Spring and Hibernate

EHSDI

eBuzima

Spring and Hibernate

- Spring as a framework is designed to work with other libraries and frameworks
- Spring's data access framework has support for Hibernate, JDO, Oracle Toplink, etc

Hibernate configuration

- Normally when using Hibernate, we create a *hibernate.cfg* file and build a `SessionFactory` from that in our application
- In a Spring application, we use create suitable objects in the application context

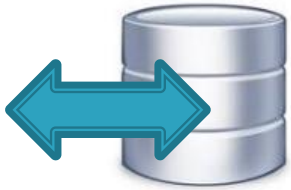
Data source

- This class creates a database connection
- It can work with any kind of SQL database

```
<bean id="exampleDataSource"
    class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost/exampledb" />
  <property name="username" value="root" />
  <property name="password" value="XXXXXX" />
</bean>
```

Parameters that would have been in *hibernate.cfg*

Session factory



- A `Session` is an object which allows us to interact with a database, execute SQL etc
- A `SessionFactory` is an object which generates sessions, based on a data source

```
<bean id="exampleSessionFactory"
   class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="exampleDataSource" />
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
    </props>
  </property>
</bean>
```

Hibernate specific properties, that would have been in *hibernate.cfg*

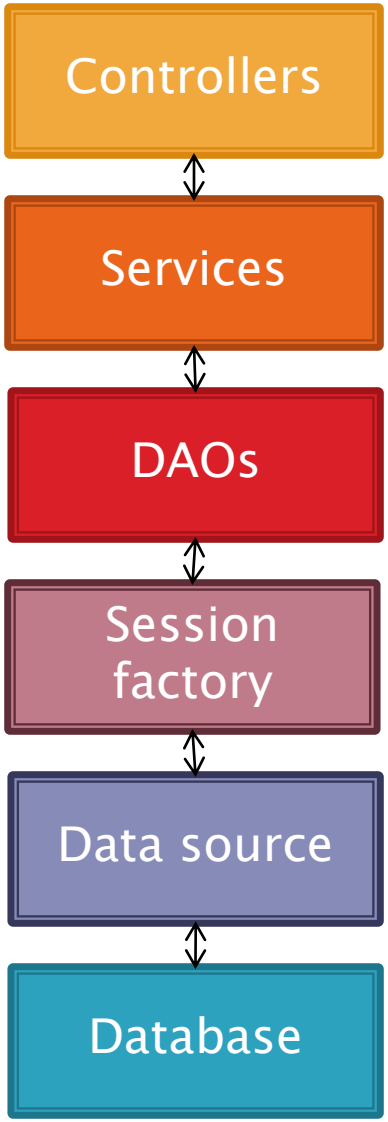
Configures this factory to use the data source we created

Mapping files

- These are specified in the session factory bean, e.g.

```
<bean id="exampleSessionFactory"
   class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="exampleDataSource" />
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
    </props>
  </property>
  <property name="mappingResources">
    <list>
      <value>example/Person.hbm.xml</value>
      <value>example/Job.hbm.xml</value>
    </list>
  </property>
</bean>
```

Application architecture



Generate models for views, **not dependant on any database technology**

Provides functionality to the controllers, **not dependant on any database technology**

Provides data access to services, dependant only on a data access library such as Hibernate

