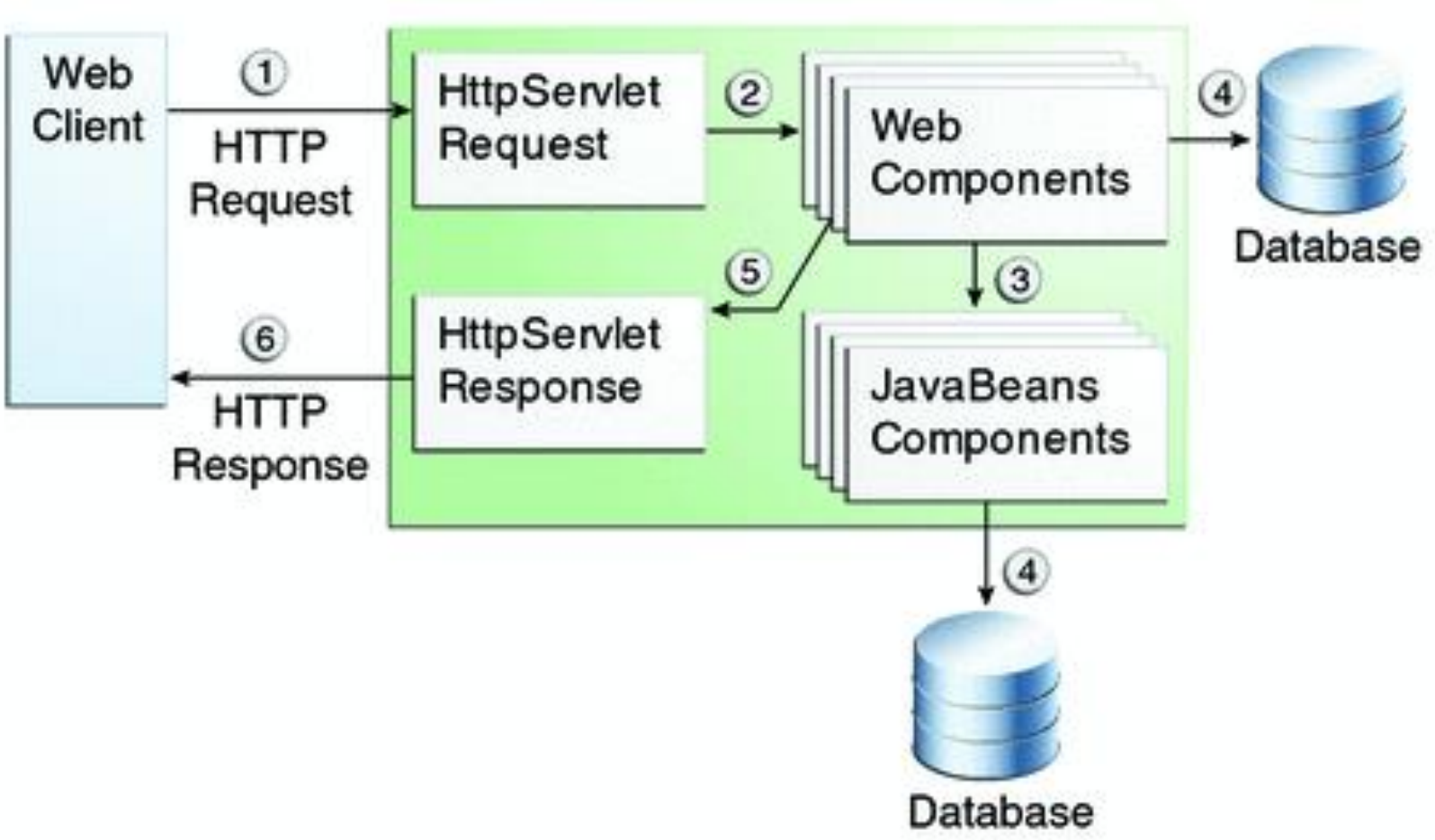
- Rodzaje
  - Strony implementowane za pomocą Java Server Faces (Facelets)
  - Endpointy usług webowych

    np. serwlety
  - Strony Java Server Pages (obecnie oznaczone jako przestarzałe)
- Działają w środowisku zapewnianym przez *web container*

Web Container

- Zapewnia
  - Przekazywanie żądań
  - Bezpieczeństwo
  - Współbieżność
  - Zarządzanie cyklem życia aplikacji
  - Dostęp do dodatkowych API (np. transakcje)

Komponenty aplikacji



Komponenty aplikacji

- Obszary zastosowań poszczególnych rodzajów komponentów są podobne
- Każdy charakteryzuje się pewnymi właściwościami
  - Serwlety
    - Dedykowane klasy przystosowane do przetwarzania żądań i generowania odpowiedzi na nie
    - Implementacja endpointów usług
  - Java Server Faces
    - Wykorzystywane raczej do prezentacji danych

Konfiguracja aplikacji

- Niektóre mechanizmy związane z działaniem aplikacji mogą zostać skonfigurowane dopiero po instalacji
- Metody konfiguracji
  - Anotacje (np. Java EE)
  - Deployment descriptor
    - Plik tekstowy z konfiguracją w formacie XML

Cykl życia aplikacji

- Tworzenie
  - Implementacja komponentów aplikacji
  - Stworzenie deskryptora (deployment descriptor)
  - Kompilacja
- Wdrożenie
  - Pakowanie (opcjonalnie)
  - Załadowanie na serwer aplikacji (web container)
- Uruchomienie
  - Wysłanie żądania HTTP na odpowiedni adres URL

Aplikacje webowe

ETAPY CYKLU ŻYCIA

Aplikacje webowe

TWORZENIE APLIKACJI

Komponenty aplikacji

- Java Servlet
- Java Server Faces
- Facelets

Tworzenie aplikacji

SERWLETY

Java Servlet

- Klasa rozszerzająca funkcjonalność serwera
  - Może rezydować na dowolnym serwerze
  - Często używana do implementowania logiki biznesowej na serwerach WWW

    Alternatywa dla CGI (Common Gateway Interfaces)
- Musi implementować interfejs `Servlet`
  - Dowolny serwlet dziedziczy po: `GenericServlet`

    `javax.servlet`
  - Serwlet obsługujące żądania HTTP: `HttpServlet`

    `javax.servlet.http`

Klasa HttpServlet

```
import javax.servlet.http.HttpServlet;

MoodServlet extends HttpServlet {

  ...

}
```

- Implementuje metody niezbędne do obsługi żądań HTTP
  - doGet
  - doPost
  - ...

Cykl życia

- Kontrolowany przez środowisko, na którym został wdrożony
- Po nadejściu żądania przypisanego do określonego serwletu serwer
  1. Jeśli nie istnieje jeszcze instacja serwletu
     a) Ładuje odpowiednią klasę
     b) Tworzy jej instancję
     c) Inicjalizuje instancję wywołując odpowiednią metodę
  2. Wywołuje odpowiednią metodę i przekazuje do niej żądanie
- Niepotrzebne instancje zostają usunięte

Listenery zdarzeń

Object	Event	Listener Interface and Event Class
Web context	Initialization and destruction	`javax.servlet.ServletContextListener` and `ServletContextEvent`
Web context	Attribute added, removed, or replaced	`javax.servlet.ServletContextAttributeListener` and `ServletContextAttributeEvent`
Session	Creation, invalidation, activation, passivation, and timeout	`javax.servlet.http.HttpSessionListener`, `javax.servlet.http.HttpSessionActivationListener`, and `HttpSessionEvent`
Session	Attribute added, removed, or replaced	`javax.servlet.http.HttpSessionAttributeListener` and `HttpSessionBindingEvent`
Request	A servlet request has started being processed by web components	`javax.servlet.ServletRequestListener` and `ServletRequestEvent`
Request	Attribute added, removed, or replaced	`javax.servlet.ServletRequestAttributeListener` and `ServletRequestAttributeEvent`

Obsługa zdarzeń

- Interfejsy

```
javax.servlet.ServletContextListener
javax.servlet.ServletContextAttributeListener
javax.servlet.ServletRequestListener
javax.servlet.ServletRequestAttributeListener
javax.servlet.http.HttpSessionListener
javax.servlet.http.HttpSessionAttributeListener
```

- Definiowanie obiektu do nasłuchiwania zdarzeń

```
public class SimpleServletListener implements
    ServletContextListener,

    ServletContextAttributeListener {

    ...

}
```

