

Listenery, Filtry, JNDI

Listeners – vytvoření

- Listeners jsou Java komponenty umožňující reagovat na události vyvolávané webovým kontejnerem:
 - Vznik/zánik oborů platnosti: Servlet kontextu, session, requestu,
 - přidání/odebrání/nahrazení atributu v servlet kontextu, session, requestu.
- Listener se tvoří jako třída implementující předpřipravená rozhraní. Podle toho, na jaké události chceme reagovat, implementujeme rozhraní:
 - `ServletContextListener`,
 - `ServletContextAttributeListener`,
 - `HttpSessionListener`,
 - `HttpSessionAttributeListener`,
 - `ServletRequestListener`,
 - `ServletRequestAttributeListener`.

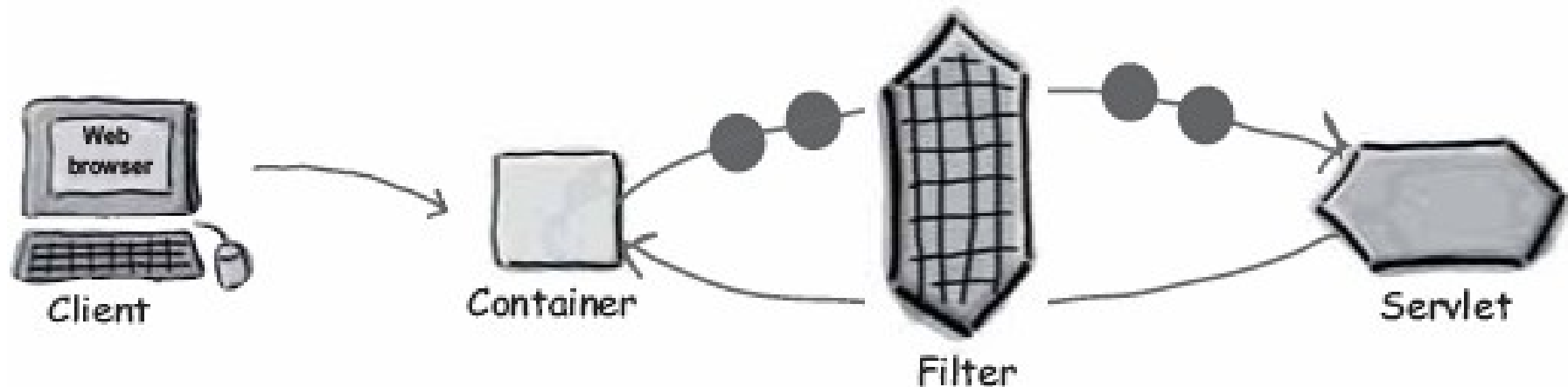
Listeners – zapojení

- Listeners se konfigurují ve `web.xml`. V konfiguraci stačí uvést třídu listeneru, aby kontejner "věděl", co instanciovat při nasazení aplikace:

```
<listener>  
    <description>ServletContextListener</description>  
    <listener-class>cz.firma.app.web.MyListener</listener-class>  
</listener>
```

Filtry

- Filtry jsou Java komponenty velice podobné Servletům. Slouží pro zachycení a modifikaci requestu předtím, než byl poslán Servletu, nebo pro zpracování response potom, co Servlet svůj kód již ukončil. Servlet tudíž nemá o filtru žádné tušení.



Použití filtrů

- Filtr na requestu:
 - Nastavení bezpečnostního přístupu k Servletu
 - Změna obsahu requestu získaného od klienta
 - Provádění auditu nebo logování requestu
 - Filtr na response:
 - Komprimování response výstupu
 - Změna obsahu response
 - Vytvoření úplně jiného response
 - Pozn: Neexistuje fyzické členění na request a response filtry!
Existuje pouze jedno rozhraní `javax.servlet.Filter`, které filtr implementuje!
- Lepší řešení je
Java EE Security
- Lepší řešení je
předřazení Apache
HTTPD serveru
- Může znepréhlednit
chování aplikace
-