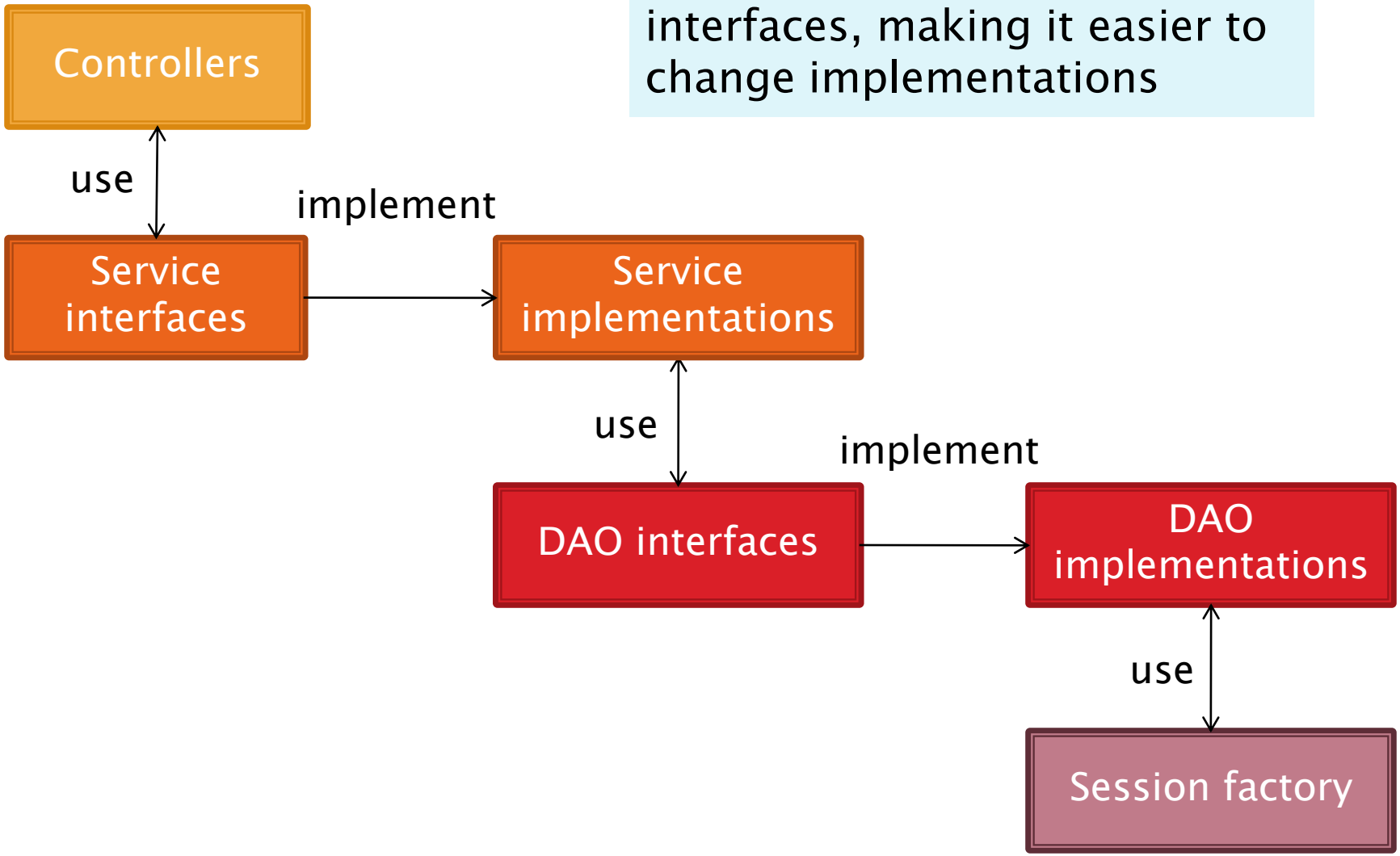
A data access library such as Hibernate or JDO

A generic SQL database connection

A database such as MySQL, Oracle, etc

Interfaces

The different layers interact with each other through interfaces, making it easier to change implementations



In OpenMRS...

Layer	Package / JAR	Example
Services	org.openmrs.api	PatientService
	org.openmrs.api.impl	PatientServiceImpl
DAOs	org.openmrs.db	PatientDAO
	org.openmrs.db.hibernate	HibernatePatientDAO
Hibernate	hibernateXX.jar	
JDBC	java.sql	
MySQL	mysql-connector-java-XX.jar	

Example: DAO interface

- The DAO interface contains all the methods we need to interact with the database, e.g.

```
public interface ExampleDAO {
  /**
   * Gets all the examples from the database
   * @return list of examples
   */
  public List<Example> getExamples();

  /**
   * Saves an example to the database
   * @param example the example
   */
  public void saveExample(Example example);
}
```

Example: DAO implementation

- We implement it using a database library such as Hibernate
- We need to first give it access to the session factory in the application context…

```
public class HibernateExampleDAO implements ExampleDAO {
  private SessionFactory sessionFactory;

  /**
   * Sets the session factory
   */
  public void setSessionFactory(SessionFactory sf) {
    this.sessionFactory = sf;
  }
  ...
}
```