Współdzielenie informacji

Do współpracy między komponentami konieczna wymiana informacji

- Wykorzystanie pomocniczych obiektów (np. Java Beans)
- Atrybuty zadeklarowane jako publiczne
- Baza danych
- Odwołania do zewnętrznych zasobów

Scope objects

- Informacje przechowywane w atrybutach obiektów
  - getAttribute
  - setAttribute

Scope Object	Klasa
Web context	javax.servlet.ServletContext
Session	javax.servlet.http.HttpSession
Request	javax.servlet.ServletRequest
Page	javax.servlet.jsp.JspContext

Inicjalizacja

- Metoda `init`
- Dostępna domyślna implementacja
- Możliwe jej nadpisanie w celu
  - Wczytania zapisanych parametrów (konfiguracja)
  - Inicjalizacja niezbędnych zasobów
  - Dowolna inna czynność, wykonywana jednorazowo przed rozpoczęciem pracy serwletu

Jeśli proces inicjalizacji się nie powiedzie, metoda powinna zakończyć się wywołaniem wyjątku UnavailableException

Metody realizujące usługi

- do*Method*

  np. do*Get*, do*Delete*, do*Options*, do*Post*, do*Put*, do*Trace*

- Możliwe definiowanie dowolnych metod
- Procedura generowania odpowiedzi
  1. Pobranie strumienia wyjściowego odpowiedzi
  2. Ustawienie nagłówków
  3. Uzupełnienie sekcji danych

Pobieranie informacji z żądania

- Wszystkie żądania implementują interfejs `ServletRequest`
- Zapewniają dostęp do
  - Parametrów przekazywanych przez aplikacje klienckie do serwletów
  - Obiekty przekazywane pomiędzy serwletami/serwletami i kontenerem
  - Informacje o protokole komunikacyjnym
  - Informacje dotyczące lokalizacji
  - Strumienie danych

Pobieranie informacji z żądania

- Obiekt `HttpServletRequest`
  - URL

    http://[*host*]:[*port*][*request-path*]?[*query-string*]

    - Request-path
      - Context path
      - Servlet path
      - Path info
  - Nagłówki HTTP
  - Zawartość sekcji danych

Tworzenie odpowiedzi

- Wszystkie odpowiedzi implementują interfejs `ServletResponse`
- Informacje
  - Strumień wyjściowy
  - Typ danych
  - Dotyczące lokalizacji
- HTTP
  - Kod statusu
  - Ciasteczka

Filtrowanie żądań i odpowiedzi

- Filtry – obiekty potrafiące modyfikować nagłówki i dane żądania lub odpowiedzi
  - Zwykle same nie tworzą odpowiedzi
    - *Mogą być dodatkiem do innych komponentów*
  - Zadania realizowane przez filtry
    - Przeglądanie żądania i działanie w zależności od niego
    - Zapobieganie dalszemu przekazywaniu pary żądanie/odpowiedź
    - Modyfikacja nagłówków i treści żądania
    - Modyfikacja nagłówków i danych odpowiedzi
    - Interakcja z zewnętrznymi zasobami

Filtrowanie żądań i odpowiedzi

- Zastosowanie filtrów
  - Autentykacja

    Blokowanie żądań na podstawie tożsamości użytkownika
  - Logowanie
  - Konwersja obrazów

    Np. skalowanie map itp.
  - Kompresja danych
  - Lokalizacja
  - …
- Komponent może być filtrowany przez łańcuch kilku filtrów w określonej kolejności

Filtry API

`javax.servlet`

`Filter, FilterChain,FilterConfig`

- Filtr musi implementować interfejs `Filter`
- Może definiować wzorce URL

  Na ich podstawie komunikat może zostać przekazany do filtra

Definiowanie filtrów

```
import javax.servlet.Filter;

public final class HitCounterFilter implements Filter {
  private FilterConfig filterConfig = null;

  public void init(FilterConfig filterConfig) throws
  ServletException {…}

  public void destroy() {…}

  public void doFilter(ServletRequest request,
  ServletResponse response, FilterChain chain) throws
  IOException, ServletException {

     …

     chain.doFilter(request, wrapper);

  }

}
```

Filtrowanie

- Najistotniejsza metoda to `doFilter`
  - Parametry: żądanie, odpowiedź, łańcuch filtrów
  - Umożliwia
    - Analizę nagłówków żądania
    - Dostosowywanie żądania (modyfikacja nagłówków lub danych)
    - Dostosowanie odpowiedzi (modyfikacja nagłówków lub danych)
    - Wywołanie kolejnego filtra w łańcuchu (lub zablokowanie go)
    - Analiza nagłówków odpowiedzi po zastosowaniu kolejnego filtra
    - Wygenerowanie błędu podczas przetwarzania

Filtrowanie

- Konieczna implementacja dodatkowych metod
  - init
    - Wywoływana podczas tworzenia instancji filtra
    - Parametry: obiekt `FilterConfig`

      Z niego można pobrać wszelkie ustawienia filtra
  - destroy

Modyfikowanie żądań i odpowiedzi

- Metody
  - Dodawanie atrybutów do żądania
  - Dodawanie danych do odpowiedzi
- Stand-in stream
  - Zapobiega zamknięciu oryginalnego strumienia odpowiedzi przed wprowadzeniem zmian przez filtry
  - Wrappery
    - Żądania
    - Odpowiedzi

Wrappery

- Nadpisują metody (dla odpowiedzi)
  - `getWriter`
  - `getOutputStream`
- Przekazywane do metody doFilter
- Wrappery żądania
  - `ServletRequestWrapper`
  - `HttpServletRequestWrapper`
- Wrappery odpowiedzi
  - `ServletResponseWrapper`
  - `HttpServletResponseWrapper`

Przykłady: HitCounter

```
public void doFilter(ServletRequest request,
   ServletResponse response, FilterChain chain) throws
   IOException, ServletException {

   if (filterConfig == null) return;


   Counter counter = (Counter)filterConfig.
   getServletContext(). getAttribute("hitCounter");

   //tu tworzony jest wrapper

   chain.doFilter(request, wrapper);

   ...
}
```

Przykłady: HitCounter

```
//Definicja na kolejnym slajdzie

CharResponseWrapper wrapper = new CharResponseWrapper(
   (HttpServletResponse)response);

chain.doFilter(request, wrapper);

if(wrapper.getContentType().equals("text/html")) {
   CharArrayWriter caw = new CharArrayWriter();
   caw.write(wrapper.toString().substring(0,
   wrapper.toString().indexOf("</body>")-1));
   caw.write("<p>\nYou are visitor number <font
   color='red'>" + counter.getCounter() + "</font>");
   caw.write("\n</body></html>");
   response.setContentLength(caw.toString().length());
   out.write(caw.toString());

}
```

Przykłady: HitCounter

```
public class CharResponseWrapper extends
   HttpServletResponseWrapper {

   private CharArrayWriter output;


   public CharResponseWrapper(HttpServletResponse
      response){

      super(response);

      output = new CharArrayWriter();

   }

   public PrintWriter getWriter(){

      return new PrintWriter(output);

   }

}
```

Przykłady: Modyfikacja kodowania

```
protected String selectEncoding(ServletRequest request) {
  return (this.encoding);

}

public void doFilter(ServletRequest request,
  ServletResponse response, FilterChain chain) throws
  IOException, ServletException {

  String encoding = selectEncoding(request);

  if (encoding != null)
     request.setCharacterEncoding(encoding);

  chain.doFilter(request, response);

}
```

Przykłady: Kompresja odpowiedzi

```
public void write(int b) throws IOException {

   ...

   if ((bufferCount >= buffer.length) ||
   (count>=compressionThreshold)) {
      compressionThresholdReached = true;

   }

   if (compressionThresholdReached) {

      writeToGZip(b);

   }


   else { buffer[bufferCount++] = (byte) b; count++;

   }

}
```

Przykłady: Transformacja odpowiedzi (XSLT)

Przykładowa odpowiedź serwera:

```
<book>
  <isbn>123</isbn>
  <title>Web Servers for Fun and
  Profit</title>
  <quantity>10</quantity>
  <price>$17.95</price>
</book>
```

Przykłady: Transformacja odpowiedzi

Szablon formatujący odpowiedź:

