Вопросы и ответы на собеседование по Spring

(31-45)

31). Как проверить (валидировать) данные формы в Spring Web MVC Framework?

Spring поддерживает аннотации валидации из JSR-303, а так же возможность создания своих реализаций классов валидаторов. Пример использования аннотаций:

```
@Size(min=2, max=30)
private String name;

@NotEmpty @Email
private String email;

@NotNull @Min(18) @Max(100)
private Integer age;

@NotNull
private Gender gender;

@DateTimeFormat(pattern="MM/dd/yyyy")
@NotNull @Past
private Date birthday;
```

32). Что вы знаете о Spring MVC Interceptor и как он используется?

Перехватчики в Spring (**Spring Interceptor**) являются аналогом **Servlet Filter** и позволяют перехватывать запросы клиента и обрабатывать их. Перехватить запрос клиента можно в трех местах: preHandle, postHandle и afterCompletion.

- **preHandle** — метод используется для обработки запросов, которые еще не были переданы в метода обработчик контроллера. Должен вернуть **true** для передачи следующему перехватчику или в handler method. **False** укажет на обработку запроса самим обработчиком и отсутствию необходимости передавать его дальше. Метод имеет возможность выкидывать исключения и пересылать ошибки к представлению.
- **postHandle** — вызывается после handler method, но до обработки **DispatcherServlet** для передачи представлению. Может использоваться для добавления параметров в объект **ModelAndView**.
- **afterCompletion** — вызывается после отрисовки представления.

Для создания обработчика необходимо расширить абстрактный класс HandlerInterceptorAdapter или реализовать интерфейс HandlerInterceptor. Так же нужно указать перехватчики в конфигурационном файле Spring.

```
<!-- Configuring interceptors based on URI -->
   <interceptors>
      <interceptor>
         <mapping path="/home" />
         <beans:bean class="ru.readandwritecode.spring.RequestProcessingTimeInterceptor"></beans:bean>
      </interceptor>
   </interceptors>
```

33). Spring JdbcTemplate класс и его применение

Spring предоставляет отличную поддержку JDBC API и предлагает класс утилиту JdbcTemplate, с помощью которого можно избавиться от многократного повторения похожего кода в приложении (вроде операций **open \ closing connection**; **ResultSet**, **PreparedStatement** и др.). Для подключения необходимо настроить файл конфигурации spring и получить объект **JdbcTemplate**. Например.
spring.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="employeeDAO" class="com.readandwritecode.spring.jdbc.dao.EmployeeDAOImpl">
      <property name="dataSource" ref="dataSource" />
   </bean>

   <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

      <property name="driverClassName" value="com.mysql.jdbc.Driver" />
      <property name="url" value="jdbc:mysql://localhost:3306/TestDB" />
      <property name="username" value="rwc" />
      <property name="password" value="rwc123" />
   </bean>

</beans>
```

33). Продолжение

Пример использования **JdbcTemplate**:

```
@Override
  public void save(Employee employee) {
     String query = "insert into Employee (id, name, role) values (?,?,?)";

     JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

     Object[] args = new Object[] {employee.getId(), employee.getName(),employee.getRole()};

     int out = jdbcTemplate.update(query, args);

     if(out !=0){
        System.out.println("Employee saved with id="+employee.getId());
     }else System.out.println("Employee save failed with id="+employee.getId());
  }
```

34). Как использовать Tomcat JNDI DataSource в веб-приложении Spring?

Для использования контейнера сервлетов настроенного на использование **JNDI DataSource**, необходимо задать соответствующее свойство в файле конфигурации и затем **внедрять его как зависимость**. Далее мы можем использовать объект **JdbcTemplate** для выполнения операций с базами данных.

```
<beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>
</beans:bean>
```

35). Каким образом можно управлять транзакциями в Spring?

Транзакциями в Spring управляют с помощью Declarative Transaction Management (программное управление). Используется аннотация **@Transactional** для описания необходимости управления транзакцией. В файле конфигурации нужно добавить настройку **transactionManager** для DataSource.

