infogain

SPRING MVC

Objectives

- In this session, you will learn
- Spring MVC form validation
- Spring MVC Internalization and Localization
- Spring ORM

MVC Form Validation

- Hibernate Bean Validator Framework uses annotations to apply validation rules on properties of a bean.
- you will need the validation-api-1.1.0.Final.jar and hibernate-validator-5.0.1.Final.jar files in order to use the Bean Validation API in your Spring MVC application
- Use @Valid annotation and BindingResult class through which we can get the errors raised by Validator implemented in the controller request handler method.

MVC Form Validation(contd.)

- The Bean Validation API allows to declare validation constraints on object models via annotations.
  - public class Employee {
  - @NotEmpty
  - @Email
  - private String email;
  - @NotEmpty
  - @Size(min = 6, max = 15)
  - private String password;
  - // getters and setters
  - }

Configuring Support For MVC Form Validation

- Spring MVC provides full support for the Bean Validation with minimal configuration.
- Add the following entry to Spring's application context XML file:

```
<mvc:annotation-driven />
```

- Spring MVC will detect and enable the validation support automatically.
- Now in the controller class, annotate the model object that is backing the form by the @Valid annotation (javax.validation.Valid):

Spring Controller Class

infogain

```
@Controller
public class LoginController {
   @RequestMapping(value = "/login", method =RequestMethod.POST)
   public String doLogin(@Valid  Employee emp, BindingResult result) {
      // login logic here    }}
```

Spring validate the model object annotated by the @Valid annotation after binding its properties with inputs from JSP page. Any constraint violations will be exposed as errors in the BindingResult object, thus we can check the violation in the controller's method like this:

```
if (result.hasErrors()) {
   // form validation error
} else {
   // form input is ok }
```

Spring Error Message

- We would return the input form back to the user when any validation errors occurred. And in the JSP form, we can show validation error messages using the Spring's form errors tag as follows:
- <form:errors path="email" />
- The error message can be specified in the validation annotation, for example:
- @NotEmpty(message = "Please enter your email addresss.")
- private String email;
- If you want to localize the message, specify a key in the properties file in the following convention:
- ConstraintName.CommandName.propertyName=validation error message
- E.g
- NotEmpty.userForm.email=Please enter your e-mail.

Hibernate Validators to validate bean properties

- **@NotEmpty** : Property can not be null or empty.
- **@Length** : Used to define min and max length of a string.
- **@NumberFormat** : Can define a particular number format or style to String.
- **@Min** : Used to apply constraint on minimum value of number.
- **@Max** : Used to apply constraint on maximum value of number.
- **@Email :** used to validate Email Id

Spring MVC Internationalization & Localization

- **Internationalization**
  - It is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes.
- **Localization**
  - It is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text.

- The terms are frequently abbreviated as i18n (where 18 stands for the number of letters between the first i and last n in internationalization) and L10n respectively, due to the length of the words.

Configure i18n & l10n to Spring MVC

- Declare these files in spring configuration file.
- **ReloadableResourceBundleMessageSource class**
  - org.springframework.context.support.ReloadableResourceBundleMessageSource
  - This class defines the message resources.
- **LocaleChangeInterceptor class**
  - org.springframework.web.servlet.i18n.LocaleChangeInterceptor
  - This interceptor is configured to intercept the user request and identify the user locale. It intercept any changes in the locale. These changes are then saved in cookies for future request.
- **CookieLocaleResolver class**
  - org.springframework.web.servlet.i18n.CookieLocaleResolver
  - This class will be used to set a cookie in the client request.It store the locale changes in cookies.

Spring Hibernate Integration

- **DriverManagerDataSource**
  - Used to contain the information about the database such as driver class name, connection URL, username and password.

Implementing ORM (Contd.)

- **Configuring the SessionFactory object in spring:**
  - Spring helps you create a Hibernate session factory by providing several factory beans in the Spring container.
- To configure a session factory object in Spring:
  - Declare a session factory by using the org.springframework.orm.hibernate3.LocalSessionFactoryBean class.
  - Or using org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean
  - Provide the path of the Hibernate configuration file

Implementing ORM (Contd.)

- **LocalSessionFactoryBean/AnnotationSessionFactoryBean**
  - It is a spring factory bean that creates hibernate sessionfactory.
  - The main purpose of this class is to set up the Hibernate SessionFactory in a spring context.
  - The hibernate configuration properties can be passed within the XML.
  - The configuration properties include the hibernate mapping resources, hibernate properties and a datasource.

Implementing Transactions

- Transactions:
  - Are not part of the primary business logic of the application. They just help in maintaining data integrity.
  - Are considered as cross-cutting concerns that cut across various modules of the application.
  - Are considered and implemented as aspects.

Implementing Transactions (Contd.)

- To implement the transactions, you need to select a transaction manager to implement the transaction specific operations, such as:
  - Committing or rolling-back a transaction.
  - Configuring a transaction.

Defining a Transaction Manager (Contd.)