…

```
<xsl:element name="book">

<xsl:attribute name="isbn"><xsl:value-of select="isbn"/>

</ xsl:attribute>

<xsl:element name="quantity"><xsl:value-of
  select="quantity"/>

</xsl:element>
```

…

Przykłady: Transformacja odpowiedzi

```
public void doFilter(ServletRequest request, ServletResponse
  response, FilterChain chain) throws IOException,
  ServletException {

  String contentType;

  String styleSheet;

  String type = request.getParameter("type");

  if (type == null || type.equals("")) {

     contentType = "text/html";

     styleSheet = "/xml/html.xsl";

  } else {…}

  response.setContentType(contentType);

  …
```

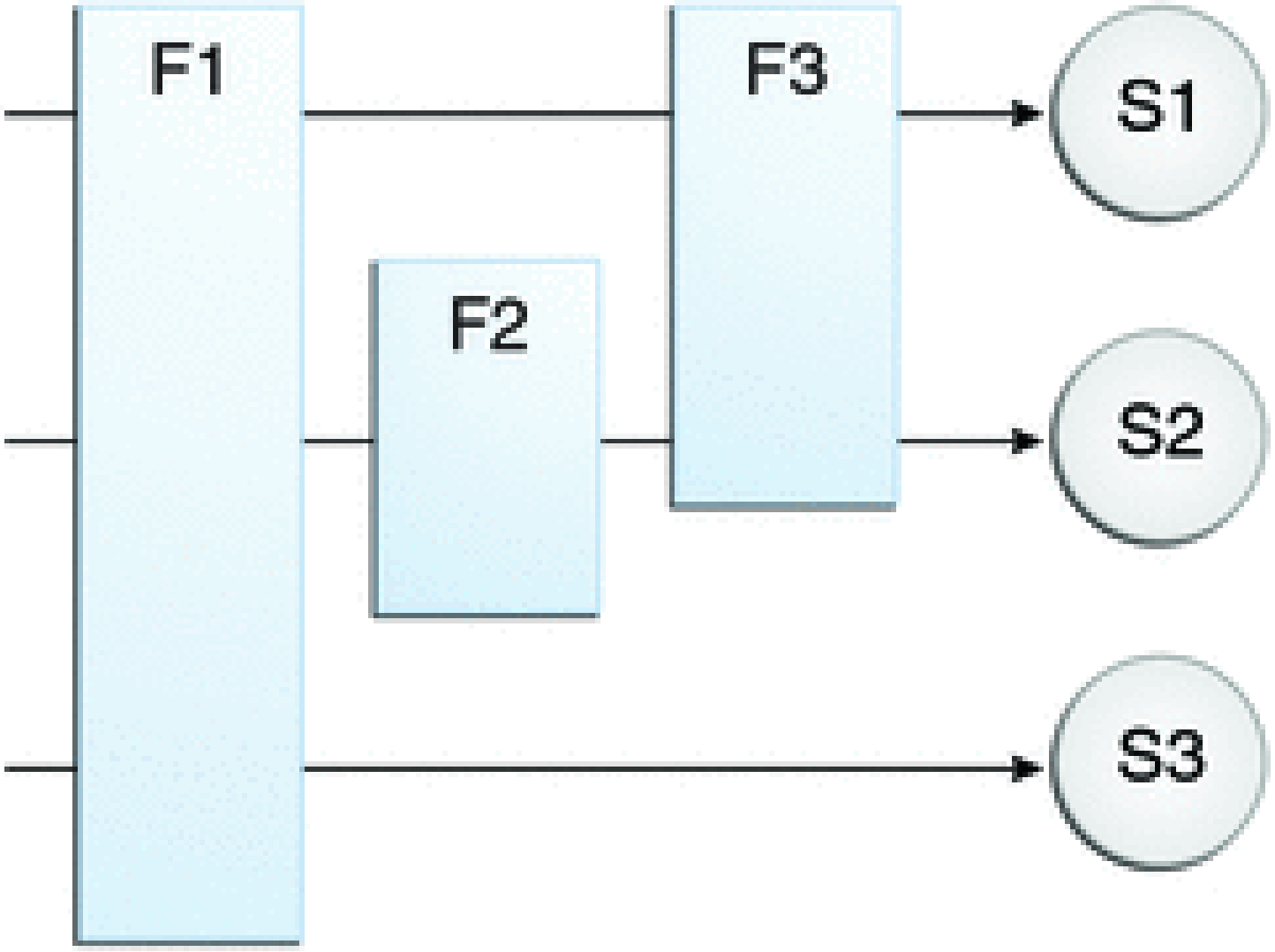
Przykłady: Transformacja odpowiedzi

```
CharResponseWrapper responseWrapper = new
  CharResponseWrapper( (HttpServletResponse)response);

chain.doFilter(request, wrapper);

// Get response from servlet

StringReader sr = new StringReader( new
  String(wrapper.getData()));

Source xmlSource = new StreamSource((Reader)sr);

try {…

CharArrayWriter caw = new CharArrayWriter();

StreamResult result = new StreamResult(caw);

transformer.transform(xmlSource, result);

response.setContentLength(caw.toString().length());

} catch(Exception ex) {…}}
```

Mapowanie filtrów

- Web container wykorzystuje mapowania do powiązania filtrów z odpowiednimi komponentami (np. serwletami)
- Filtry wywoływane w kolejności, w jakiej zostały zdefiniowane mapowania
- Defniowane w deskryptorze

Mapowanie filtrów



Definiowanie filtrów aplikacji

- W deskryptorze (plik web.xml)
  - Definicja filtra

    (jakie filtry są dostępne dla aplikacji)

    Element <filter>
  - Definicja mapowania

    (z jakich filtrów korzystają konkretne komponenty, np. serwlety)

    Element <filter-mapping>

Definiowanie filtrów aplikacji

```
<filter>
<filter-name>Compression Filter</filter-name>
<filter-class>CompressionFilter</filter-class>
<init-param>
  <param-name>compressionThreshold</param-name>
  <param-value>10</param-value>
</init-param>
</filter>
```

Definiowanie filtrów aplikacji

```
<filter-mapping>

  <filter-name>Compression Filter</filter-name> <servlet-
  name>CompressionTest</servlet-name>

</filter-mapping>

<servlet>

  <servlet-name>CompressionTest</servlet-name> <servlet-
  class>CompressionTest</servlet-class>

</servlet>

<servlet-mapping>

  <servlet-name>CompressionTest</servlet-name> <url-
  pattern>/CompressionTest</url-pattern>

</servlet-mapping>
```

Odwołania do innych zasobów sieciowych

- Pośrednio
  - Umieszczanie adresu URL zasobu
- Bezpośrednio
  - Umieszczanie samego zasobu
- RequestDispatcher
  - Umożliwia uzyskanie dostępu do zasobu na tym samym serwerze aplikacji co bieżący komponent
  - Metoda `getRequestDispatcher("URL")`
    - Pobierany z żądania - można podać adres względny
    - Z kontekstu (ang. web context) – konieczny pełny adres
  - Jeśli zasób nie jest dostępny – zwracany null

Dołączanie zasobów do odpowiedzi

- Metoda

  RequestDispatcher. include(request, response)

- Zasoby statyczne
  - Zwykły „programistyczny” include
- Komponent webowy
  1. Wysłanie żądania do wskazanego komponentu
  2. Dołączenie rezultatów do odpowiedzi

  Dołączany obiekt ma (ograniczony) dostęp do żądania
  - *Nie może modyfikować i wysyłać odpowiedzi do klienta*
  - *Nie może wywoływać metod modyfikujących nagłówki odpowiedzi (np. setCookie)*

Przekazanie sterowania do innego komponentu

- *Serwlet przeprowadza wstępne przetwarzanie, a inny komponent zwraca odpowiedź*
- *Metoda*

  RequestDispatcher.forward

  - *Oryginalne URI zapisywane jako atrybut żądania*

    javax.servlet.forward.[*request-uri*|*context-path*|*servlet-path*|*path-info*|*query-string*]
  - Jej użycie wyklucza wcześniejszy dostęp do obiektów ServletOutputStream lub PrintWriter

    Oznacza całkowite przekazanie odpowiedzialności za wysłanie odpowiedzi do innego komponentu

Dostęp do kontekstu

(ang. web context)

- Obiekt implementujący interfejs `ServletContext`
- Zapewnia dostęp do
  - Parametrów inicjalizacyjnych
  - Zasobów powiązanych z kontekstem
  - Atrybutów
  - Możliwości logowania
- `Metoda` getServletContext