```
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

36). Расскажите о Spring DAO

Spring DAO предоставляет возможность работы с доступом к данным с помощью технологий вроде JDBC, Hibernate в удобном виде. Существуют специальные классы: **JdbcDaoSupport**, **HibernateDaoSupport**, **JdoDaoSupport**, **JpaDaoSupport**.
В Spring DAO поддерживается иерархия исключений, что помогает не обрабатывать некоторые исключения.

```
import java.util.List;
import org.springframework.jdbc.core.support.JdbcDaoSupport;

public class PersonDao extends JdbcDaoSupport{

public void insert(Person person){
String insertSql ="INSERT INTO PERSON (NAME, EMAIL) VALUES(?,?);";
String name = person.getName();
String email = person.getEmail();

getJdbcTemplate().update(insertSql,new Object[]{name,email});
}

public List<Person> selectAll(){
String selectAllSql = "SELECT * FROM PERSON;";

return getJdbcTemplate().query(selectAllSql, new PersonRowMapper());
}

}
```

37). Как внедрить java.util.Properties в Spring Bean?

Для возможности использования Spring для внедрения свойств (properties) в различные бины необходимо определить propertyConfigure bean, который будет загружать файл свойств.

```
<bean id="propertyConfigurer"
  class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">
    <property name="location" value="/WEB-INF/application.properties" />
</bean>

<bean class="com.readandwritecode.spring.EmployeeDaoImpl">
    <property name="maxReadResults" value="${results.read.max}"/>
</bean>
```

Или через аннотации:

```
@Value("${maxReadResults}")
private int maxReadResults;
```

38). Как задаются файлы маппинга Hibernate в Spring?

Через applicationContext.xml в web/WEB-INF. Например:

```
<property name="mappingResources">
 <list>
  <value>com/example/domain/Entity1.hbm.xml</value>
 list>
property>
```

39). Как добавить поддержку Spring в web-приложение

Достаточно просто указать ContextLoaderListener в web.xml файле приложения:

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListenerlistener-class>
listener>
```

40). Можно ли использовать xyz.xml вместо applicationContext.xml?

ContextLoaderListener – это ServletContextListener, который инициализируется когда ваше web-приложение стартует. По-умолчанию оно загружает файл WEB-INF/applicationContext.xml. Вы можете изменить значение по-умолчанию, указав параметр **contextConfigLocation**.

Пример:

```
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener
      <context-param>
         <param-name>contextConfigLocation</param-name>
         <param-value>/WEB-INF/xyz.xml</param-value>
      context-param>
   listener-class>
listener>
```

41). Как настраивается соединение с БД в Spring?

Используя datasource “org.springframework.jdbc.datasource.DriverManagerDataSource”.

Пример:

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
   <property name="url" value="jdbc:mysql://localhost/mydb" />
   <property name="username" value="myuser" />
   <property name="password" value="mypassword" />
</bean>
```

42). Назовите некоторые из шаблонов проектирования, используемых в Spring Framework?

Spring Framework использует множество шаблонов проектирования, например:

1. Singleton Pattern
2. Factory Pattern
3. Prototype Pattern
4. Adapter Pattern
5. Proxy Pattern
6. Template Method Pattern
7. Front Controller
8. Data Access Object
9. Dependency Injection и Aspect Oriented Programming

43). Для чего нужен @ComponentScan?

Если вы понимаете как работает Component Scan, то вы понимаете Spring

Первый шаг для описания Spring Beans это добавление аннотации — @Component, или @Service, или @Repository.

Однако, Spring ничего не знает об этих бинах, если он не знает где искать их. То, что скажет Spring где искать эти бины и называется Component Scan. В **@ComponentScan** вы указываете пакеты, которые должны сканироваться.

Spring будет искать бины не только в пакетах для сканирования, но и в их подпакетах.

Read & Write Code

44). Как вы добавите Component Scan в Spring Boot?

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

@SpringBootApplication определяет автоматическое сканирование пакета, где находится класс Application

Всё будет в порядке, ваш код целиком находится в указанном пакете или его подпакетах.

Однако, если необходимый вам компонент находится в другом пакете, вы должны использовать дополнительно аннотацию **@ComponentScan**, где перечислите все дополнительные пакеты для сканирования.

45). **В чём отличие между** @Component и @ComponentScan?

@Component и @ComponentScan предназначены для разных целей.

@Component помечает класс в качестве кандидата для создания Spring бина.
@ComponentScan указывает где Spring искать классы, помеченные аннотацией @Component или его производной.