TJV – BI-WSI-SI-27

Architektura podnikových aplikací. Popis jednotlivých vrstev JEE aplikací: klientská vrstva, webová vrstva, vrstva obchodní logiky, EIS vrstva.

1 Architektura podnikových aplikací

Java Enterprise Edition (JEE) definuje 4 vrstvy architektury. Každá využívá zdroje vrstvy, která je hierarchicky pod ní a poskytuje rozhraní vrstvě nad ní.

Tato architektura se občas nazývá 3vrstvá - business a aplikační (webová) vrstva totiž většinou běží na jednom stroji a spojuje se do jedné.

1.1 Klientská vrstva

Nemusí být nutně v Javě. Může to být třeba webová aplikace. Komunikuje se střední vrstvou (odesílá ji a přijímá od ní data). Běžná forma komunikace je přes HTTPS protokol s využitím například RESTu. Držet se může i session.

Klientská aplikace může komunikovat s webem nebo přímo s EJB komponentami v business vrstvě.

1.2 Webová vrstva

Webová vrstva je tvořena servlety, JSP a JSF soubory a případně také JavaBeans komponentami. Tyto komponenty zpracovávají klientské požadavky a generují odpověď, která je následně zaslána do webového prohlížeče klienta.

Při zpracovávání odpovědi může webová vrstva komunikovat s vrstvou obchodní logiky (business vrstvou) a získávat od ní určité informace.

Servlet dědí od třídy HttpServlet a má například metody: `HttpServlet#doGet(HttpServletRequest, HttpServletRespo`
`HttpServlet#doPost(HttpServletRequest, HttpServletResponse)`

Umí tak reagovat na GET, POST, PUT, DELETE, PATCH, … požadavky, které mohou přijít. Do HttpServletResponse vypisuje odpověď třeba v HTML kódu.

Java Server Pages alias JSP jsou textové soubory s příponou .jsp, umístěné ve webové aplikaci, které jsou při prvním požadavku na zobrazení automaticky převedeny servletovým kontejnerem na servlet (.java) a přeloženy do Java třídy (.class). Servlety vzniklé z JSP stránek jsou tedy mapovány na URL původního textového souboru.

```
<body>
 <%  //kód uvnitř metody service()
     if(request.getMethod().equals("GET")) {
 %>
  <form method="post">
      Zadejte číslo: <input name="cislo" value="">
      <input type="submit" >
  </form>
  <%
      } else if (request.getMethod().equals("POST")) {
```

```
            String cislo = request.getParameter("cislo");
            String plusjedna = prictiJedna(cislo);
    %>
        Výsledek <%out.println(cislo);%> + 1 je <%=plusjedna%>
    <%
        }
    %>
</body>
```

JSF potom taky slouží k jednoduššímu generování webovek.

1.3 Obchodní (business) vrstva

Celá logika aplikace. EJB komponenty přijímají požadavky z webové nebo klientské vrstvy, zpracovávají je a následně vygenerují odpověď. Obsahuje například mapování entit DB na objekty (@Entity). Počítá nějaké šílené věci, dělá validace dat, …O EJB více v další otázce.

1.4 EIS vrstva

Enterprise Information system (= podnikový informační systém)

Tato vrstva představuje veškeré externí systémy, jejichž funkcionalitu či data enterprise aplikace využívá. Může se jednat například o ERP systém, databázový systém atp. Komunikaci s EIS zpravidla zajišťuje business vrstva.

2 Zdroje

Celý převzato z https://docs.google.com/document/d/1OU75LDsImR4cEsQoyfGNyibG5ECJhGRKCfJqrUlpl1Q/edit#