Przechowywanie stanu klienta

- API do zarządzania sesjami
  - Obiekt `HttpSession`
  - Dostęp do sesji powiązanej z żądaniem
    - Metoda `getSession`
  - Można powiązać atrybut z żądaniem
  - Możliwe nasłuchiwanie zdarzeń związanych z sesją
    - `javax.servlet.http.HttpSessionBindingListener`
    - `javax.servlet.http.HttpSessionActivationListener`

Zarządzanie sesją

- Sesje mają przypisany timeout
  - `getMaxInactiveInterval`
  - `setMaxInactiveInterval`
- Po tym czasie zasoby związane z sesją są zwalniane
- Zapewnienie, że aktywna sesja nie wygaśnie
  - Okresowe uzyskiwanie dostępu do sesji
- Zamknięcie sesji
  - Metoda `invalidate`

Śledzenie sesji

- Kilka metod powiązania identyfikatora klienta z sesją
  - Ciasteczko przechowywane po stronie klienta
  - Identyfikator dołączany do każdego URL'a
- Jeśli aplikacja wykorzystuje sesje konieczne modyfikowanie URL'a jeśli ciasteczka u klienta są wyłączone
  - Metoda encodeURL(URL)
    - W obiekcie reprezentującym odpowiedź
    - Dodaje id sesji do URL, jeśli wyłączona obsługa ciasteczek

Kończenie pracy serwletu

- Serwer może zadecydować o zakończeniu pracy serwletu
  - np. odzyskanie pamięci
- Wywoływana metoda `destroy`
  - Zwalnianie zasobów, zapisanie stanu
- Serwer stara się wywoływać metodę dopiero kiedy
  1. Obsługa wszystkich żądań zostanie zakończona
  2. Minie grace period dla wszystkich niezakończonych usług

Monitorowanie liczby obsługiwanych żądań

```
public class ShutdownExample extends HttpServlet {

  private int serviceCounter = 0;

  ...

  // Access methods for serviceCounter

  protected synchronized void enteringServiceMethod() {
      serviceCounter++;

  }

  protected synchronized void leavingServiceMethod() {
      serviceCounter--;

  }

  protected synchronized int numServices() { return
      serviceCounter;

  } }
```

Monitorowanie liczby żądań

```
protected void service(HttpServletRequest req,
   HttpServletResponse resp) throws
   ServletException,IOException {


   enteringServiceMethod();


   try {
      super.service(req, resp);
   } finally {
      leavingServiceMethod();
   }
}
```

Informowanie o konieczności zakończenia pracy

```
public class ShutdownExample extends HttpServlet {

  private boolean shuttingDown;

  ...

  //Access methods for shuttingDown

  protected synchronized void setShuttingDown(boolean
  flag) {

     shuttingDown = flag;

  }

  protected synchronized boolean isShuttingDown() {

     return shuttingDown;

  }

}
```

Informowanie o konieczności zakończenia pracy

```
public void destroy() {

   /* Check to see whether there are still service methods
   /* running, and if there are, tell them to stop. */

   if (numServices() > 0) {

      setShuttingDown(true);

   }

   /* Wait for the service methods to stop. */
   while(numServices() > 0) {

      try {

            Thread.sleep(interval);

      } catch (InterruptedException e) { }

   }

}
```

Informowanie o konieczności zakończenia pracy

```
public void doPost(...) {

   ...

   for(i = 0; ((i < lotsOfStuffToDo) && !isShuttingDown());
   i++) {

      try {

            partOfLongRunningOperation(i);

      } catch (InterruptedException e) { ... }

   }

}
```

Obsługa błędów

- Domyślnie w wyniku błędu serwer zwraca stronę z informacją, że przetwarzanie żądania zakończyło się błędem
- Możliwe zdefiniowanie konkretnych odpowiedzi, związanych z konkretnymi błędami

Tworzenie aplikacji

JAVA SERVER FACES

Wprowadzenie

- Składa się z następujących komponentów
  - API
    - Reprezentacja komponentów i zarządzanie ich stanem
    - Obsługa zdarzeń
    - Konwersja danych
    - Nawigacja
    - Internacjonalizacja
    - …
  - Biblioteki tagów
    - Dodawanie komponentów do strony
    - Łączenie komponentów z obiektami po stronie serwera

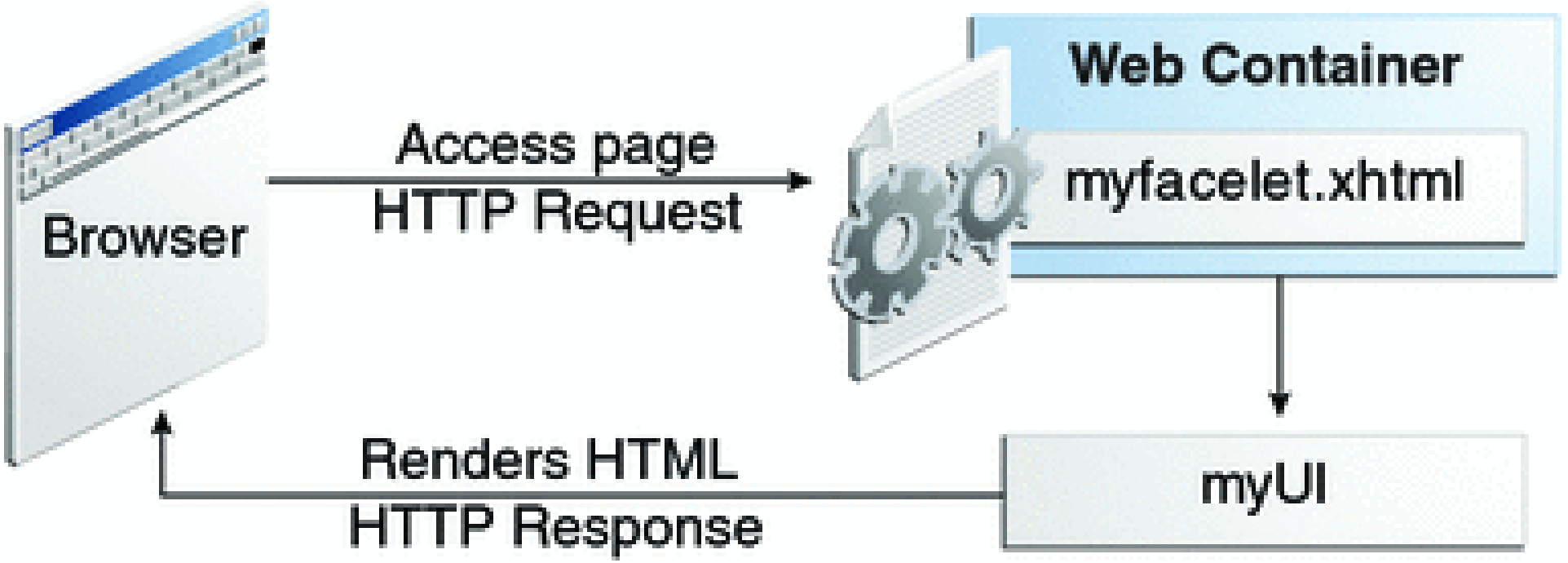
Zastosowania

- Tworzenie stron
- Dodawanie elementów do strony (tagi)
- Dodawanie powiązań pomiędzy elementami strony a danymi po stronie serwera
- Powiązanie zdarzeń generowanych przez komponenty z kodem aplikacji po stronie serwera
- Przechowywanie/odtwarzanie stanu aplikacji
- Ponowne wykorzystanie elementów

Elementy aplikacji

- Zbiór stron
- Zbiór tagów umożliwiających dodawanie komponentów do stron
- Dedykowane obiekty do przechowywania danych
- Deployment descriptor
- inne pliki konfiguracyjne (opcjonalnie)

