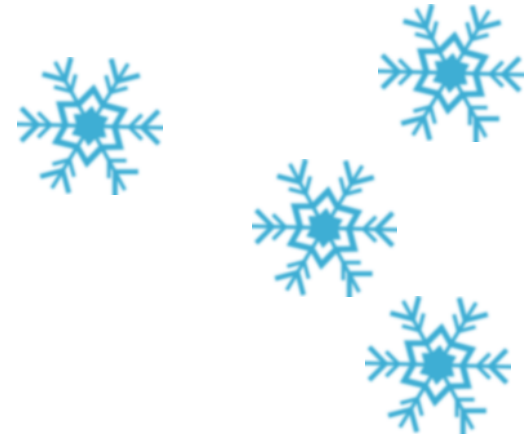




CERIT  
Masarykova univerzita



---

# PV168 – Java Database Connectivity (JDBC)

Petr Adámek & Tomáš Pitner

Březen 2019

---



## Persistence dat pomocí JDBC

- Technika přístupu k především **relačním databázím**
- **Java DataBase Connectivity (JDBC)**
  - Součást **Java SE** (+nutný **ovladač** pro DBMS)
- **Nízkoúrovňové** rozhraní
- Často překryté **Object-Relational Mapping (ORM)**
- Nebo pomocnými rozhraními jako **Spring JDBC**
- Někdy však nezbytné –
  - **Výkon**
  - **Přímý** přístup k DBMS

## Rámcový postup práce s JDBC

1. Aktivace **ovladače**
2. Vyhledání datového zdroje + **otevření** spojení
3. Používání spojení
  - Veškeré **operace** nad DB
  - Je nutné znát **SQL**, s tím se de facto komunikuje
4. **Uzavření** spojení (i v případě chyby!)
  - Lze přitom využít javové „try-with-resources“

## Aktivace JDBC ovladače

- Zavedení třídy **ovladače** pro příslušný DBMS
  - Manuální
  - Automatické
  - Třída ovladače musí vždy být na `classpath`
- K dispozici pro konkrétní DBMS, např.:
  - `Class.forName("com.mysql.jdbc.Driver");`

## Navázání spojení přímo

- Pomocí třídy `DriverManager`, metoda `getConnection`
- Vč. uvedení příp. **jména**, **hesla** databázového uživatele
- Nutnost znát všechny údaje pro připojení a mít je konfigurovatelné mimo zdrojový kód, tj. nemít je v kódu „natvrdo“, ale v konfiguraci jinde

```
String url="jdbc:mysql://localhost:3306/database?useUnicode=true";  
Connection conn  
    = DriverManager.getConnection(url, "user", "password");
```

## Navázání spojení přes DataSource

- Umožní vyjmout starost o **konfiguraci** z programu směrem ke správci aplikace/databáze
- Využití javových anotací, pomocí nichž provedeme **Dependency Injection** (vložení závislostí)

```
@Resource (name="jdbc/moje")
```

```
private DataSource source;
```

```
// Spojení získáme pomocí:
```

```
Connection conn = source.getConnection() ;
```

## Navázání spojení přes JNDI

- Vlastně jiná technika vyhledání DataSource
- Bez anotací, přímo vyhledáním datového zdroje
- Rozhraní **Java Naming and Directory Interface (JNDI)**

Context *context*

```
= new InitialContext().lookup("java:comp/env");
```

DataSource *source*

```
=(DataSource) context.lookup("jdbc/test");
```

## Konfigurace JNDI

- Nutná **konfigurace** dle konvencí aplikačního serveru nebo webového kontejneru
- Např. u **Tomcat** soubor `context.xml` v `META-INF`

```
<Resource auth="Container"  
driverClassName="org.hsqldb.jdbcDriver"  
maxActive="100" maxIdle="30" maxWait="10000"  
name="jdbc/test"  
password="" type="javax.sql.DataSource"  
url="jdbc:hsqldb:mem:addressbook"  
username="sa"/>
```



## Znovupoužití spojení – Connection pooling

- Otvírání u udržování spojení je náročné na zdroje
- Proto se často po opuštění znovuvyžívá prostřednictvím „poolu“ otevřených spojení
  - Vypůjčím, použiji a vrátím spojení
- Je třeba speciální knihovna, která je k dispozici u kontejneru (např. Tomcat) nebo poskytnuta třetí stranou
- Použití známého Apache Commons DBCP (DB Connection Pool) <https://www.baeldung.com/java-connection-pooling>

