DevCAST ZONE

Tworzenie aplikacji internetowych z użyciem platformy Java EE

Moduł 2

Serwlety – podstawa aplikacji webowych

Spis treści

DevCAST ZONE

Struktura aplikacji webowej

- / – katalog zawierający wszelkie widoki, szablony, a także wszystkie pliki, które mają być dostępne publicznie. Serwer nie udostępnia jedynie plików zawartych w podkatalogach WEB-INF i META-INF
- /WEB-INF – katalog zawierający dodatkowe zasoby niezbędne do prawidłowego działania aplikacji, m.in. skompilowane klasy Java i biblioteki (JAR)

Struktura aplikacji webowej

- /WEB-INF/web.xml – deskryptor aplikacji webowej – plik zawierający wszystkie istotne ustawienia, mający kluczowe znaczenie dla działania całej aplikacji
- /WEB-INF/classes – katalog zawierający skompilowane klasy
- /WEB-INF/lib – katalog zawierający dodatkowe biblioteki

Struktura aplikacji webowej

- /META-INF – zawiera plik manifest.mf – jego obecność wynika z faktu, że każda aplikacja webowa przygotowana do wdrożenia ma postać pojedynczego archiwum WAR (Web ARchive), będącego specyficznym przypadkiem zwykłego pliku JAR

Struktura aplikacji webowej

- Typowy projekt (np. tworzony w środowisku Eclipse) zawiera również zintegrowany z projektem katalog z plikami źródłowymi aplikacji. Pliki te nie są jednak włączane do archiwum WAR
- Opcjonalnie: odwołania do usług sieciowych, plik persistence.xml (plik konfiguracyjny stosowany w technologii JPA)

Plik konfiguracyjny web.xml

- W pliku /WEB-INF/web.xml zawarte są praktycznie wszystkie informacje dotyczące konfiguracji aplikacji webowej Java EE
- W Java EE 6 jego znaczenie nieco maleje, z uwagi na możliwość definiowania pewnych informacji bezpośrednio w plikach źródłowych przy użyciu adnotacji

Plik konfiguracyjny web.xml

- Podstawowe zastosowanie pliku web.xml to powiązanie wzorców adresów URL z konkretnymi klasami serwletów
- Powiązanie odbywa się dwuetapowo:
  - nazwa klasy serwletu <-> nazwa serwletu
  - nazwa serwletu <-> jeden lub wiele wzorców URL

Plik konfiguracyjny web.xml

- Pierwszy etap: przydzielenie klasie serwletu przyjaznej nazwy:

```
<servlet>
    <servlet-name>ListaUzytkownikow</servlet-name>
    <servlet-class>szkolenie.serwlety.ListaUzytkownikow</servlet-class>
</servlet>
```

- Nazwa może zawierać spacje
- W pozostałej treści pliku web.xml należy odwoływać się do serwletu tylko za pomocą nazwy

Plik konfiguracyjny web.xml

- Drugi etap: przypisanie serwletowi przynajmniej jednego wzorca URL:

```
<servlet-mapping>
   <servlet-name>ListaUzytkownikow</servlet-name>
   <url-pattern>/serwlety/Lista</url-pattern>
</servlet-mapping>
```

- \W tym momencie użytkownik po wejściu na stronę http://localhost/Aplikacja/serwlety/Lista wywoła *ListaUzytkownikow*

Plik konfiguracyjny web.xml

- Jako wartość dla znacznika url-pattern można podać następujące rodzaje wzorców:
  - wzorzec dokładny – np. /serwlety/Serwlet
  - wzorzec wieloznaczny – np. /serwlety/* – zostaną dopasowane wszystkie żądania, które zawierają fragment /serwlety/
  - wzorzec rozszerzeń – *.xhtml – zostaną dopasowane wszystkie żądania, które odwołują się do zasobów o danym rozszerzeniu

Plik konfiguracyjny web.xml

- Serwlet może otrzymać parametry inicjalizacyjne
- Są one umieszczane w znaczniku <servlet>
- Dzięki nim można wpływać na działanie aplikacji bez rekompilacji plików źródłowych