Interakcja klienta z serwerem

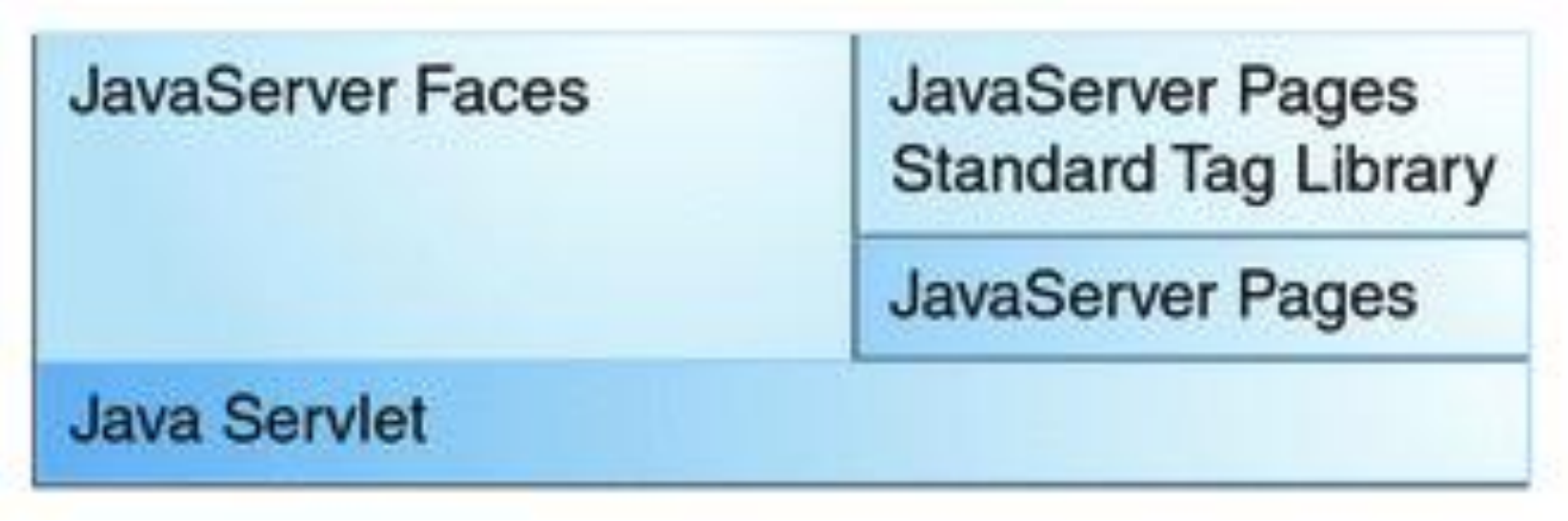


Zalety technologii

- Oddzielenie logiki aplikacji od prezentacji
  - Osoba projektująca stronę może wykorzystać tagi JSF do powiązania zdarzeń z kodem po stronie serwera

    Bez konieczności pisania skryptów
- Mapowanie żądań HTTP na obsługę konkretnego zdarzenia

Zalety technologii



Facelets

- Technologia będąca częścią specyfikacji JavaServer Faces 2.0
- Technologia tworzenia widoku dla Java Server Faces
  - Następca Java Server Pages
- Wykorzystuje XHTML do generowania stron
- Wsparcie dla Expression Language (EL)

Facelets

- Zalety
  - Ponowne wykorzystanie kodu (szablony)
  - Anotacje ułatwiające zarządzanie pewnymi obiektami
  - Uproszczona konfiguracja aplikacji
  - Zarządzanie stanem, przetwarzanie danych, weryfikacja danych wprowadzonych przez użytkownika, obsługa zdarzeń

Tagi

Tag Library	URI	Prefix	Example	Contents
JavaServer Faces Facelets Tag Library	`http://java.sun.com/jsf/facelets`	`ui:`	`ui:component` `ui:insert`	Tags for templating

JavaServer Faces HTML Tag Library	`http://java.sun.com/jsf/html`	`h:`	`h:head`
			`h:body`
			`h:outputText`
			`h:inputText`
				JavaServer Faces component tags for all `UIComponent` objects

JavaServer Faces Core Tag Library	`http://java.sun.com/jsf/core`	`f:`	`f:actionListener` `f:attribute`	Tags for JavaServer Faces custom actions that are independent of any particular render kit
JSTL Core Tag Library	`http://java.sun.com/jsp/jstl/core`	`c:`	`c:forEach` `c:catch`	JSTL 1.2 Core Tags
JSTL Functions Tag Library	`http://java.sun.com/jsp/jstl/functions`	`fn:`	`fn:toUpperCase` `fn:toLowerCase`	JSTL 1.2 Functions Tags

Tagi

Tag	Function
`ui:component`	Defines a component that is created and added to the component tree.
`ui:composition`	Defines a page composition that optionally uses a template. Content outside of this tag is ignored.
`ui:debug`	Defines a debug component that is created and added to the component tree.
`ui:decorate`	Similar to the composition tag but does not disregard content outside this tag.
`ui:define`	Defines content that is inserted into a page by a template.
`ui:fragment`	Similar to the component tag but does not disregard content outside this tag.
`ui:include`	Encapsulate and reuse content for multiple pages.
`ui:insert`	Inserts content into a template.
`ui:param`	Used to pass parameters to an included file.
`ui:repeat`	Used as an alternative for loop tags, such as `c:forEach` or `h:dataTable`.
`ui:remove`	Removes content from a page.

http://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/

Szablony: Przykład

- Przykładowa strona template.xhtml
- Trzy sekcje
  - Górna
  - Lewa
  - Główna
- Z sekcjami powiązane arkusze stylów

template.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">

    <h:head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8" />
        <h:outputStylesheet library="css" name="default.css"/>
        <h:outputStylesheet library="css" name="cssLayout.css"/>
        <title>Facelets Template</title>
    </h:head>

    <h:body>
        <div id="top" class="top">
            <ui:insert name="top">Top Section</ui:insert>
        </div>
        <div>
        <div id="left">
             <ui:insert name="left">Left Section</ui:insert>
        </div>
        <div id="content" class="left_content">
             <ui:insert name="content">Main Content</ui:insert>
        </div>
        </div>
    </h:body>
</html>
```

Wykorzystanie szablonów: Przykład

- Przykładowa strona wykorzystująca szablony templateclient.xhtml
- Zaznaczenie wykorzystania szablonu
  - Tag `ui:composition`
- Dodawanie elementów do szablonu
  - Tag `ui:define`

templateclient.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">

    <h:body>
        <ui:composition template="./template.xhtml">
            <ui:define name="top">
                Welcome to Template Client Page
            </ui:define>

            <ui:define name="left">
                <h:outputLabel value="You are in the Left Section"/>
            </ui:define>

            <ui:define name="content">
                <h:graphicImage value="#{resource['images:wave.med.gif']}"/>
                <h:outputText value="You are in the Main Content Section"/>
            </ui:define>
        </ui:composition>
    </h:body>
</html>
```

Złożone komponenty

- Specjalny rodzaj szablonu wykorzystywany jako komponent
- Każdy komponent
  - Kod działający w określony sposób

    np. kontrolka do wprowadzania danych przez użytkownika
  - Może posiadać mechanizmy weryfikacji/ konwersji danych, listenery
- Złożony komponent
  - Zbiór tagów i innych komponentów
  - Może posiadać mechanizmy weryfikacji/ konwersji danych, listenery

Tagi złożonych komponentów

Tag	Function
`composite:interface`	Declares the usage contract for a composite component. The composite component can be used as a single component whose feature set is the union of the features declared in the usage contract.
`composite:implementation`	Defines the implementation of the composite component. If a `composite:interface` element appears, there must be a corresponding `composite:implementation`.
`composite:attribute`	Declares an attribute that may be given to an instance of the composite component in which this tag is declared.
`composite:insertChildren`	Any child components or template text within the composite component tag in the using page will be reparented into the composite component at the point indicated by this tag's placement within the `composite:implementation` section.
`composite:valueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ValueHolder` suitable for use as the target of attached objects in the using page.

