

# Programovanie, algoritmy, zložitosť / ÚINF/PAZ1C



PAZ1C

Róbert Novotný  
robert.novotny@upjs.sk

15. 12. 2010

# Prístup k databázam

PAZ1C

- relačné databázy sú **dominantným** spôsobom ukladania dát
  - finančné, osobné, ...
- bohatý **výskum**
  - relačná algebra
- tona **implementácií**
  - komerčné: Oracle, DB2
  - open-source: MySQL, PostgreSQL



# Relačné databázy a Java

PAZ1C

- Java má v sebe zabudovaný spôsob pre prístup k relačným databázam
- špecifikácia **JDBC** – **Java Database Connectivity**
- priamo zabudované v JDK



# Čo potrebujeme

PAZ1C

- Na prípravu konektivity potrebujeme
- 1 ks bežiaceho databázového systému
  - Oracle, DB2, ...
- 1 kg JDBC ovládača
  - prostredník medzi Java kódom a databázovým systémom
- štipku kódu



# JDBC ovládač

PAZ1C

- **medzivrstva** medzi Java kódom a samotným databázovým systémom
- rieši **špinavú robotu**: pripájanie sa k serveru, nízkoúrovňový protokol
- typicky distribuovaný v podobe **JAR** súboru, ktorý sa zapojí do projektu
- každá významná implementácia poskytuje vlastný JDBC ovládač

# Problémy s JDBC

PAZ1C

- špecifikácia JDBC je síce funkčná, ale ťažkopádna
- mnoho opakujúceho sa kódu
- náchylné na chyby
- nutnosť korektne obsluhovať a uvoľňovať prostriedky
  - začiatčovníkom sa často ukazuje len jadro
  - vynecháva alebo zamlčiava sa správna obsluha chýb

Takmer každý rozsiahlejší projekt si vytvorí vlastnú nadstavbu nad JDBC.

# Spring Framework



- Rod Johnson zhrnul skúsenosti z vývoja enterprise aplikácií
- klasický spôsob: **J2EE**
  - sada špecifikácií pre komplexné problémy
    - webové aplikácie, transakcie, webové služby...
  - problém: „*tava je kôň, ktorého navrhla komisia.*“
- **Spring Framework**: iný prístup k problémom

# Klasický JDBC přístup

PAZ1C

- vytvorenie inštancie JDBC ovládača (raz)
- získanie objektu pre pripojenie (**Connection**)
- vytvorenie dopytu (**Statement**)
- vykonanie dopytu a získanie výsledku (**ResultSet**)
- spracovanie výsledku (iterácia cez **ResultSet**)
- uvoľnenie prostriedkov (**close()**)
- ošetrenie chýb (**SQLException**)

jediná premenlivá  
časť



# Prístup á la Spring

PAZ1C

- deklarácia dátového zdroja **DataSource** (raz)
- získanie objektu pre pripojenie (**Connection**)
- vytvorenie dopytu (**Statement**)
- vykonanie dopytu a získanie výsledku (**ResultSet**)
- spracovanie výsledku (iterácia cez **ResultSet**)
- uvoľnenie prostriedkov (**close()**)
- ošetrenie chýb (**SQLException**)

zelené časti  
spraví Spring!