Plik konfiguracyjny web.xml

- Przykład:

```
<servlet>
    <servlet-name>SerwletListaUzytkownikow</servlet-name>
    <servlet-class>szkolenie.serwlety.ListaUzytkownikow</servlet-class>
    <init-param>
        <param-name>TrybRaportu</param-name>
        <param-value>mail</param-value>
    </init-param>
</servlet>
```

Serwlety – tworzenie, konfiguracja, wdrożenie

- Serwlet jest klasą, która dziedziczy po klasie javax.servlet.GenericServlet
- Zadaniem serwletu jest obsługa żądań (*request*) i generowanie odpowiedzi (*response*)
- Serwlet nie jest samodzielną aplikacją – jest on uruchamiany przez kontener webowy

Serwlety – tworzenie, konfiguracja, wdrożenie

- Mimo istnienia uniwersalnej architektury, znaczna większość serwletów dziedziczy po klasie HttpServlet, przystosowanej rzecz jasna do obsługi żądań i odpowiedzi HTTP
- Po utworzeniu w odpowiedni sposób klasy serwletu i dodaniu niezbędnego kodu XML do pliku web.xml (lub adnotacji), serwlet jest gotowy do obsługi żądań (oczywiście po uruchomieniu serwera)

Serwlety – tworzenie, konfiguracja, wdrożenie

- Klasa serwletu, dziedzicząc po klasie HttpServlet, musi zawierać definicję co najmniej jednej z metod obsługujących żądania HTTP:
  - doGet(), doPost(), doDelete(), doPut()
- Każda z tych metod obsługuje żądania o określonej metodzie HTTP (GET/POST/PUT/DELETE)

Serwlety – tworzenie, konfiguracja, wdrożenie

- Wszystkie te metody mają identyczne sygnatury (poza nazwami):

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, java.io.IOException
```

- Parametr req reprezentuje obiekt żądania, podczas gdy resp – obiekt odpowiedzi

Serwlety – tworzenie, konfiguracja, wdrożenie

- Działanie serwletu w dużej mierze można uprościć do schematu:
  - pobierz niezbędne informacje z żądania
  - wykonaj logikę biznesową
  - wygeneruj odpowiedź, korzystając ze strumieni dostępnych w obiekcie odpowiedzi

Serwlety – tworzenie, konfiguracja, wdrożenie

- Przykład:

```
public class PierwszySerwlet extends HttpServlet {
        protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                PrintWriter pw = response.getWriter();
                pw.println("Pierwszy serwlet!");
                pw.flush();
        }
}
```

Serwlety – tworzenie, konfiguracja, wdrożenie

- Przy użyciu obiektu klasy PrintWriter można wygenerować dane tekstowe, które trafią do przeglądarki internetowej klienta
- W związku z tym w wywołaniach metod print/println można przekazywać kod HTML/CSS lub w dowolnym innym pożądanym języku obsługiwanym przez przeglądarkę (np. SVG)

Serwlety – tworzenie, konfiguracja, wdrożenie

- Do określenia typu przesyłanych danych należy stosować metodę setContentType odpowiedzi

- Jeśli zamiast obiektu klasy PrintWriter zostanie wykorzystany obiekt klasy ServletOutputStream, pobrany za pomocą metody getOutputStream odpowiedzi, można przesyłać dane binarne

Serwlety – tworzenie, konfiguracja, wdrożenie

- Przykład:

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        byte[] bufor = ... // pobranie pliku z dysku
        response.setContentType("image/png");
        response.addHeader("Content-Disposition", "attachment;
        filename=obrazek.png");
        response.setContentLength(bufor.length);
        OutputStream strumien = response.getOutputStream();
        strumien.write(b);
        strumien.flush();
}
```

Serwlety – tworzenie, konfiguracja, wdrożenie

- Istnieje możliwość dołączenia do odpowiedzi nagłówków i ciasteczek:
  - void addHeader(String nazwa, String wartosc)
  - void addCookie(Cookie ciastko)

  gdzie c jest obiektem reprezentującym pojedyncze ciastko (javax.servlet.http.Cookie)