- **The transaction manager:**
  - Is an important component of a transaction processing environment.
  - Is responsible for:
  - Creating transactions, as and when requested by application components.
  - Establishing and maintaining transaction context.
  - Maintaining association between a transaction and the resources taking part in a transaction.

Defining a Transaction Manager (Contd.)

- These transaction managers provide a set of technology-independent methods for managing transactions.
- This makes the transaction management code independent of any specific data access technology.
- When you change the data access technology:
  - You just need to change the declaration of the transaction manager in the Spring application context file.
  - The code to manage the transactions remains the same.

Defining a Transaction Manager (Contd.)

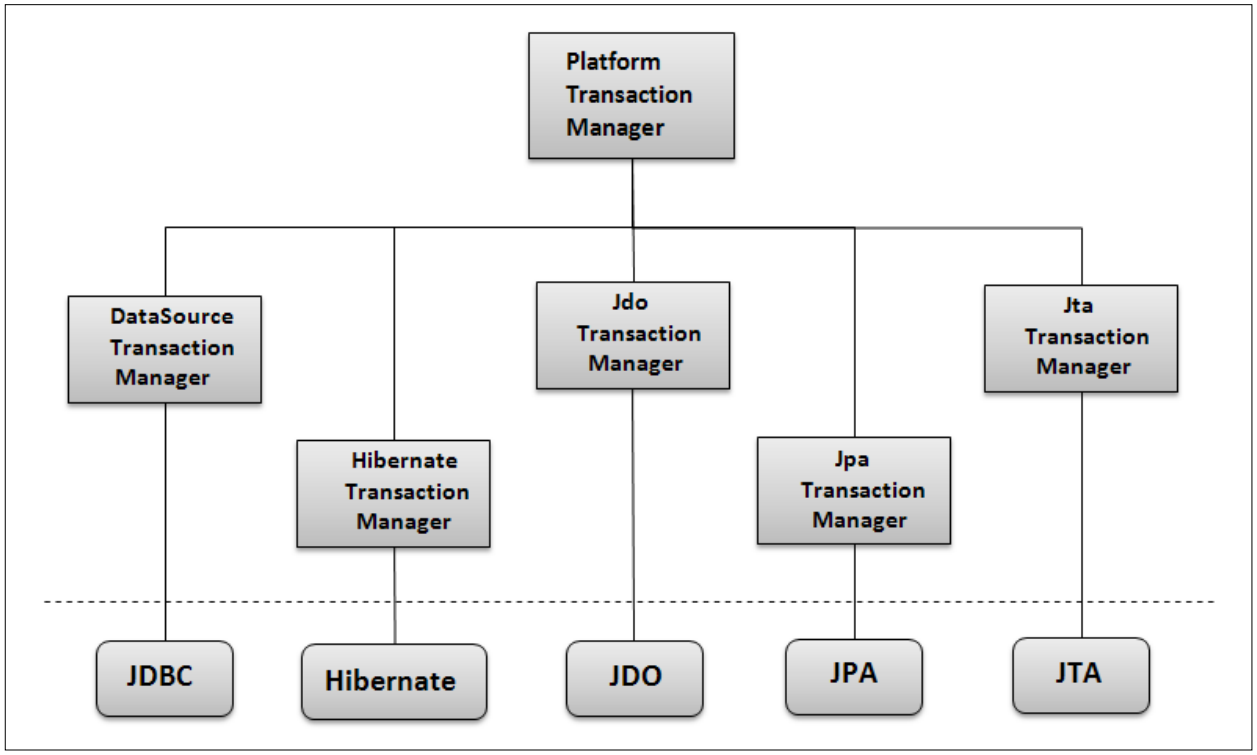
- **The PlatformTransactionManager interface:**
- Encapsulates the core functionality of transaction management.
- Provides the following methods for working with transactions:
  - getTransaction()
  - commit()
  - rollback()

Defining a Transaction Manager (Contd.)

- Spring provides the following built-in implementations of the **PlatformTransactionManager** interface for use with different transaction management APIs:
  - DataSourceTransactionManager
  - JtaTransactionManager
  - HibernateTransactionManager
  - JpaTransactionManager
  - JdoTransactionManager

Defining a Transaction Manager (Contd.)

- The following figure shows implementation of the PlatformTransactionManager interface for managing transactions.



Defining a Transaction Manager (Contd.)

- To use a transaction manager, you need to declare it in the configuration file as a bean.
- For example:

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

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

Configuring Transactions (Contd.)

- **Using annotations**
- Spring uses the following elements to support the annotation-driven transaction management:
  - The @Transactional element of the org.springframework.transaction.annotation.Transactional class
  - The <tx:annotation-driven> element has a transaction-manager attribute.
  - This attribute specifies reference to an existing PlatformTransactionManager bean that will be called to implement the transactions.

Configuring Transactions (Contd.)

- To declare a method or a class as transactional, simply annotate it with the @Transactional annotation.
- If a class is declared as transactional, then all the public methods of that class will be defined as transactional.
- Spring supports Transaction using AOP

infogain

Thank You