JAVA FULL STACK DEVELOPMENT PROGRAM

Session 15: Spring

OUTLINE

- Spring Expression Language (SpEL)
- Useful Core Annotations
  - @Value
  - @PropertySource(s)
  - @Primary
- Spring Data Access

SPRING EXPRESSION LANGUAGE

- The Spring Expression Language (SpEL) is a powerful expression language that supports querying and manipulating an object graph at runtime.
  - Annotation
  - XML
- **Available operators**

Type	*Operators*
Arithmetic	+, -, *, /, %, ^, div, mod
Relational	<, >, ==, !=, <=, >=, lt, gt, eq, ne, le, ge
Logical	and, or, not, &&, ||, !
Conditional	?:

SPRING EXPRESSION LANGUAGE

- SpEL expressions begin with the *#* symbol, and are wrapped in braces: *#{expression}*.

- Properties can be referenced in a similar fashion, starting with a *$* symbol, and wrapped in braces: *${property.name}*.

- Property placeholders cannot contain SpEL expressions, but expressions can contain property references: #{*${someProperty} + 2*}

ARITHMETIC OPERATORS

```
@Value("#{19 + 1}")
private double add;

@Value("#{'String1 ' + 'string2'}")
private String addString;

@Value("#{20 - 1}")
private double subtract;

@Value("#{10 * 2}")
private double multiply;

@Value("#{36 / 2}")
private double divide;

@Value("#{36 div 2}")
private double divideAlphabetic;
```

```
@Value("#{37 % 10}")
private double modulo;

@Value("#{37 mod 10}")
private double moduloAlphabetic;

@Value("#{2 ^ 3}")
private double powerOf;

@Value("#{(2 + 2) * 2 + 9}")
private double brackets;
```

RELATIONAL AND LOGICAL OPERATORS

```
@Value("#{1 == 1}")
private boolean equal;

@Value("#{1 eq 1}")
private boolean equalAlphabetic;

@Value("#{1 != 1}")
private boolean notEqual;

@Value("#{1 ne 1}")
private boolean notEqualAlphabetic;

@Value("#{1 < 1}")
private boolean lessThan;

@Value("#{1 lt 1}")
private boolean lessThanAlphabetic;
```

```
@Value("#{1 <= 1}")
private boolean lessThanOrEqual;

@Value("#{1 le 1}")
private boolean lessThanOrEqualAlphabetic;

@Value("#{1 > 1}")
private boolean greaterThan;

@Value("#{1 gt 1}")
private boolean greaterThanAlphabetic;

@Value("#{1 >= 1}")
private boolean greaterThanOrEqual;

@Value("#{1 ge 1}")
private boolean greaterThanOrEqualAlphabetic;
```

REFERENCE PROPERTIES

- It is very common to define configuration values, such as database connection, error messages, in the property files.

- Annotations to access the property files:

  - @PropertySource

  - @Value

REFERENCE PROPERTIES

- Several types of properties
  - Single value
  - Default value
  - List value
  - Map value
  - Object value

@PRIMARY

- Sometimes we need to define multiple beans of the same type. In these cases, the injection will be unsuccessful because Spring has no clue which bean we need.

```
@Component
@Primary
class Car implements Vehicle {}

@Component
class Bike implements Vehicle {}
```

```
@Component
class Driver {
    @Autowired
    Vehicle vehicle;
}


@Component
class Biker {
    @Autowired
    @Qualifier("bike")
    Vehicle vehicle;
}
```

SPRING DATA ACCESS

- Spring Data Access
- DAO Support
- JdbcTemplate
- Hibernate Integration

DAO SUPPORT

- Spring is aimed at making it easy to work with data access technologies (such as JDBC, Hibernate, or JPA) in a consistent way.