Serwlety – tworzenie, konfiguracja, wdrożenie

- Aby pobrać parametry przesłane w żądaniu (czy to za pomocą metody GET, czy też POST), należy skorzystać z metody typu HttpServletRequest:

- String getParameter(String nazwa)

Serwlety – obsługa sesji

- Dostępna za pomocą metody getSession() obiektu żądania
- Reprezentowana za pomocą klasy HttpSession
- Kluczowe metody:
  - getAttribute() i setAttribute()
  - invalidate() – czyści sesję
  - isNew() – określa, czy sesja została utworzona w ramach aktualnego żądania

Atrybuty i ich zasięgi

- W aplikacjach Java EE istnieje możliwość umieszczania danych w różnych zasięgach
- Przykład – sesja i metody get/setAttribute()
- Oprócz tego można deklarować atrybuty w zasięgach żądania i kontekstu aplikacji

Atrybuty i ich zasięgi

- W przypadku żądania dostęp do atrybutów regulują metody get/setAttribute typu HttpServletRequest

- Kontekst aplikacji to specjalny obiekt, który udostępnia informacje na temat całej aplikacji i umożliwia dynamiczne modyfikowanie struktury aplikacji (np. dodawanie serwletów)

Atrybuty i ich zasięgi

- Atrybuty umieszczone w zasięgu kontekstu są dostępne w obrębie całej aplikacji!

- Dostęp do atrybutów jest możliwy za pomocą metody get/setAttribute typu ServletContext

- Instancja typu ServletContext jest dostępna na różne sposoby, m.in. przy użyciu obiektu żądania (req.getServletContext()) lub samego serwletu (this.getServletContext() w metodach doGet()/doPost())

Atrybuty i ich zasięgi

- Korzystając z atrybutów o zasięgu kontekstu należy pamiętać o możliwości współbieżnego dostępu przez różnych klientów

- Z tego względu warto synchronizować metody/fragmenty kodu, które modyfikują mapę atrybutów

Cykl życia aplikacji a obiekty nasłuchujące

- Java EE udostępnia szereg obiektów nasłuchujących, dzięki którym istnieje możliwość reagowania na różne sytuacje, które mają miejsce podczas działania aplikacji
- Najpierw należy wybrać interfejs, który udostępnia żądane zdarzenia, a następnie dodać niezbędną informację do pliku web.xml

Cykl życia aplikacji a a obiekty nasłuchujące

- Zdarzenia cyklu życia dla obiektów różnych zasięgów:
  - ServletRequestListener – dla żądań:
    - requestInitialized, requestDestroyed
  - HttpSessionListener – dla obiektów sesji
    - sessionCreated, sessionDestroyed
  - ServletContextListener – dla kontekstu aplikacji
    - contextInitialized, contextDestroyed

Cykl życia aplikacji a obiekty nasłuchujące

- Zdarzenia modyfikacji atrybutów o różnych zasięgach:
  - ServletRequestAttributeListener:
    - attributeAdded, attributeRemoved, attributeReplaced
  - ServletContextAttributeListener:
    - j.w.
  - HttpSessionAttributeListener:
    - j.w.

Cykl życia aplikacji a obiekty nasłuchujące

- Zdarzenia odpowiedzialne za informowanie obiektów o umieszczeniu ich w zasięgu sesji:
  - HttpSessionBindingListener:
    - valueBound() – obiekt został dodany do mapy atrybutów o zasięgu sesji
    - valueUnbound() – obiekt został usunięty z mapy atrybutów o zasięgu sesji

Cykl życia aplikacji a obiekty nasłuchujące

- Zdarzenia odpowiedzialne za informowanie obiektów o przeniesieniu sesji do innej maszyny wirtualnej
  - HttpSessionActivationListener:
    - sessionDidActivate() – sesja została aktywowana po przeniesieniu z innej VM
    - sessionWillPassivate() – sesja zostanie wyłączona w celu przeniesienia do innej VM