Tagi złożonych komponentów

`composite:editableValueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `EditableValueHolder` suitable for use as the target of attached objects in the using page.
`composite:actionSource`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ActionSource2` suitable for use as the target of attached objects in the using page.

http://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/

Złożone komponenty: Przykład

- Kontrolka do pobierania adresu e-mail
- Szablon zapisany w
  - Pliku `email.xhtml`
  - Folderze `resources/emcomp`

    Ten folder będzie traktowany jako źródło bibliotek JSF

email.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:composite="http://java.sun.com/jsf/composite"
  xmlns:h="http://java.sun.com/jsf/html">

    <h:head>
        <title>This content will not be displayed</title>
    </h:head>
    <h:body>
        <composite:interface>
            <composite:attribute name="value" required="false"/>
        </composite:interface>

        <composite:implementation>
            <h:outputLabel value="Email id: "></h:outputLabel>
            <h:inputText value="#{cc.attrs.value}"></h:inputText>
        </composite:implementation>
    </h:body>
</html>
```

#{cc.attrs.attribute-name} cc w Java Server Faces zarezerwowane dla interfejsu composite components

Wykorzystanie złożonego komponentu (emuserpage.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
  transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:em="http://java.sun.com/jsf/composite/emcomp/">

  <h:head> <title>Using a sample composite
  component</title>

  </h:head>

  <body> <h:form> <em:email value="Enter your email id" />
  </h:form>

  </body>

</html>
```

Expression Language

- Był wykorzystywany także przez technologię Java Server Pages
- Połączenie pomiędzy prezentacją strony a logiką aplikacji
  - Dynamiczny dostęp do danych

np. wartość atrybutu test jest zwracana przez wyrażenie EL

```
<c:if test="${sessionScope.cart.numberOfItems > 0}">
    ...
</c:if>
```

Expression Language

- Zadania
  - Wczytywanie danych generowanych przez aplikację webową
  - Zapis danych (np. wczytanych za pośrednictwem formularzy)
  - Wywoływanie statycznych, publicznych metod
  - Wykonywanie operacji arytmetycznych

Expression Language

- Typy wyrażeń
  - Natychmiastowe
    - Sprawdzane natychmiast
  - Odroczone
    - Mogą zostać sprawdzone później
  - Odnoszące się do wartości
    - Odnosi się bezpośrednio do danych
  - Odnoszące się do metody
    - Odpowiada za wywołanie metody

Expression Langugage

- Typy wyrażeń (c.d)
  - R-wartości
    - Może jedynie odczytywać wartość
  - L-wartości
    - Może zarówno odczytywać jak i zapisywać wartości w określonych obiektach
- Mechanizm rozszerzeń do implementowania własnych wyrażeń

Wyrażenia natychmiastowe i odroczone

- Natychmiastowe

  `${...}`

  - Po wyświetleniu strony są już obliczone
- Odroczone

  `#{...}`

  - Technologia wykorzystująca EL może zdecydować, kiedy je obliczyć
  - Często wykorzystywane przez Java Server Faces

Wyrażenia natychmiastowe

np. jako atrybut value jednego z tagów

```
<fmt:formatNumber value="${sessionScope.cart.total}"/>
```

Java Server Faces

1. Pobiera wartość wyrażenia
2. Konwertuje
3. Przekazuje do atrybutu

Zawsze są to wartości tylko do odczytu!

Wyrażenia odroczone

np. ustawienie nazwy wprowadzanej przez Użytkownika

```
<h:inputText id="name" value="#{customer.name}" />
```

Mogą to być

- Wyrażenia przechowujące wartość
- Wyrażenia wywołujące metodę

Wyrażenia odwołujące się do wartości

- Dzielą się na
  - R-watości
  - L-wartości
- Mogą być
  - Natychmiastowe `${customer.name}`
    - Zawsze r-wartości
  - Odroczone `#{customer.name}`
    - Mogą być zarówno r-wartościami jak i l-wartościami

Wyrażenia odwołujące się do wartości

- Zarówno r-wartości jak i l-wartości mogą odwoływać się do
  - Komponentów JavaBean
  - Kolekcji
  - Enumeracje
  - Obiekty

  np. odwołanie do JavaBean o nazwie customer

  `${customer}`

Wyrażenia odwołujące się do wartości

- Do pobierania wartości wyrażenia kontener wykorzystuje konteksty

  `PageContext.findAttribute(String)`

  - Argument to nazwa obiektu użyta w wyrażeniu
  - Przeszukiwanie kontekstów
    - Strony
    - Żądania
    - Sesji
    - Aplikacji

Wyrażenia odwołujące się do wartości

- Możliwe zdefiniowanie własnych obiektów, które implementują inny sposób wyszukiwania obiektów
- Odwołania do enumeracji

```
public enum Suit {hearts, spades, diamonds, clubs}
```

```
${mySuit == "hearts"}
```

Wyrażenia odwołujące się do wartości

- Odwołania do właściwości obiektów i elementów kolekcji
  - Notacja z kropką

    `${customer.name}`
  - Notacja z nawiasami kwadratowymi

    `${customer["name"]}`
- Możliwe łączenie notacji

  np. `${customer.address["street"]}`

Wyrażenia odwołujące się do wartości

- Odwołania do enumeracji
  - Możliwe odwołania w taki sam sposób jak opisano wcześniej

  Enumeracja musi przestrzegać reguł Java Beans

  `${myPlanet.mass}`

  Zakłada, że klasa Planet ma getter `getMass()`

Wyrażenia odwołujące się do wartości

- Dostęp do elementów kolekcji
  - Przez bezpośrednie podanie numeru obiektu na liście

    ```
    ${customer.orders[1]}
    ```

  - Przez podanie zmiennej liczbowej

    ```
    ${customer.orders.socks}
    ```

  - Przez podanie klucza (kolekcje typu Map)

    ```
    ${customer.orders["socks"]}
    ```

Wyrażenia odwołujące się do wartości

- R-wartości mogą przyjmować też wyrażenia nie będące obiektami
  - `${"literal"}`
  - `${customer.age + 20}`
  - `${true}`
  - `${57}`

Wyrażenia odwołujące się do wartości

- Dostępne typy wartości
  - Boolean: true and false
  - Integer: jak w Javie
  - Floating-point: jak w Javie
  - String: podwójny lub pojedynczy cudzysłów
  - Null: null

Atrybut value

- Atrybut value może zostać ustawiony na kilka sposobów
  - Pojedynczym wyrażeniem

    ```
    <some:tag value="${expr}"/>
    <another:tag value="#{expr}"/>
    ```
  - Jednym lub kilkoma wyrażeniami otoczonymi tekstem

    ```
    <some:tag value="some${expr}${expr}text${expr}"/>
    <another:tag value="some#{expr}#{expr}text#{expr}"/>
    ```
  - Samym tekstem

    ```
    <some:tag value="sometext"/>
    ```

Wyrażenia odwołujące się do metod

- Pozwalają wywoływać metody na rzecz obiektów

  np. podczas obsługi zdarzeń

```
<h:form>
  <h:inputText id="name"
  value="#{customer.name}"
  validator="#{customer.validateName}"/>
  <h:commandButton id="submit"
  action="#{customer.submit}" />
</h:form>
```

Wyrażenia odwołujące się do metod

- Zawsze są odroczone

  Metody mogą być wywoływane na różnych etapach cyklu życia

- Notacje podobnie jak w przypadku wartości
  - Z kropką

    ```
    #{object.method}
    ```

  - Z nawiasami kwadratowymi

    ```
    #{object["method"]}
    ```

Wyrażenia odwołujące się do metod

- Przekazywanie parametrów
  - `expr-a[expr-b](parameters)`
  - `expr-a.identifier-b(parameters)`