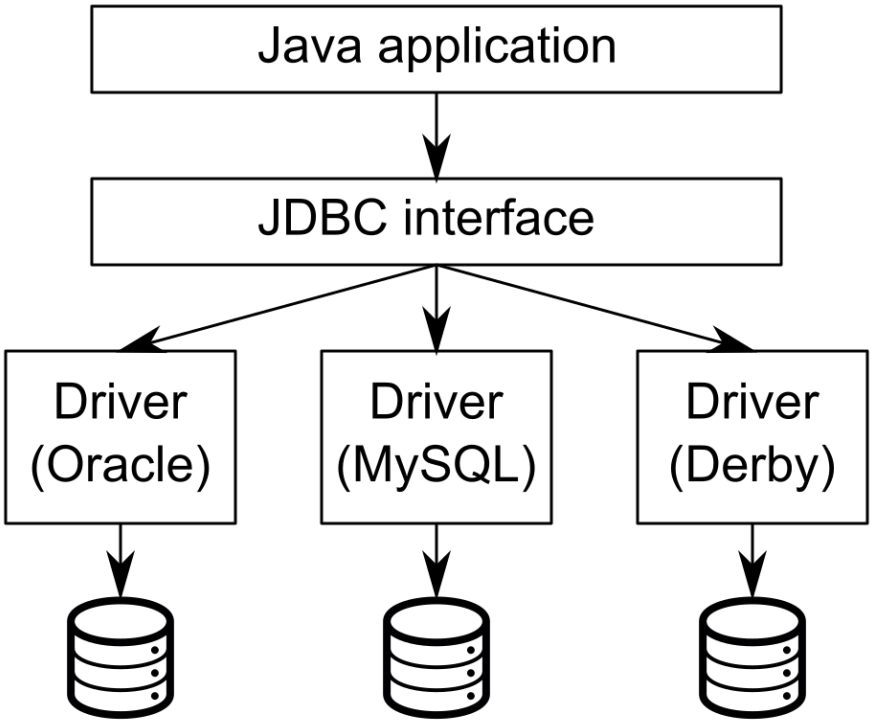
- JSP - MUNI - https://kore.fi.muni.cz/wiki/index.php/Java_Server_Pages
- 4vrstvá architektura - VŠE - https://java.vse.cz/jsf/chunks/ch02s03.html
- BI-TJV 4. Přednáška - https://docs.google.com/presentation/d/12jr7jnWbx_S4sIwLd0v9HmUyX1uuAP3-z6HOdI edit#

TJV – BI-WSI-SI-28

JDBC v podnikových aplikacích: popis JDBC obecně, popis dat datového zdroje (DataSource), nastavení datových zdrojů v aplikačním serveru (JDBC Connection Pool, JDBC Resources).

1 JDBC Java Database Connectivity

- Java rozhraní pro připojení k relační databázi
- Součástí přímo Java SE už od verze 1.1
- Umožňuje vytvořit takzvané spojení (connection) s databází
- Přes toto spojení lze pak spouštět dotazy SELECT, CREATE, INSERT, UPDATE a DELETE.
- JDBC je pouze obecné rozhraní. Pro připojení ke konkrétnímu typu databáze (MySQL, PostgreSQL....) je potřeba takzvaný driver.
- JDBC Driver realizuje spojení s konkrétní databází → implementuje JDBC interface



2 Připojení se k DB

Nejdříve se k databázi připojíme (pozn.: try-with-resources funguje od Javy 7) pomocí třídy `DriverManager`. Před Javou 6 bylo třeba explicitně definovat třídu driveru a načíst ji.

```
try (Connection conn = DriverManager.getConnection(
     "jdbc:mysql://localhost/mydb",
     "myLogin",
     "myPassword")) {

  // zde používáme vytvořené spojení

}  // Java VM se postará o uzavření spojení
```

Využití DriverManageru je vcelku nepraktické (musíme být v try bloku), a proto vznikl DataSource. Jedná se o interface, která vrací stejnou Connection jako DriverManager. Vývojáři DB musí poskytovat konkrétní implementaci DataSource. Connection lze získat bez parametrů nebo s pomocí dvojice „jméno a heslo“.

```
com.dbaccess.BasicDataSource ds = new com.dbaccess.BasicDataSource();
ds.setServerName("mujserver");
ds.setDatabaseName("mojedatabaze");

// Metoda odstinena od konkretniho driveru → polymorfismus
public void doThings(DataSource ds) {
  Connection con = ds.getConnection("user", "psswd");
}
```

Detaily připojení tak nemusí být napevno v aplikaci. DriverManager dále nabízí connection pooling (viz níže).

