CS544 EA

Applications

# SH Web Apps: Combining Spring and Hibernate

# Spring and Hibernate-JPA

- Spring can **fully configure and start Hibernate**
  - Removing the need for persisntence.xml
  - Makes EntityManagerFactory Spring Bean (singleton)
  - Gives ThreadLocal functionality for EntityManager
  - Also provides OpenEntityManagerInView filter
    - Which integrates nicely with Spring TX management

# Spring JPA Config XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans">

      <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
            <property name="driverClassName"
                  value="com.mysql.jdbc.Driver" />
            <property name="url" value="jdbc:mysql://localhost/cs544" />
            <property name="username" value="root" />
            <property name="password" value="root" />
      </bean>

      <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
            <property name="dataSource" ref="dataSource" />
            <property name="jpaVendorAdapter">
                  <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                        <property name="generateDdl" value="true" />
                        <property name="database" value="MYSQL" />
                  </bean>
            </property>
            <property name="jpaProperties">
                  <props>
                        <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
                        <prop key="hibernate.show_sql">true</prop>
                        <prop key="hibernate.format_sql">true</prop>
                        <prop key="hibernate.id.new_generator_mappings">false</prop>
                        <prop key="javax.persistence.schema-generation.database.action">drop-and-create</prop>
                  </props>
            </property>
            <property name="packagesToScan" value="cs544" />
      </bean>
      ...
```

# Spring JPA Config Java

```
@Configuration
@ComponentScan("cs544")
public class Config {
      @Bean
      public DataSource dataSource() {
            DriverManagerDataSource dataSource = new DriverManagerDataSource();
            dataSource.setDriverClassName("com.mysql.jdbc.Driver");
            dataSource.setUsername("root");
            dataSource.setPassword("root");
            dataSource.setUrl("jdbc:mysql://localhost/cs544");
            return dataSource;
      }

      @Bean
      public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
            LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();
            emf.setDataSource(dataSource());
            emf.setPackagesToScan("cs544");

            Properties properties = new Properties();
            properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");
            properties.setProperty("hibernate.id.new_generator_mappings", "false");
            properties.setProperty("hibernate.show_sql", "true");
            properties.setProperty("hibernate.hbm2ddl.auto", "create-drop");

            JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
            emf.setJpaVendorAdapter(vendorAdapter);
            emf.setJpaProperties(properties);
            return emf;
      }
}
```

# Example from DB to Web

```
@Entity
public class Customer {
     @Id
     @GeneratedValue
     private Long id;
     private String name;

     public Long getId() {
          return id;
     }

     public void setId(Long id) {
          this.id = id;
     }

     public String getName() {
          return name;
     }

     public void setName(String name) {
          this.name = name;
     }
}
```

Using either the web.xml or WebApplicationInitializer shown earlier

Import.sql

```
INSERT INTO Customer VALUES(NULL, "James Reagon");
INSERT INTO Customer VALUES(NULL, "Lilly Johnson");
INSERT INTO Customer VALUES(NULL, "George Tall");
```

## Example DAO

```
@Repository
public class CustomerDao {
	@PersistenceContext
	private EntityManager em;

	public List<Customer> getAll() {
		return em.createQuery("from Customer", Customer.class).getResultList();
	}
}
```

## Example Service

```
@Service
public class CustomerService {
	@Resource
	private CustomerDao customerDao;

	public List<Customer> getCustomers() {
		return customerDao.getAll();
	}
}
```

Cannot do BMT, throws exception that you should use Spring TX (CMT).

We'll add these in the next section (for now Transaction Per Operation!)

# Example Controller

```
@WebServlet(name = "Customers", urlPatterns = { "/customers" })
public class Customers extends HttpServlet {
     private static final long serialVersionUID = 1L;

     @Override
     protected void doGet(HttpServletRequest request, HttpServletResponse response)
               throws ServletException, IOException {

         ServletContext context = getServletContext();
         WebApplicationContext applicationContext =
             WebApplicationContextUtils.getWebApplicationContext(context);
         CustomerService custServ = applicationContext.getBean(
             "customerService", CustomerService.class);


          request.setAttribute("customers", custServ.getCustomers());
          String jsp = "/Customers.jsp";
          RequestDispatcher dispatcher = context.getRequestDispatcher(jsp);
          dispatcher.forward(request, response);
     }
}
```

# Example JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Customers</title>
</head>
<body>
     <h1>Customers:</h1>
     <ul>
          <c:forEach items="${customers}" var="customer">
               <li>${customer.name}</li>
          </c:forEach>
     </ul>
</body>
</html>
```