```
Top 50 câu hỏi phỏng vấn Spring Framework
```

1. Spring Framework là gì?

- Spring là một framework mạnh mẽ cho Java EE, hỗ trợ **Dependency Injection (DI)** và **Aspect Oriented Programming (AOP)**.

Ví dụ:

```
@Component
class Engine {}

@Component
class Car {
  @Autowired
  Engine engine;
}
```

2. Tính năng và lợi thế của Spring Framework?

Tính năng nổi bật:

- Nhẹ, dễ tích hợp
- Quản lý Bean qua DI
- Hỗ trợ AOP, MVC, JDBC, ORM

Lợi thế:

- Tách biệt thành phần => dễ test
- Giảm code dư thừa (boilerplate)
- Module hóa rõ ràng, dễ mở rộng

3. Dependency Injection (DI) là gì?

DI giúp **tách biệt phụ thuộc** giữa các class, để việc khởi tạo đối tượng do framework quản lý thay vì lập trình viên.

Ví dụ:

```
class Service {
  Repository repo;
  Service(Repository repo) {
    this.repo = repo;
```

```
  }
}
```

4. Làm sao để triển khai DI trong Spring?

Có 2 cách:

- Cấu hình XML:

```
<bean id="car" class="Car">
  <property name="engine" ref="engine"/>
</bean>
```

- Annotation:

```
@Autowired
private Engine engine;
```

5. Tính năng mới của Spring 5?

- Yêu cầu Java 8+
- Hỗ trợ **WebFlux (reactive programming)**
- Hỗ trợ **Kotlin**
- Tích hợp JUnit 5
- Sử dụng `spring-jcl` thay vì Commons Logging

6. Spring WebFlux là gì?

WebFlux là **mô-đun mới trong Spring 5** để xây dựng ứng dụng **asynchronous** và **non-blocking** theo kiểu **reactive**.

Ví dụ:

```
@GetMapping("/hello")
public Mono<String> hello() {
  return Mono.just("Hello Reactive World!");
}
```

7. Lợi ích của Spring Tool Suite (STS)?

- Cài sẵn plugin Spring
- Hỗ trợ AOP rõ ràng, debug dễ
- Tích hợp Maven, Tomcat, Templates

STS giúp phát triển Spring nhanh và trực quan hơn.

8. Các module quan trọng trong Spring?

- `Spring Context` - quản lý DI
- `Spring AOP` - hỗ trợ AOP
- `Spring JDBC` - thao tác CSDL
- `Spring MVC` - web app
- `Spring ORM` - tích hợp Hibernate

9. Lập trình hướng khía cạnh (AOP) là gì?

AOP giúp tách các logic "xuyên suốt" như logging, transaction, authentication ra khỏi business logic.

Ví dụ:

```
@Aspect
class LoggingAspect {
  @Before("execution(* com.app.service.*.*(..))")
  public void logBefore() {
    System.out.println("Gọi hàm rồi nè!");
  }
}
```

10. Aspect, Advice, Pointcut, JoinPoint là gì?

Thành phần	Giải thích ngắn
Aspect	Class chứa logic AOP (logging, auth, etc.)
Advice	Code thực thi tại JoinPoint
Pointcut	Biểu thức chọn JoinPoint
JoinPoint	Điểm thực thi cụ thể (method call, etc.)
Advice Args	Đối số truyền vào Advice nếu cần

Ví dụ:

```
@Before("execution(* transfer(..))")
public void logBefore(JoinPoint joinPoint) {
  System.out.println("Before: " + joinPoint.getSignature().getName());
}
```

11. Sự khác biệt giữa Spring AOP và AspectJ AOP là gì?

- **AspectJ** là triển khai AOP đầy đủ tính năng, hỗ trợ tất cả các loại joinpoint (method, constructor, field...).
- **Spring AOP** chỉ hỗ trợ AOP dựa trên proxy => chỉ intercept được các method.
- **Spring AOP** dễ sử dụng hơn vì không cần xử lý quá trình "weaving" phức tạp như AspectJ.
- **Spring AOP** có thể dùng các annotation của AspectJ như `@Aspect`, `@Before`, `@After`...
- **Hạn chế**: Spring AOP chỉ áp dụng cho các bean được Spring quản lý.