## Příklad použití Connection pool

```
import org.apache.commons.dbcp2.BasicDataSource;
public DataSource dataSource() throws IOException {
    Properties p = new Properties();
    p.load(this.getClass()
        .getResourceAsStream("/jdbc.properties"));

    BasicDataSource bds = new BasicDataSource();
    bds.setDriverClassName(p.getProperty("jdbc.driver"));
    bds.setUrl(p.getProperty("jdbc.url"));
    bds.setUsername(p.getProperty("jdbc.user"));
    bds.setPassword(p.getProperty("jdbc.password"));
    return bds;
}
```

## Komunikace s databází přes JDBC

- De facto posílání **SQL příkazů a čtení výsledků**
- Nutné znát **SQL**
- Vč. **dialektu** příslušného DBMS
- Možnost využít **specifických** vlastností DBMS
- Určitá **závislost** na konkrétním DBMS
- Psaní spousty „**boilerplate** code“
- Lze využít **knihoven** třetích stran pro usnadnění

## (JDBC) Statement

- `// musíme již mít Connection conn`
- `Statement st = conn.createStatement() ;`

## Provedení SQL příkazu nad `Statement`

```
boolean result
```

```
= st.execute("SELECT * FROM myTable;");
```

- V `result` je pouze indikován úspěch/neúspěch dotazu
- Takto se vykonávají jak operace typu čtení, tak změny v databázi
  - U čtení získáme vrácené záznamy pomocí následného volání `getResultSet`
  - U zápisu pomocí volání `getUpdateCount`

## Provedení SQL příkazu nad `Statement`

- **Čtení (SELECT):** výsledek = relace (`ResultSet`)

```
ResultSet resultSet = st.executeQuery (  
    "SELECT * FROM myTable;");
```

- **Modifikace:** výsledek = počet (`int`)

```
int updatesCount = st.executeUpdate (  
    "DELETE FROM myTable WHERE a = 1;");
```

## Zpracování výsledků

- **Výsledkem čtení (SELECT) je** `ResultSet`, ten následně lze zpracovat:
  - Postupným procházením jednotlivých *záznamů* (vrácených řádků)
  - Zpřístupněním jednotlivých *atributů*, např.

```
while (resultSet.next()) { // iterace
    // 1. atribut, musíme znát jeho typ
    int a = resultSet.getInt(1);
    String b = resultSet.getString(2);
    boolean c = resultSet.getBoolean(3);
}
```

## Další možnosti ResultSet

- Indexováno od **1** (nikoli od 0!)
- Lze dle **jména** atributu získat jeho index

```
int indexColumnA =  
    resultSet.findColumn("columnA");  
while (resultSet.next()) {  
    int a = resultSet.getInt(indexColumnA);  
    ...  
}
```



## Pohyb po záznamech ResultSet

- Některé typy JDBC ovladačů podporují:
  - zpětný posun v tabulce  
`ResultSet.previous()`
  - posun o libovolný počet záznamů (řádků v tabulce)  
`ResultSet.relative(int)`
  - přístup k libovolnému záznamu  
`ResultSet.absolute(int)`
- Nutno nastavit před provedením příkazu

## Předpřipravené dotazy

- Některé typy JDBC ovladačů podporují **PreparedStatement**:

```
PreparedStatement insertStatement =  
conn.prepareStatement (  
    "INSERT INTO myTable (a,b,c) VALUES (?, ?, ?) ;") ;  
insertStatement.setInt (1, 1) ;  
insertStatement.setString (2, "Ahoj") ;  
insertStatement.setBoolean (3, false) ;  
insertStatement.execute () ;
```

## Předpřipravené dotazy

- Výhody *PreparedStatement*:
  - Vyšší **odolnost proti SQL Injection** (vložení výkonného kódu SQL namísto prosté hodnoty parametru) u *PreparedStatement* z principu nelze
  - Vyšší **výkon**

## Získávání generovaných klíčů

```
st.execute(  
    "INSERT INTO myTable (b,c) VALUES ('hello', false);",  
    Statement.RETURN_GENERATED_KEYS);  
ResultSet keys = st.getGeneratedKeys();
```

## Výjimky v JDBC kódu try-with-resources

```
try(Connection con=dataSource.getConnection()) {
    try(PreparedStatement st=con.prepareStatement(
        "select * from books")) {
        try(ResultSet rs=st.executeQuery()) {
            List<Book> books = new ArrayList<>();
            while (rs.next()) {
                books.add(new Book(rs.getLong("id"),
                    rs.getString("name"))); }
            return books;
        }
    }
} catch (SQLException e) {
    log.error("cannot select books", e);
}
```



# Závěr