- DAO Annotations

  - @Repository — This annotation also lets the component scanning support find and configure your DAOs and repositories without having to provide XML configuration entries for them

JDBC TEMPLATE

Action	Spring	You
Define connection parameters.		X
Open the connection.	X	
Specify the SQL statement.		X
Declare parameters and provide parameter values		X
Prepare and execute the statement.	X	
Set up the loop to iterate through the results (if any).	X	
Do the work for each iteration.		X
Process any exception.	X	
Handle transactions.	X	
Close the connection, the statement, and the resultset.	X	

CONNECTION CONFIGURATION

- Spring obtains a connection to the database through a DataSource
- A DataSource is part of the JDBC specification and is a generalized connection factory.
- It lets a container or a framework hide connection pooling and transaction management issues from the application code

```java
@Bean
public DataSource mysqlDataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc");
    dataSource.setUsername("guest_user");
    dataSource.setPassword("guest_password");

    return dataSource;
}
```

JDBC TEMPLATE

- Let's take a look at Spring JdbcTemplate Example
  - Single Object — *queryForObject*
  - List Object — *query*
  - RowMapper — *mapping to domain*
  - Named Parameters — *IN Clause*

JDBC TEMPLATE

- Instances of the JdbcTemplate class are thread-safe, once configured
  - It means that you can configure a single instance of a JdbcTemplate and then safely inject this shared reference into multiple DAOs (or repositories)
- If your application accesses multiple databases, you may want multiple JdbcTemplate instances, which requires multiple DataSources and, subsequently, multiple differently configured JdbcTemplate instances

HIBERNATE INTEGRATION

- Bootstrapping a SessionFactory with the native Hibernate API is a bit complicated and would take us quite a few lines of code

- Spring supports bootstrapping the SessionFactory – so that we only need a few lines of Java code or XML configuration

- Note: Hibernate Template is deprecated since Spring 3.0 and Hibernate 3.0.1 — We can use Hibernate API with Spring Transaction Management directly.

HIBERNATE INTEGRATION

- Let's take a look at the Hibernate-Spring example
- Hibernate configuration
- Abstraction

SPRING TRANSACTION MANAGEMENT

- Spring provides support for both programmatic and declarative transactions
- Programmatic Transactions
  - With programmatic transactions, transaction management code needs to be explicitly written so as to commit when everything is successful and rolling back if anything goes wrong. The transaction management code is tightly bound to the business logic in this case.
- Declarative Transactions
  - Declarative transactions separates transaction management code from the business logic. Spring supports declarative transactions using transaction advice (using AOP) via XML configuration in the spring context or with *@Transactional* annotation.

```xml
<beans>

    <bean id="myTxManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>

    <bean id="myProductService" class="product.ProductServiceImpl">
        <property name="transactionManager" ref="myTxManager"/>
        <property name="productDao" ref="myProductDao"/>
    </bean>

</beans>
```

```java
public class ProductServiceImpl implements ProductService {

    private TransactionTemplate transactionTemplate;
    private ProductDao productDao;

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
    }

    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    public void increasePriceOfAllProductsInCategory(final String category) {
        this.transactionTemplate.execute(new TransactionCallbackWithoutResult() {
            public void doInTransactionWithoutResult(TransactionStatus status) {
                List productsToChange = this.productDao.loadProductsByCategory(category);
                // do the price increase...
            }
        });
    }
}
```

DECLARATIVE TRANSACTIONS

- *@Transactional* can be either at the class or method level with further configuration
  - The *Propagation Type* of the transaction — that the same transaction will propagate from a transactional caller to transactional callee
    - e.g. if a read-only transaction calls a read-write transaction method, the whole transaction will be read-only
  - The *Isolation Level* of the transaction — Isolation level defines a contract between transactions.
  - A *Timeout* for th operation wrapped by the transaction
  - A *readOnly flag* – a hint for the persistence provider that the transaction should be read only
  - The *Rollback* rules for the transaction — by default, rollback happens for runtime, unchecked exceptions only

ANY QUESTIONS?