expr-a – obiekt, na którego rzecz wywołujemy metodę

expr-b – nazwa metody

identifier-b – nazwa metody

parameters – parametry oddzielone przecinkami

Wyrażenia odwołujące się do metod

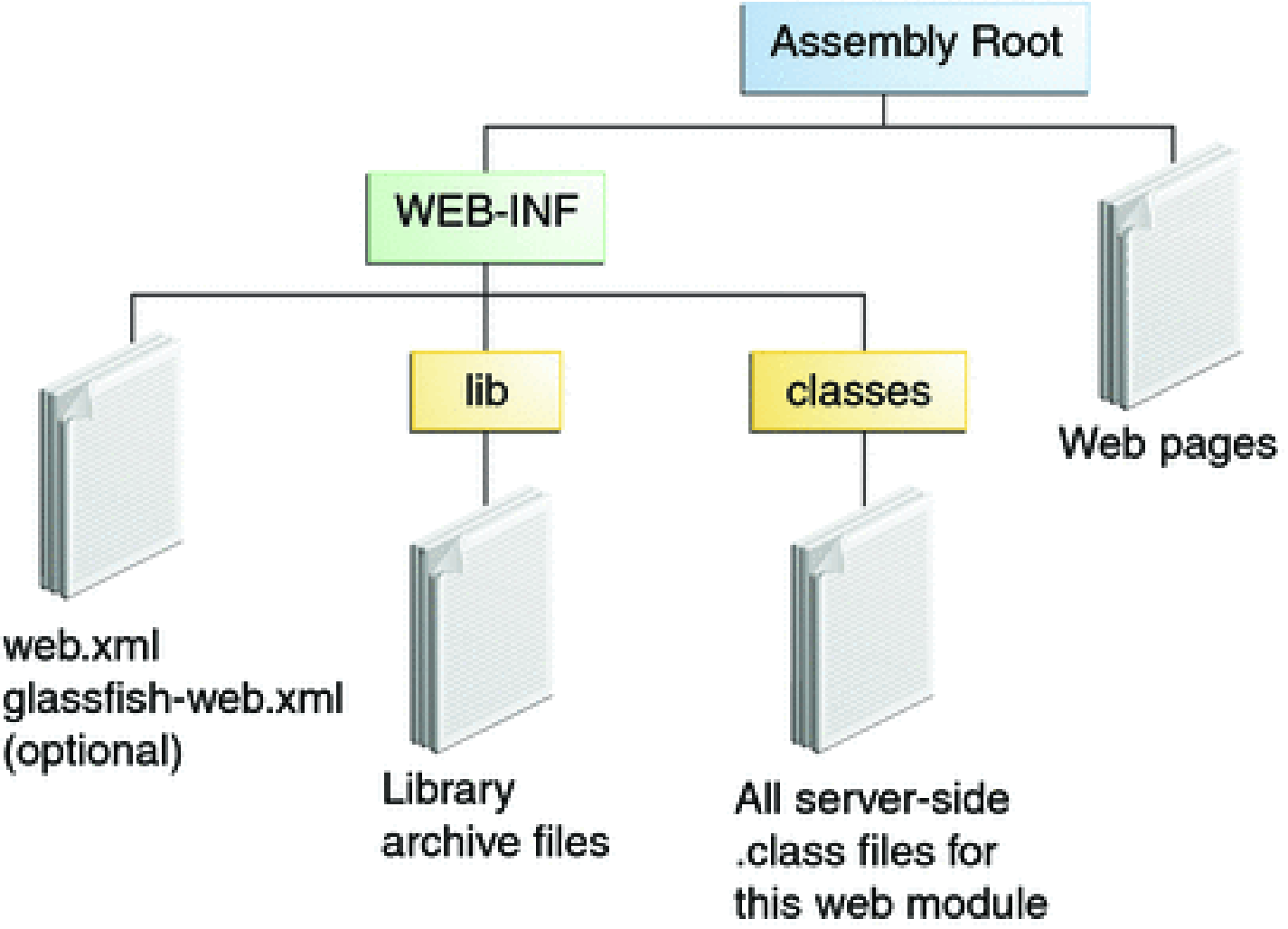
- Przykłady

```
<h:inputText value="#{userNumberBean.userNumber('5')}">


<h:commandButton action="#{trader.buy}" value="buy"/>


<h:commandButton action="#{trader.buy('SOMESTOCK')}"
value="buy"/>
```

Aplikacje webowe

WDROŻENIE APLIKACJI

Struktura aplikacji



Wdrażanie

- Aplikacja webowa może zostać wdrożona jako spakowane archiwum
  - Plik z rozszerzeniem .war
  - Tak naprawdę zwykłe archiwum zawierające wcześniej opisaną strukturę katalogów
- Aplikację można zainstalować na różnych serwerach aplikacji
  - Glass Fish (Java EE)
  - Jetty (standalone lub bazujące na nim rozszerzenia)
  - …

Wdrażanie aplikacji

SERWERY APLIKACJI

Wprowadzenie

- Dostarczają środowisko do uruchamiania aplikacji
- Java
  - Komercyjne
    - WebLogic (Oracle)
    - WebSphere (IBM)
    - …
  - Open source
    - JBoss
    - Geronimo (Apache)
    - GlassFish (Oracle)
    - Jetty
    - …

Jetty

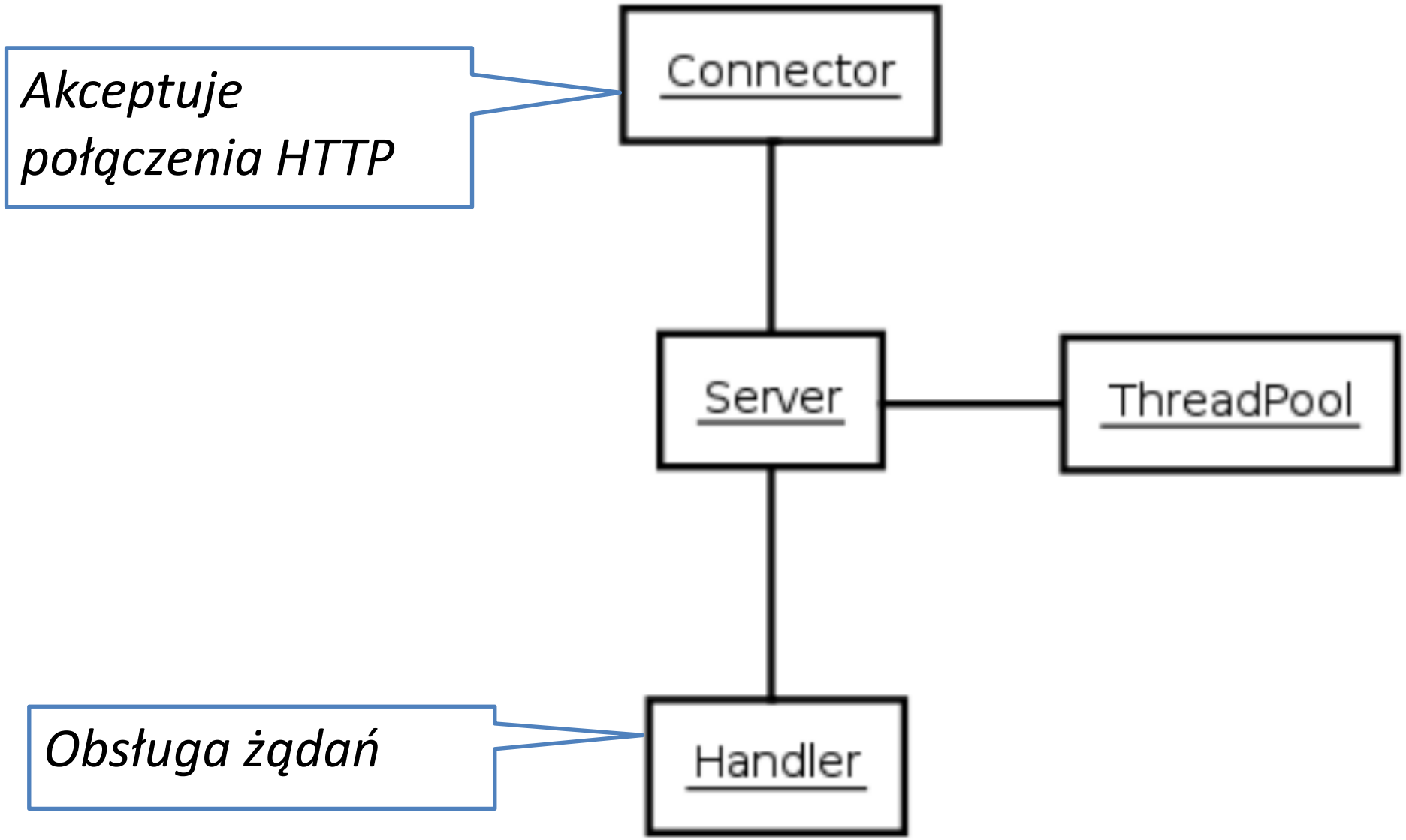
- Projekt open-source
- Dostarcza
  - Serwer HTTP
  - Klienta HTTP
  - Środowisko do uruchamiania serwletów
- Uruchamianie serwera

  java -jar start.jar

Jetty

- Domyślna konfiguracja
  - Jedna instancja JVM związana z jedną instancją serwera

Architektura



etc/jetty.xml

- Zawiera domyślną konfigurację serwera
- Mapowanie Java API na XML
- Można korzystać z wielu plików konfiguracyjnych

  java -jar start.jar etc/abcd.xml etc/xyz.xml

  Jeśli ID serwerów w plikach są różne, kilka serwerów będzie działało w ramach jednej maszyny wirtualnej

etc/jetty.xml

- Elementy
  - **Configure**

    Wybór komponentu, który będziemy konfigurować
  - **Set**

    Ustawianie atrybutu klasy
  - **New**

    Deklaracja nowego obiektu
  - **Call**

    Wywołanie metody
  - **Arg**

    Przekazanie argumentu do metody

etc/jetty.xml: Configure

```
<Configure id="Server" class="org.eclipse.jetty.server.Server">
    …
</Configure>
```