# Optická redukcia kódu

PAZ1C

```
Connection con = null;
Statement statement = null;
ResultSet resultSet = null;

try {
    Class.forName("com.mysql.jdbc.Driver"); konfigurácia JDBC ovládača

    con = DriverManager.getConnection("jdbc:mysql://dbserver/mojadb", "db", "db"); získanie
    objektu typu Connection
    statement = con.createStatement(); vytvorenie Statement-u
    resultSet = statement.executeQuery("SELECT * FROM tabulka"); spustenie Statement-u
    while(resultSet.next()) { spracovanie výsledku
        System.out.print(resultSet.getInt("id"));
        System.out.print(resultSet.getString("stlpec"));
        System.out.println();
    }
} catch (Exception e) {
    System.out.println("Nastala chyba." + e.getMessage()); ošetrenie chýb
} finally { upratanie a uvoľnenie databázových prostriedkov
    if(resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException ee) {
            // nic rozumne uz nevieme spravit
        }
    }
    if(statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            // nic rozumne uz nevieme spravit
        }
    }
    if(con != null) {
        try {
            con.close();
        } catch (SQLException e) {
            // nic rozumne uz nevieme spravit
        }
    }
}
}
```

```
jdbcTemplate.update("DELETE FROM student WHERE id = " + s.getId());
```

```
<bean name="dataSource"
class="com.mysql.jdbc.jdbc2.optional.MysqlDataSource">
  <property name="user" value="${database.username}" />
  <property name="password"
value="${database.password}" />
  <property name="serverName" value="${database.host}" />
  <property name="databaseName" value="${database.schema}" />
</bean>

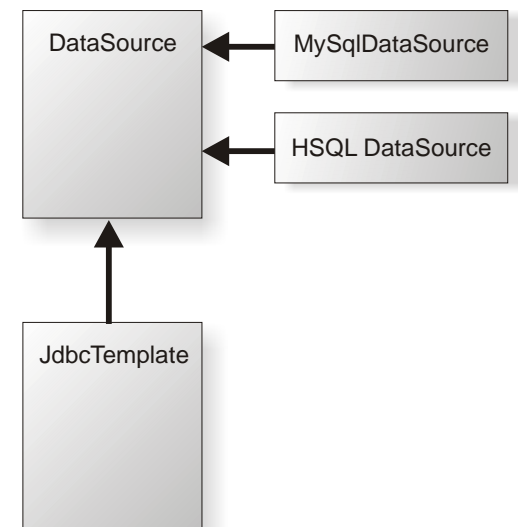
<bean name="beerDao" class="spring.dao.BeerDao">
  <property name="dataSource" ref="dataSource" />
</bean>
```

**Spring:**  
1 riadok v Java  
2 deklarácie beanov v XML

# Základné triedy

PAZ1C

- **javax.sql.DataSource** – dátový zdroj, získavame z neho objekty pripojení k databáze
- **JdbcTemplate** – trieda, ktorou môžeme spúšťať SQL dopyty
  - JdbcTemplate musí obsahovať referenciu na DataSource



# Továrneň na JdbcTemplate

PAZ1C

```
public class JdbcTemplateFactory {  
    public static JdbcTemplate JDBC_TEMPLATE;  
  
    static {  
        JDBCDataSource dataSource = new JDBCDataSource();  
        dataSource.setDatabase(  
            "jdbc:hsqldb:hsqldb://localhost/spring");  
        dataSource.setUser("sa");  
        dataSource.setPassword("");  
        jdbcTemplate = new JdbcTemplate(ds);  
    }  
}
```

použijeme  
implementáciu z  
HSQLDB

# Továrěň na JdbcTemplate

PAZ1C

```
public class JdbcTemplateFactory {
    public static JdbcTemplate JDBC_TEMPLATE;

    static {
        JDBCDataSource dataSource = new JDBCDataSource();
        dataSource.setDatabase(
            "jdbc:hsqldb:hsqldb://localhost/spring");
        dataSource.setUser("sa");
        dataSource.setPassword("");
        jdbcTemplate = new JdbcTemplate(ds);
    }
}
```

vytvoríme  
jdbcTemplate na  
základe dátového  
zdroja

# Továrěň na JdbcTemplate

PAZ1C

```
public class JdbcTemplateFactory {  
    public static JdbcTemplate JDBC_TEMPLATE;  
  
    static {  
        JDBCDataSource dataSource = new JDBCDataSource();  
        dataSource.setDatabase(  
            "jdbc:hsqldb:hsqldb://localhost/spring");  
        dataSource.setUser("sa");  
        dataSource.setPassword("");  
        jdbcTemplate = new JdbcTemplate(ds);  
    }  
}
```

ak máme  
výnimku, je to  
fatálna chyba

# Továrneň na JdbcTemplate

PAZ1C

```
public class JdbcTemplateFactory {  
    public static JdbcTemplate JDBC_TEMPLATE;  
  
    static {  
        JDBCDataSource dataSource = new JDBCDataSource();  
        dataSource.setDatabase(  
            "jdbc:hsqldb:hsqldb://localhost/spring");  
        dataSource.setUser("sa");  
        dataSource.setPassword("");  
        jdbcTemplate = new JdbcTemplate(ds);  
    }  
}
```

static blok sa  
vykoná raz pri  
vytváraní  
inštalácie triedy

# Spring a JDBC– šablóna

PAZ1C

- získanie celého čísla:

```
int rows =  
    jdbcTemplate.queryForInt("SELECT COUNT(*) FROM beer");
```