Cykl życia aplikacji a obiekty nasłuchujące

- Po utworzeniu klasy obiektu nasłuchującego należy dodać niezbędny kod do pliku web.xml (lub zastosować adnotację):

```
<listener>
  <listener-class>szkolenie.sluchacze.SluchaczSesji</listener-class>
</listener>
```

Bezpieczeństwo

- Aplikacje webowe pozwalają na konfigurację różnych aspektów bezpieczeństwa w pliku web.xml (w Java EE 6 także za pomocą adnotacji)
- Konfiguracja ta jest zupełnie odseparowana od kodu serwletów czy też stron JSP – pomijając opcjonalne mechanizmy, z których można korzystać (np. sprawdzenie, czy użytkownik przynależy do roli)

Bezpieczeństwo

- Najważniejsze aspekty konfiguracji bezpieczeństwa:
  - określenie ról aplikacji
  - określenie źródła danych bezpieczeństwa (w przypadku serwera Apache Tomcat może być to statyczny plik tomcat-users.xml, a mogą być to wybrane tabele z bazy danych)

Bezpieczeństwo

- Najważniejsze aspekty konfiguracji bezpieczeństwa:
  - określenie wzorców URL, których dotyczą reguły bezpieczeństwa, wraz z dozwolonymi metodami HTTP
  - powiązanie wzorców i ról (w pliku web.xml), a także ról i użytkowników (np. w bazie danych) pozwala na utworzenie kompletnego systemu bezpieczeństwa w aplikacji

Bezpieczeństwo

- Najważniejsze aspekty konfiguracji bezpieczeństwa:
  - Istnieją też dodatkowe aspekty, jak określenie minimalnego stopnia zabezpieczeń (np. SSL), metody uwierzytelniania, itd.
- Separacja ustawień bezpieczeństwa od kodu Java pozwala na przydział zadań związanych z konfiguracją i bezpieczeństwem np. innemu pracownikowi

Bezpieczeństwo

- Przykład:

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>user-resource</web-resource-name>
     <url-pattern>/nowyTemat</url-pattern>
     <url-pattern>/temat</url-pattern>
     <url-pattern>/index</url-pattern>
     <http-method>GET</http-method>
     <http-method>POST</http-method>
   </web-resource-collection>
   <auth-constraint>
     <role-name>uzytkownik</role-name>
   </auth-constraint>
   <user-data-constraint>
     <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
 </security-constraint>
 <login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
     <form-login-page>/login</form-login-page>
     <form-error-page>/blad</form-error-page>
   </form-login-config>
 </login-config>
 <security-role>
   <role-name>uzytkownik</role-name>
 </security-role>
```

Filtry a żądania

- Filtr to mechanizm, który pozwala na wykonanie pewnej czynności przed przekazaniem sterowania do odpowiedniego serwletu
- Filtry, podobnie jak serwlety, mogą być aplikowane do wzorców URL

Filtry a żądania

- Filtry są stosowane wszędzie tam, gdzie konieczna jest wstępna obróbka żądań, np.:
  - rejestrowanie informacji w dzienniku
  - szyfrowanie/deszyfrowanie
  - kompresja
  - dynamiczna decyzja o przekazaniu sterowania do serwletu

Filtry a żądania

- Przykład:

```
// przekaż sterowanie do serwletu, jeśli istnieje parametr o nazwie X
public class FiltrUzytkownikow implements Filter {
        public void doFilter(ServletRequest request, ServletResponse
        response, FilterChain chain) throws IOException,
        ServletException {
                System.out.println("Przed obsługą");
                if (request.getParameter("nazwaUzytkownika")!=null)
                        chain.doFilter(request, response);
                System.out.println("Po obsłudze");
        }
        public void init(FilterConfig fConfig) throws ServletException {}
        @Override
        public void destroy() {}
}
```

Filtry a żądania

- Przykład (konfiguracja w pliku web.xml):

```
<filter>
  <filter-name>FiltrUzytkownikow</filter-name>
  <filter-class>szkolenie.filtry.FiltrUzytkownikow</filter-class>
</filter>
<filter-mapping>
  <filter-name>FiltrUzytkownikow</filter-name>
  <url-pattern>/serwlety/*</url-pattern>
</filter-mapping>
```