- Atrybuty
  - id

    „nazwa” obiektu wykorzystywana w celu odwoływania się do konkretnego obiektu
  - Class

    Klasa, której instancją jest dany obiekt

    (w dokumentacji tej klasy możemy szukać znaczenia poszczególnych ustawień)

etc/jetty.xml: Set

```
<Set name="ThreadPool">
    …
</Set>
```

- Atrybuty
  - name

    Nazwa konfigurowanego atrybutu

    W tym przypadku:

    http://download.eclipse.org/jetty/stable-7/apidocs/org/eclipse/jetty/util/thread/ThreadPool.html

etc/jetty.xml: New

```
<Set name="ThreadPool">
    <New class="org.eclipse.jetty.util.thread.QueuedThreadPool">
        …
    </New>
</Set>
```

- Atrybuty
  - class

    Klasa, do której należy nowy obiekt

  Wewnątrz możemy umieścić kolejne settery konfigurujące nowy obiekt

etc/jetty.xml: Call

```
<Call name="addConnector">
    …
</Call>
```

- Atrybuty
  - name

    Nazwa metody, która zostanie wywołana

Element zagnieżdżony w odpowiednim elemencie <Configuration>

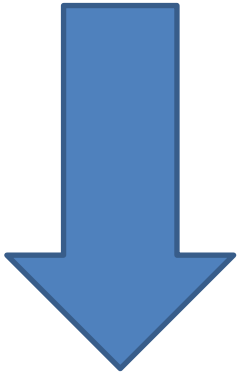
etc/jetty.xml: Arg

```
<Call name="addConnector">
    <Arg>
        …
    </Arg>
</Call>
```

- Tu podajemy parametry niezbędne do wywołania metody

etc/jetty.xml: Przykłady

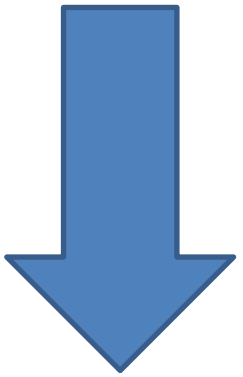
```
<Call name="test">
  <Arg>value1</Arg>
  <Set name="Test">Value2</Set>
</Call>
```



```
Object o2 = o1.test("value1");
o2.setTest("value2");
```

etc/jetty.xml: Przykłady

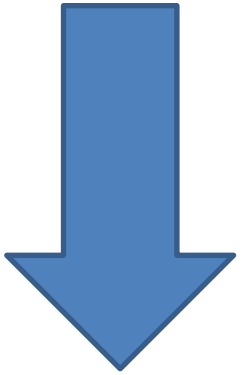
```
<New id="myobject" class="com.acme.MyClass">
  <Arg>value1</Arg><Set name="Test">Value2</Set>
</New>
```



```
Object o = new com.acme.MyClass("value1");
o.setTest("value2");
```

etc/jetty.xml: Przykłady

```
<Array type="java.lang.String">
  <Item>value0</Item>
  <Item><New class="java.lang.String"><Arg>value1</Arg></New></Item>
</Array>
```



```
String[] a = new String[]{"value0", new String("value1")};
```

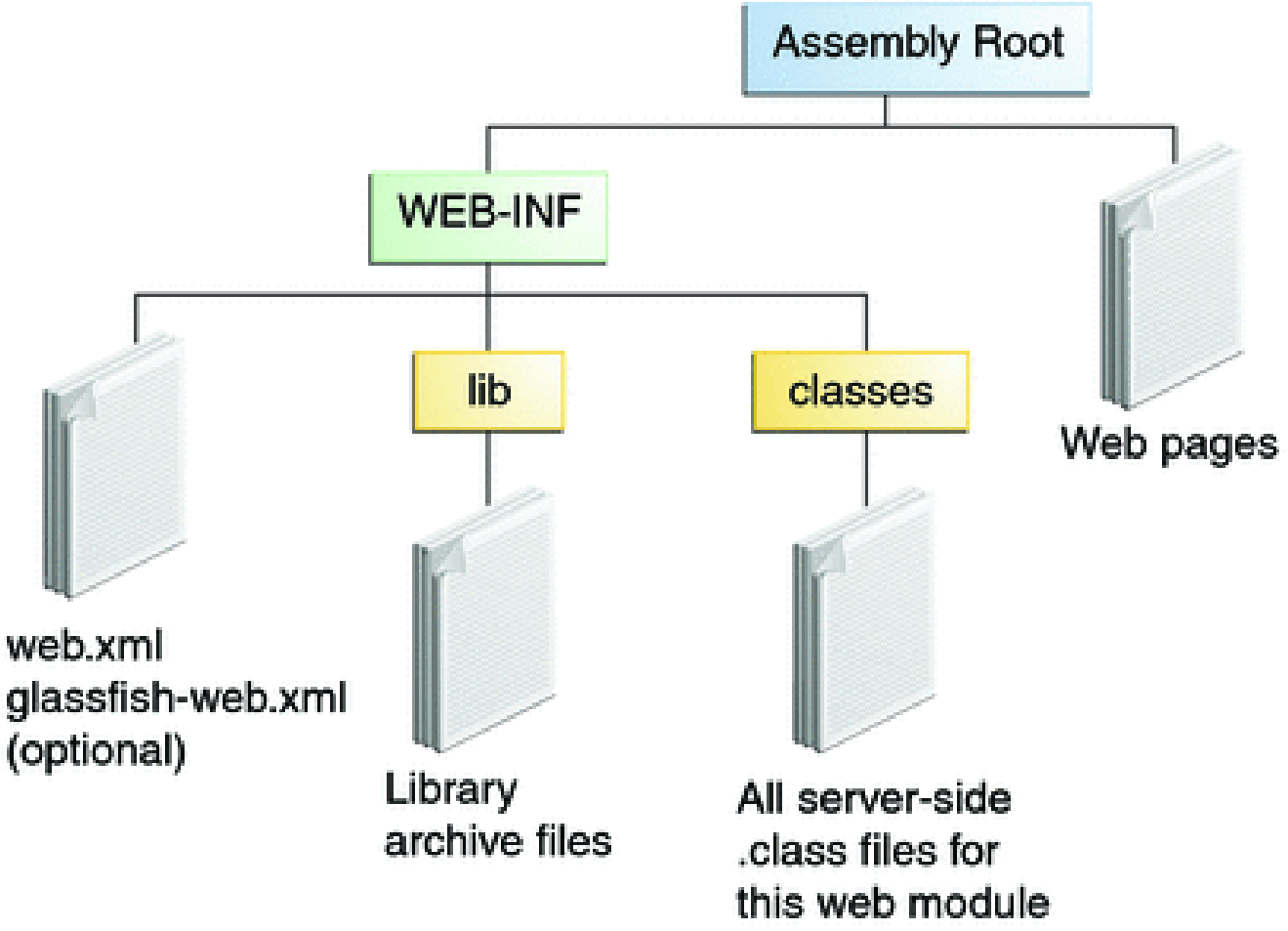
etc/jetty.xml

- Więcej przykładów

  http://docs.codehaus.org/display/JETTY/Syntax+Reference

Folder z aplikacjami

<jetty_home>/webapps/



webapps/appName/WEB-INF/jetty-web.xml

- Konfiguracja związana z konkretną aplikacją

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD Configure//EN"
"http://jetty.mortbay.org/configure.dtd">
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
  ...
</Configure>
```

W momencie wdrażania aplikacji serwer ładuje zawarte w pliku ustawienia po wczytaniu wszystkich innych

webapps/appName/WEB-INF/jetty-web.xml

- Domyślnie tylko zapis do logów

```
<Configure class="org.eclipse.jetty.webapp.WebAppContext">
    <Call class="org.eclipse.jetty.util.log.Log" name="debug">
        <Arg>executing jetty-web.xml</Arg>
    </Call>
</Configure>
```

Podsumowanie

- Zintegrowany m. in. ze środowiskiem Eclipse

Będziemy o tym mówić przy okazji zapoznawania się z GWT…