- pridávanie / aktualizácia dát

```
String sql = "UPDATE beer SET name = ?, countryCode = ?  
            WHERE id = ?"  
int rows = jdbcTemplate.update(sql,  
    beer.getName(),  
    beer.getCountryCode(),  
    beer.getId());
```



# Spring a JDBC – mapovanie objektov

PAZ1C

- priama podpora mapovania objektov podľa mennej konvencie

```
class Beer {  
    int id  
    String name;  
    String country;  
}
```

```
CREATE TABLE BEER  
    id INTEGER,  
    name VARCHAR(25);  
    country VARCHAR(25);  
)
```

# Spring a JDBC – mapovanie objektov

PAZ1C

- priama podpora mapovania beanov podľa mennej konvencie

```
BeanPropertyRowMapper<Book> mapper =  
    BeanPropertyRowMapper.newInstance(Beer.class);
```

- mapuje názvy stĺpcov na inštančné premenné
- automatická konverzia štandardných dátových typov
- inštanciu použijeme pri **query()**

# Spring a JDBC– šablóna

PAZ1C

- získanie objektu – potrebujeme mapovanie z riadkov tabuľky na inštancie

```
public class BeerMapper implements RowMapper<Beer> {  
    public Beer mapRow(ResultSet rs, int rowNum)  
        throws SQLException  
    {  
        Beer beer = new Beer();  
        beer.setId(rs.getInt("id"));  
        beer.setName(rs.getString("name"));  
        beer.setCountryCode(rs.getString("country"));  
  
        return beer;  
    }  
}
```

# Spring a JDBC– šablóna

PAZ1C

- získanie objektu – metóda **query()**

```
List<Beer> beers = jdbcTemplate.query(  
    "SELECT * FROM BEER",  
    mapper);
```

- pre každý riadok výsledku sa zavolá metóda **mapRow()**
- mapovač vráti inštanciu piva
- metóda **query()** vyzbiera všetky inštancie a vráti ich ako zoznam

# Jednoduchá šablóna

PAZ1C

- často možno vidieť aj použitie triedy **SimpleJdbcTemplate**
  - podporuje pomenované parametre, viacparametrové metódy...
- oproti klasickej **JdbcTemplate** vynecháva zriedkavejšie používané metódy
- v prípade potreby však vieme:

```
JdbcOperations jdbcTemplate  
    = simpleJdbcTemplate.getJdbcOperations();
```

# SimpleJdbcInsert

PAZ1C

- elegantné vkladanie objektov cez INSERT
- automaticky sa vybuduje INSERT do tabuľky BEER
- stĺpce sa prevezmú z *params*
- možno ich však prispôbiť cez **withColumnNames()**

```
SimpleJdbcInsert insert
    = new SimpleJdbcInsert(dataSource);
insert.setTableName("beer");
//...
SqlParameterSource params
    = new BeanPropertySqlParameterSource(beer);
insert.execute(params);
```

# SimpleJdbcInsert

PAZ1C

- v prípade INSERTov často vraciame vygenerovaný kľúč
- ak máme autogenerované primárne kľúče
- metóda **executeAndReturnKey()**

```
java.lang.Number key = insert.executeAndReturnKey(params);
```

- z **Number** si vieme vytiahnuť číselnú hodnotu podľa príslušného typu

# Inštalácia

PAZ1C

- **hsqldb.jar – databáza HSQL ([hsqldb.org](http://hsqldb.org))**
- **org.springframework**  
**(<http://www.springframework.com/download/community>)**
  - aop,
  - beans,
  - context,
  - context-support,
  - core,
  - jdbc,
  - transaction
- **commons-logging-1.1.1.jar**  
**(<http://commons.apache.org/downloads>)**



databaza.zip