Konfigurace filtru

- Filtry se zapojují ve web.xml. Je možné je řetězit, nemají ale o sobě navzájem tušení. Níže je filtr pro vynucení UTF-8 v Tomcatu, více zde: <http://wiki.apache.org/tomcat/FAQ/CharacterEncoding>

```
<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
  <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CharacterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Konfigurace filtru

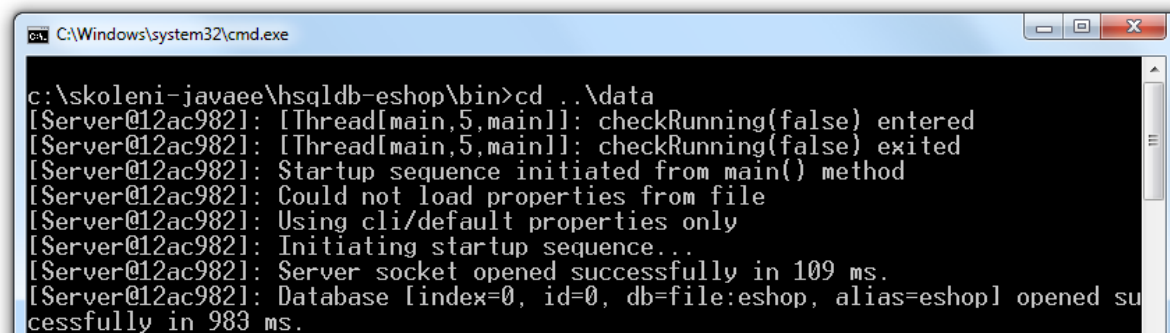
- Pokud používáte Spring, pak můžete použít tento filtr:

```
<filter>
  <filter-name>characterEncodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>characterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

JNDI, spuštění HSQLDB databáze

- JNDI (Java Naming and Directory Interface) je služba, která umožňuje Java aplikacím získat objekty na základě jejich jména.
- JNDI se velice často používá k tomu, aby aplikace získala připojení do databáze, LDAPu, ale i proměnné prostředí.
- V této přednášce si ukážeme připojení do HSQLDB databáze pomocí JNDI. Nejprve je nutné HSQLDB databázi spustit: [adresar-skoleni]/hsqldb-XXX/bin/runDatabase.bat Výsledkem by mělo být cmd okno, ve kterém je informace, že databáze byla úspěšně spuštěna. Pokud takové okno nezůstane běžet, pak zkontrolujte nastavení JAVA_HOME a Path.



```
C:\Windows\system32\cmd.exe
c:\skoleni-javaee\hsqldb-eshop\bin>cd ..\data
[Server@12ac982]: [Thread[main,5,main]]: checkRunning(false) entered
[Server@12ac982]: [Thread[main,5,main]]: checkRunning(false) exited
[Server@12ac982]: Startup sequence initiated from main() method
[Server@12ac982]: Could not load properties from file
[Server@12ac982]: Using cli/default properties only
[Server@12ac982]: Initiating startup sequence...
[Server@12ac982]: Server socket opened successfully in 109 ms.
[Server@12ac982]: Database [index=0, id=0, db=file:eshop, alias=eshop] opened successfully in 983 ms.
```


JDBC ovladač

- Pro připojení do databáze je nutné mít v classspat aplikace, která bude toto připojení používat, JDBC ovladač. Touto aplikací je Apache Tomcat. JNDI se připojuje do databáze a bude nakonfigurované právě na Tomcatu. Zkopírujte do adresáře [adresar-skoleni]/apache-tomcat-XXXX/lib soubor [adresar-skoleni]/hsqldb-XXXX/lib/hsqldb.jar.
- Poté do souboru context.xml v conf adresáři serveru Apache Tomcat přidejte definici JNDI Resource:

```
<Resource name="jdbc/hsqldb"
```

Textový název
tohoto Resource

```
auth="Container" type="javax.sql.DataSource"
```

```
driverClassName="org.hsqldb.jdbcDriver"
```

```
factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
```

```
username="sa" password=""
```

```
url="jdbc:hsqldb:hsqldb://localhost/eshop" />
```

JDBC
ovladač

Connection pool pomocí
knihovny DBCP,
která je v Tomcatu

Pokud máte
server
integrovaný
v Eclipse,
pak musíte
změnit soubor
context.xml
v projektu
Servers!!!

Získání DataSource pomocí JNDI

- Poté v aplikaci kde potřebujeme přístup k DataSource použijeme následující kus kódu:

```
DataSource dataSource = null;

try {

    InitialContext ctx = new InitialContext();

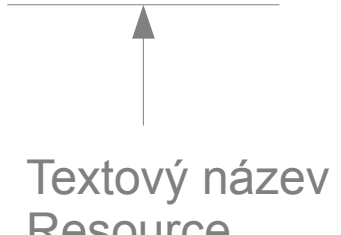
    // java:comp/env je standardne pouzivana cesta,
    // za niz je jmeno zdroje

    dataSource = (DataSource) ctx.lookup("java:comp/env/jdbc/hsqldb");

} catch (NamingException ex) {

    ex.printStackTrace();

}
```



Textový název
Resource

- V Java EE 6 nebo při použití Springu lze instanci DataSource přes JNDI získat výrazně jednodušeji.