Becomes a settable bean property

Example: DAO implementation

- It's then added to the application context as a bean..

```
<bean id="exampleDAO"
    class="example.db.HibernateExampleDAO">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Calls `setSessionFactory` method on the bean

The existing session factory bean

Example: DAO implementation

- And the we can implement the DAO methods using the Hibernate session factory…

```
public class HibernateExampleDAO implements ExampleDAO {
  private SessionFactory sessionFactory;
  ...

  public List<Example> getExamples() {
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    List<Example> examples =
      session.createCriteria(Example.class).list();
    session.getTransaction().commit();
    session.close();
    return examples;
  }
}
```

Example: DAO vs Service

- We now have a working DAO, which we could use in our program, e.g.

```
ExampleDAO dao = (ExampleDAO)appContext.getBean("exampleDAO");
List<Example> examples = dao.getExamples();
```

- But in Spring's application architecture we shouldn't access DAO's directly in our programs
  - Makes it harder to change database technologies
  - Mixes database code and business logic

Example: Service interface

- Service may have the same methods as the DAO, but not necessarily

```
public interface ExampleService {
  /**
   * Gets all the examples
   * @return list of examples
   */
  public List<Example> getExamples();

  /**
   * Saves an example
   * @param example the example
   */
  public void saveExample(Example example);
}
```

Example: Service implementation

- We implement the service's methods using a DAO
- Need to give the service access to the DAO in the application context…

```
public class ExampleServiceImpl implements ExampleService {
  private ExampleDAO exampleDAO;

  /**
   * Sets the example DAO
   */
  public void setExampleDAO(ExampleDAO dao) {
    this.exampleDAO = dao;
  }
  ...
}
```

Becomes a settable bean property

Example: Service implementation

- It's then added to the application context as a bean..

```
<bean id="exampleService"
    class="example.ExampleServiceImpl">
  <property name="exampleDAO" ref="exampleDAO" />
</bean>
```

Calls `setExampleDAO` method on the bean

The existing DAO bean

Example: Service implementation

- And the we can implement the service methods using the DAO…

```
public class ExampleServiceImpl implements ExampleService {
  private ExampleDAO exampleDAO;
  ...

  public List<Example> getExamples() {
    return exampleDAO.getExamples();
  }

  public void saveExample(Example example) {
    exampleDAO.saveExample(example);
  }
}
```

Example: Using the service

- We now have a working service, which we can use in our program, e.g.

```
ExampleService svc =
  (ExampleService)appContext.getBean("exampleService");

List<Example> examples = svc.getExamples();
```

- The service provides functionality to the program in a *database independent* manner

Transactions

- All of the DAO methods need valid sessions and transactions
- This code has to be in every method

```
public List<Example> getExamples() {
  Session session = sessionFactory.openSession();
  session.beginTransaction();
  List<Example> examples =
    session.createCriteria(Example.class).list();
  session.getTransaction().commit();
  session.close();
  return examples;
}
```

Transaction management

- Spring provides the `@Transactional` annotation which automatically manages our transactions, e.g.

Get session and create transaction

```
@Transactional
public List<Example> getExamples() {
  return session.createCriteria(Example.class).list();
}
```

Commit transaction

Transaction management

- To enable transaction management we need to add a suitable `TransactionManager` bean

Tells it to manage our existing session factory

```
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```

Tells Spring to use the `@Transactional` annotation to control it

@Transactional

- Placing the annotation on the service layer is considered "best practice"
  - Means that a service method can execute several DAO calls inside one transaction

```
public interface ExampleService {
  @Transactional
  public List<Example> getExamples();

  @Transactional
  public void saveExample(Example example);
}
```

@Transactional

- Placing the annotation on the class rather than the methods, automatically applies it to all methods in that class, e.g.

```
@Transactional
public interface ExampleService {

  public List<Example> getExamples();

  public void saveExample(Example example);
}
```

Read-only transactions

- If Hibernate knows that a transaction won't make any changes to the database, it can optimize that transaction to make it faster
- We can use `readOnly` to mark a method's transaction as read-only, e.g.

```
@Transactional
public interface ExampleService {

  @Transactional(readOnly=true)
  public List<Example> getExamples();

  public void saveExample(Example example);
}
```

References

- http://static.springsource.org/spring/docs/2.5.x/reference/spring-middle-tier.html