3 Práce s DB

Příkazy se vykonávají pomocí třídy Statement.

```
try (Statement stmt = conn.createStatement()) {
  stmt.executeUpdate("INSERT INTO table VALUES ('neco')");
}
```

Existuje pak i PreparedStatement, do kterého lze dodávat argumenty dynamicky.

```
try (PreparedStatement ps = conn.prepareStatement(
     "SELECT * FROM person WHERE name = ? AND age = ?")
) {
    ps.setString(1, "Poor Yorick");
    ps.setInt(2, 90);
}
```

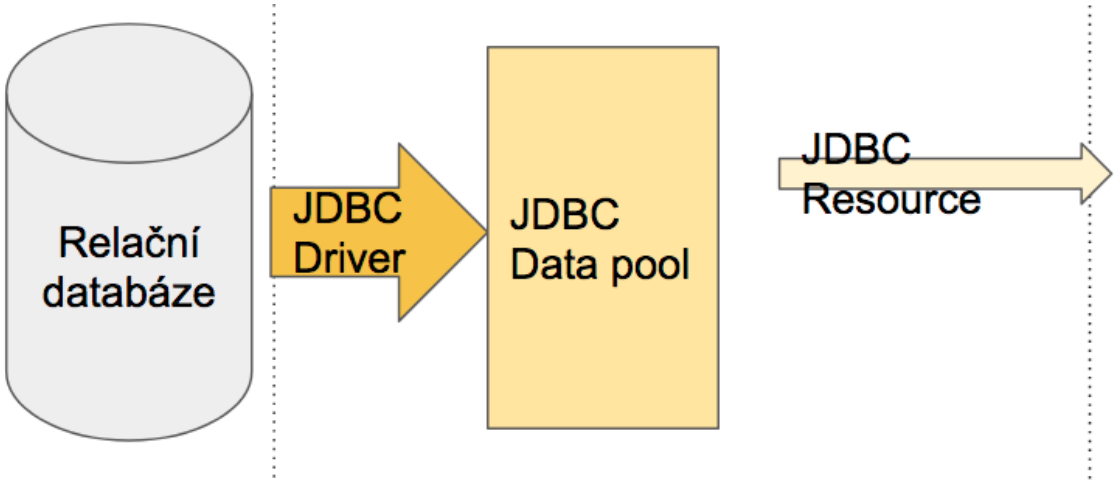
Návratová hodnota je `ResultSet`. Je to takový převlečený návrhový vzor `Cursor ~ Iterator`. Můžeme s pomocí něj iterovat přes řádky výsledků a podle indexů sloupce si sahat na jednotlivé hodnoty.

```
try (Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT * FROM person")
) {
  while(rs.next()) {
```

```
    String personName = rs.getString(1); // index sloupce
    int personAge     = rs.getInt(2);
  System.out.printf("Osoba: %s (%d)\n", personName, personAge);
  }
}
```

Dále lze pomocí příznaků specifikovat, jestli iterace může být pouze jednosměrná nebo oboustranná.

Jeden příkaz znamená jeden atomický dotaz do databáze. V rámci jedné transakce lze ale vykonat více příkazů. Ovlivňuje to metoda `Connection#setAutoCommit(boolean)`. Pokud nastavíme autocommit na false, tak je potřeba transakci ručně odeslat - `Connection#commit()`.



4 JDBC Connection pooling

Problém - pro každé vykonání dotazu je třeba se k databázi připojit. Připojení stojí nějaký čas a není dobré se neustále připojovat a odpojovat. Connection pooling zajišťuje, že připojení proběhne na začátku a zachová se i po vykonání dotazu. Pro další dotaz je existující připojení znovupoužito.

Connection pooling probíhá v Javě automaticky, pokud je využit `DataSource`. Podpora musí být na straně implementace této `interface`. JDBC 3.0 API Framework specifikuje sadu rozhraní, které musí být implementovány.

- `ConnectionPoolDataSource` - implementace od autorů databází slouží jako factory, která vytváří PooledConnection
- `PooledConnection` - řídí připojení k databázi
- `ConnectionEventListener` - listener změn v připojeních k DB

Nastavení Connection poolu se v TJV dělalo přes Glassfish. Na localhost webu se zadaly přihlašovací údaje do databáze, adresa databáze atp.

Jak na to - viz video `https://youtu.be/f1z-3zlkXj8?t=2m27s`

5 JDBC Resource

Poskytuje prostředky pro připojení k databázi. Pro vytvoření je potřeba speficikovat Connection pool, se kterým bude spjat. Má vlastní jméno (JNDI name), které dle konvence začíná "jdbc/".

Přístup ke zdroji lze získat přes `InitialContext`:

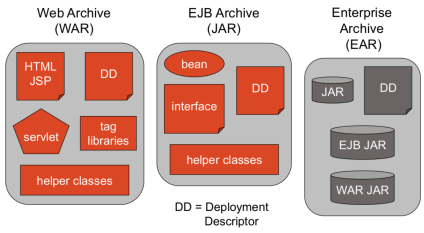
```
InitialContext ic = new InitialContext();
DataSource source = (DataSource) ic.lookup("jdbc/my_resouce_name");
```

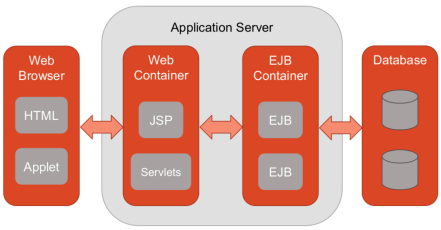
TJV – BI-WSI-SI-29

Aplikační server a JNDI služba: popis aplikačního serveru, webový kontejner, EJB kontejner, využití JNDI ke konfiguraci datových zdrojů.

1 Aplikační server

- Serverovou aplikaci zastřešující všechny knihovny, které dle specifikací Java EE platformy zajišťují požadovanou funkcionalitu, označujeme pojmem aplikační server.
- Tyto knihovny implementují veškerá API obsažená v Java EE.
- Kromě toho poskytuje aplikační server další klasické služby jako např.
  - administrátorskou konzoli,
  - logování atp.
- Ze známých implementací Java EE platformy můžeme zmínit např. JBoss od firmy Red Hat či JRun od firmy Adobe Systems. Referenční implementace Oracle je šířena pod hlavičkou projektu GlassFish.
- Zajišťuje:
  - Vývoj webových aplikací - Java Servlets, Java Server Pages (JSP), JavaServer Faces (JSF)
  - Contexts and Dependency Injection - vkládání závislostí
  - Přístup k relačním databázím - Java Persistence API (ORM)
  - Vývoj sdílené business logiky - Enterprise Java Beans (EJB)
  - Přístup k legacy systémům - Java Connector Architecture (JCA)
  - Přístup ke zprávovému middleware - Java Messaging Services (JMS)
  - Komponenty zajišťující integraci webových aplikací a portálů - Portlety
  - Podpora technologií Webových služeb- SOA, REST





Aplikační server obsahuje dva kontejnery - webový a EJB. Ty obsahují komponenty, instantizují je a řídí jejich životní cyklus. Komponenty nemohou žít bez kontejneru.

Výsledkem je .ear balíček, který je vytvořen třeba vývojovým prostředím za pomocí Mavenu, Antu nebo novějšího Gradlu. Všechny tři zmíněné nástroje jsou tzv. build systémy. Další ze známějších je Make (známe z BI-PA2), ale ten se tady nepoužívá.

EJB komponenty

- Odpovídá business vrstvě z otázky architektura aplikací
- Více o EJB v otázce EJB

- Balíček jar

Webový kontejner (podobně jako EJB kontejner - viz EJB otázka)

- Mapuje URL adresy na servlety
- Zajišťuje bezpečnost - může uživatel přistoupit k danému kontejneru?
- Řídí životní cyklus servletů - vytváření, mazání
- Pracuje s dotazy a odpověďmi serveru
- Spravuje pool servletů

Webové komponenty

- odpovídá definici webové vrstvy z otázky architektura aplikací
- Balíček .war

2 JNDI

Java Naming and Directory Interface

Aplikační server zajišťuje službu JNDI. Ta na základě klíčového slova (unikátního názvu zdroje) zajistí vyhledání a předání existující konfigurace žádající aplikaci. Využívá se třeba pro získání instance EJB nebo DataSource (pro připojení k databázi).

Aplikační server dále zajišťuje připojení ke zdroji a obnovení komunikace, pokud se něco podělá. Pro získání instancí slouží třída `InitialContext`, kterou si můžeme kdykoliv vytvořit.

Pomocí metody `InitialContext#lookup(Name)` lze získat instanci objektu. Pokud pracujeme s databází, tak je potřeba v nastavení aplikačního serveru nastavit příslušný JDBC Resource a dát mu JNDI jméno (String) - viz otázka JDBC.

Jméno EJB je generováno automaticky ze jména třídy a umístění.

```
  InitialContext ic = new InitialContext();
DataSource source = (DataSource) ic.lookup("jdbc/my_resouce_name");
MyBean bean = (MyBean) in.lookup("java:com/env/MyBean");
```