Ví dụ sử dụng Spring AOP:

```
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("Calling method: " +
joinPoint.getSignature().getName());
    }
}
```

12. Spring IoC Container là gì?

- **IoC (Inversion of Control)** là nguyên lý trong đó quyền kiểm soát việc khởi tạo và quản lý dependency được trao cho container.
- **Spring IoC Container** chịu trách nhiệm tạo, cấu hình, quản lý vòng đời của các đối tượng (bean).

Một số loại ApplicationContext:

- `AnnotationConfigApplicationContext`
- `ClassPathXmlApplicationContext`
- `FileSystemXmlApplicationContext`
- `AnnotationConfigWebApplicationContext`, `XmlWebApplicationContext`

Ví dụ:

```
ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
MyBean bean = context.getBean(MyBean.class);
```

---

13. Spring Bean là gì?

- Là bất kỳ object nào được quản lý bởi Spring IoC Container.
- Spring bean có thể được inject các dependency và được quản lý vòng đời tự động.

Ví dụ định nghĩa bean trong XML:

```xml
<bean id="myService" class="com.example.MyService" />
```

---

14. Tầm quan trọng của file cấu hình Spring Bean

- Dùng để định nghĩa tất cả các bean và các thành phần khác như Interceptors, ViewResolvers...
- Khi ApplicationContext được tạo, nó đọc file config để khởi tạo các bean.

Ví dụ:

```xml
<context:component-scan base-package="com.example" />
<bean id="myService" class="com.example.MyService" />
```

---

15. Các cách khác nhau để cấu hình một lớp thành Spring Bean

1. **Cấu hình XML**:

```xml
<bean name="myBean" class="com.example.MyBean" />
```

2. **Cấu hình Java với @Configuration và @Bean**:

```java
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

3. **Cấu hình dựa trên annotation (@Component, @Service, @Repository, @Controller)**:

```java
@Component
public class MyBean {}
```

```xml
<context:component-scan base-package="com.example" />
```

---

16. Các phạm vi (scope) khác nhau của Spring Bean

- `singleton` (mặc định): Một instance duy nhất cho mỗi container.
- `prototype`: Tạo mới mỗi lần request bean.
- `request`: Một instance cho mỗi HTTP request (chỉ dùng trong web).
- `session`: Một instance cho mỗi session.
- `global-session`: Dùng trong ứng dụng Portlet.

Cấu hình:

```
@Scope("prototype")
@Component
public class MyPrototypeBean {}
```

---

17. Vòng đời của Spring Bean

Spring hỗ trợ lifecycle thông qua:

1. Implement các interface `InitializingBean` và `DisposableBean`
2. Dùng annotation `@PostConstruct` và `@PreDestroy`
3. Dùng `init-method`, `destroy-method` trong XML.

Ví dụ:

```
@Component
public class MyBean implements InitializingBean, DisposableBean {
    public void afterPropertiesSet() throws Exception {
        // init
    }
    public void destroy() throws Exception {
        // cleanup
    }
}
```

---

18. Lấy ServletContext và ServletConfig trong Spring Bean

Cách 1: Implement interface

```
public class MyBean implements ServletContextAware {
    public void setServletContext(ServletContext context) {
        this.servletContext = context;
    }
}
```

Cách 2: Dùng `@Autowired`

```
@Autowired
ServletContext servletContext;
```

---

19. Bean wiring và @Autowired là gì?

- Bean wiring: Quá trình inject dependency vào bean.
- `@Autowired`: Tự động inject dependency theo kiểu byType.

Ví dụ:

```
@Component
public class MyService {
    @Autowired
    private MyRepository repo;
}
```

Cần bật annotation config:

```
<context:annotation-config />
```

---

20. Các loại Spring Bean Autowiring

1. **byName**: Tìm bean theo tên biến.
2. **byType**: Tìm bean theo kiểu dữ liệu.
3. **constructor**: Tự động inject thông qua constructor.
4. **@Autowired + @Qualifier**: Kết hợp annotation.

Ví dụ:

```
@Autowired
@Qualifier("myRepo")
private MyRepository repository;
```

---

21. Spring Bean có cung cấp an toàn cho thread không?

- **Không**, bởi mặc định Spring Bean có scope là `singleton`, nên mọi thread dùng chung một instance => dễ gây race condition nếu có shared state.
- Để thread-safe: tránh dùng biến instance có thể thay đổi hoặc dùng các scope khác như `prototype`, `request`, `session`.

Ví dụ:

```
@Component
@Scope("prototype")
public class MyTaskProcessor {
    private int taskCount; // không chia sẻ giữa các thread
}
```

22. Controller trong Spring MVC là gì?

- Là lớp xử lý các HTTP request.
- Được đánh dấu với `@Controller` và kết hợp `@RequestMapping` để ánh xạ URL đến method tương ứng.

Ví dụ:

```
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String homePage() {
        return "home"; // trả về view "home.jsp"
    }
}
```

23. Sự khác biệt giữa @Component, @Controller, @Repository, @Service

- `@Component`: annotation chung để đánh dấu một bean.
- `@Controller`: dành cho controller trong Spring MVC.
- `@Service`: đánh dấu lớp chứa logic nghiệp vụ (business service).
- `@Repository`: đánh dấu lớp DAO và tự động xử lý exception liên quan đến persistence.

Ví dụ:

```
@Repository
public class UserDao {}

@Service
public class UserService {}

@Controller
public class UserController {}
```

24. DispatcherServlet và ContextLoaderListener là gì?

- **DispatcherServlet**: Là Front Controller của Spring MVC, chịu trách nhiệm tiếp nhận request và phân phối đến controller tương ứng.
- **ContextLoaderListener**: Khởi tạo ApplicationContext cấp gốc dùng chung trong toàn ứng dụng.

Cấu hình:

```
<listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>

<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

25. ViewResolver trong Spring là gì?

- Dùng để ánh xạ tên view trả về từ controller thành file giao diện cụ thể (.jsp, .html...)

Ví dụ với InternalResourceViewResolver:

```
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver
">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
```

Khi controller trả về "home" → chuyển thành `/WEB-INF/views/home.jsp`.

26. MultipartResolver là gì?

- Là bean dùng để xử lý việc **upload file** trong Spring MVC.
- Spring cung cấp `CommonsMultipartResolver` và `StandardServletMultipartResolver`.

Ví dụ cấu hình:

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolve
r" />
```

Controller:

```java
@PostMapping("/upload")
public String upload(@RequestParam("file") MultipartFile file) {
    // xử lý file
}
```

27. Cách xử lý ngoại lệ trong Spring MVC

1. **@ExceptionHandler**: trong từng controller

```java
@ExceptionHandler(Exception.class)
public String handleError(Exception e) {
    return "error";
}
```

2. **@ControllerAdvice**: handler toàn cục

```java
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public String handleGlobalError(Exception e) {
        return "global-error";
    }
}
```

3. **HandlerExceptionResolver**: tùy biến xử lý exception

```xml
<bean
class="org.springframework.web.servlet.handler.SimpleMappingExceptionRes
olver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.lang.Exception">error</prop>
        </props>
    </property>
</bean>
```

28. Tạo ApplicationContext trong Java

1. Dùng annotation:

```java
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
```

2. Dùng XML:

```
ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
```

3. Dùng file hệ thống:

```
FileSystemXmlApplicationContext context = new
FileSystemXmlApplicationContext("C:/config/beans.xml");
```

29. Có thể có nhiều file cấu hình Spring không?

→ Có thể, bằng cách:

1. Trong `contextConfigLocation`:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/app-config.xml /WEB-INF/db-config.xml</param-
value>
</context-param>
```

2. Import trong XML:

```
<import resource="db-config.xml" />
```

3. Trong annotation:

```
@Import({AppConfig.class, DbConfig.class})
```

Tách file giúp chia module rõ ràng, dễ quản lý.

30. ContextLoaderListener là gì?

`ContextLoaderListener` là lớp lắng nghe trong Spring, được sử dụng để tải ngữ cảnh gốc và xác định các cấu hình Spring bean sẽ hiển thị với tất cả các ngữ cảnh khác. Được cấu hình trong file `web.xml` như sau:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
```

31. Các Cấu Hình Tối Thiểu Cần Thiết Để Tạo Ứng Dụng Spring MVC

Để tạo ứng dụng Spring MVC đơn giản, cần:

- Thêm dependencies `spring-context` và `spring-webmvc`.
- Cấu hình `DispatcherServlet` trong `web.xml`.
- Cấu hình Spring bean và view resolver.
- Tạo Controller với các ánh xạ yêu cầu.

32. Liên Kết Spring MVC với Kiến Trúc MVC

Spring MVC sử dụng mô hình MVC. `DispatcherServlet` là Front Controller xử lý các yêu cầu và ủy thác cho controller. Model là các Java Bean, và View có thể là JSP, HTML. Các view resolvers giúp xác định đúng view-page để trả về.

33. Achieve Localization Trong Spring MVC

Spring hỗ trợ localization thông qua `messageSource` bean và `localeResolver`. Ví dụ cấu hình:

```
<bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessa
geSource">
    <property name="basename" value="classpath:messages" />
    <property name="defaultEncoding" value="UTF-8" />
</bean>

<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
    <property name="defaultLocale" value="en" />
    <property name="cookieName" value="myAppLocaleCookie" />
    <property name="cookieMaxAge" value="3600" />
</bean>
```

34. Tạo Restful Web Service Trả Về Phản Hồi JSON

Spring cung cấp tích hợp Jackson để gửi phản hồi JSON trong dịch vụ web. Cấu hình:

```
<bean id="jsonMessageConverter"
class="org.springframework.http.converter.json.MappingJackson2HttpMessag
eConverter" />
<bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMapp
ingHandlerAdapter">
    <property name="messageConverters">
        <list>
            <ref bean="jsonMessageConverter"/>
        </list>
    </property>
</bean>
```

Ví dụ trong Controller:

```
@RequestMapping(value = "/employee/{id}", method = RequestMethod.GET)
public @ResponseBody Employee getEmployee(@PathVariable("id") int empId)
{
    return empData.get(empId);
}
```

35. Các Chú Thích Quan Trọng Trong Spring

- `@Controller`: Dùng cho lớp Controller.
- `@RequestMapping`: Ánh xạ URI cho phương thức xử lý.
- `@ResponseBody`: Gửi đối tượng dưới dạng JSON hoặc XML.
- `@PathVariable`: Ánh xạ các giá trị động từ URI.
- `@Autowired`: Autowiring dependency trong Spring.
- `@Qualifier`: Dùng để phân biệt các bean cùng loại.
- `@Service`: Dùng cho lớp dịch vụ.
- `@Scope`: Định nghĩa phạm vi của Spring bean.
- `@Configuration`, `@ComponentScan`, `@Bean`: Dùng trong cấu hình Java.
- Chú thích AspectJ: `@Aspect`, `@Before`, `@After`, `@Around`, `@Pointcut`.

36. Gửi Một Object Dưới Dạng Phản Hồi Của Phương Thức Xử Lý Controller

Có, chúng ta có thể gửi phản hồi dưới dạng object bằng cách sử dụng chú thích `@ResponseBody`. Đây là cách gửi dữ liệu JSON hoặc XML trong các dịch vụ web.

37. Tải File Lên Trong Ứng Dụng Spring MVC

Spring hỗ trợ tải file qua `MultipartResolver`. Cần cấu hình như sau và xử lý file trong Controller:

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolve
```

```
r" />
```

38. Xác Thực Dữ Liệu Biểu Mẫu (Form Data) Trong Spring Web MVC

Spring hỗ trợ xác thực với JSR-303 annotations và Validator interface. Để sử dụng xác thực dựa trên JSR-303, cần chú thích các bean với các xác nhận bắt buộc như `@NotNull`, `@Size`.

39. Spring MVC Interceptor Là Gì Và Làm Thế Nào Để Sử Dụng Nó?

Spring MVC Interceptors giúp chặn yêu cầu của khách hàng ở các giai đoạn `preHandle`, `postHandle`, và `afterCompletion`. Cấu hình Interceptor trong Spring:

```
<bean
class="org.springframework.web.servlet.handler.HandlerInterceptorAdapter
" />
```

40. Lớp Spring JdbcTemplate Là Gì Và Cách Sử Dụng Nó?

`JdbcTemplate` giúp thực hiện các thao tác cơ sở dữ liệu mà không cần viết nhiều mã boilerplate như mở/đóng kết nối, xử lý `ResultSet`, `PreparedStatement`.

```
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
String sql = "SELECT * FROM employees";
List<Employee> employees = jdbcTemplate.query(sql, new
EmployeeRowMapper());
```

Để sử dụng servlet container được cấu hình JNDI DataSource, chúng ta cần cấu hình nó trong file cấu hình spring bean và sau đó inject nó vào spring bean dưới dạng dependency. Sau đó, chúng ta có thể sử dụng nó với JdbcTemplate để thực hiện các hoạt động cơ sở dữ liệu. Cấu hình mẫu sẽ là:

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

42. Làm thế nào để achieve Transaction Management trong Spring?

Spring framework cung cấp hỗ trợ quản lý giao dịch thông qua Declarative Transaction Management cũng như quản lý giao dịch có lập trình. Declarative transaction management được sử dụng rộng rãi nhất vì nó dễ sử dụng và hoạt động trong hầu hết các trường hợp. Chúng ta sử dụng phương pháp chú thích với chú thích `@Transactional` để quản lý giao dịch so sánh. Chúng ta cần cấu hình trình quản lý giao dịch cho DataSource trong file cấu hình spring bean.

```
<bean id="transactionManager"

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

43. Spring DAO là gì?

Spring DAO được cung cấp để làm việc với các công nghệ truy cập dữ liệu như JDBC, Hibernate một cách nhất quán và dễ dàng. Ví dụ, chúng ta có `JdbcDaoSupport`, `HibernateDaoSupport`, `JdoDaoSupport` và `JpaDaoSupport` cho các công nghệ tương ứng. Spring DAO cũng cung cấp tính nhất quán trong phân cấp ngoại lệ (exception hierarchy) nên chúng ta không cần phải nắm bắt các ngoại lệ cụ thể.

44. Làm thế nào để tích hợp Spring và Hibernate Frameworks?

Chúng ta có thể sử dụng module Spring ORM để tích hợp các Spring Framework và Hibernate nếu đang sử dụng Hibernate 3+ trong đó `SessionFactory` cung cấp session hiện tại, khi đó bạn nên tránh sử dụng các lớp `HibernateTemplate` hoặc `HibernateDaoSupport` và tốt hơn là sử dụng mẫu DAO với dependency injection để tích hợp. Spring ORM cung cấp hỗ trợ cho việc sử dụng quản lý giao dịch khai báo Spring, vì vậy bạn nên sử dụng nó thay vì sử dụng mã Hibernate boiler-plate để quản lý giao dịch.

45. Spring Security là gì?

Spring security framework tập trung vào việc cung cấp cả xác thực (authentication) và ủy quyền (authorization) trong các ứng dụng Java. Nó cũng xử lý hầu hết các lỗ hổng bảo mật phổ biến như tấn công CSRF. Việc sử dụng Spring Security trong các ứng dụng web rất có lợi và dễ dàng thông qua việc sử dụng các chú thích như `@EnableWebSecurity`.

46. Làm thế nào để đưa (inject) `java.util.Properties` vào Spring Bean?

Chúng ta cần xác định bean `propertyConfigurer` sẽ tải các thuộc tính (Property) từ file thuộc tính đã cho. Sau đó, chúng ta có thể sử dụng hỗ trợ Spring EL để đưa các thuộc tính vào các bean dependency khác. Ví dụ:

```
<bean id="propertyConfigurer"

class="org.springframework.context.support.PropertySourcesPlaceholderCon
figurer">
    <property name="location" value="/WEB-INF/application.properties" />
</bean>

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

Nếu bạn đang sử dụng chú thích để cấu hình spring bean, thì bạn có thể chèn thuộc tính (inject property) như bên dưới:

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

47. Kể tên một số mẫu thiết kế (design pattern) được sử dụng trong Spring Framework?

Spring Framework đang sử dụng rất nhiều mẫu thiết kế, một số mẫu phổ biến là:

- **Singleton Pattern**: Tạo bean với phạm vi mặc định.
- **Factory Pattern**: Các lớp Bean Factory.
- **Prototype Pattern**: Phạm vi bean.
- **Adapter Pattern**: Spring Web và Spring MVC.
- **Proxy Pattern**: Hỗ trợ lập trình hướng Spring Aspect.
- **Template Method Pattern**: JdbcTemplate, HibernateTemplate, ...
- **Front Controller**: Spring MVC DispatcherServlet.
- **Data Access Object**: Hỗ trợ Spring DAO.
- **Dependency Injection và Aspect Oriented Programming**.

48. Một số kinh nghiệm quý báo nhất cho Spring Framework là gì?

Một số kinh nghiệm quý báo nhất cho Spring Framework là:

- Tránh số phiên bản trong tham chiếu lược đồ để đảm bảo chúng ta có cấu hình mới nhất.
- Phân chia các cấu hình spring bean dựa trên mối quan tâm của chúng như `spring-jdbc.xml`, `spring-security.xml`.
- Đối với Spring bean được sử dụng trong nhiều ngữ cảnh trong Spring MVC, hãy tạo chúng trong ngữ cảnh gốc và khởi tạo với trình nghe (listener).
- Cấu hình các bean dependency càng nhiều càng tốt, cố gắng tránh autowiring càng nhiều càng tốt.
- Đối với các thuộc tính mức ứng dụng, cách tốt nhất là tạo một file thuộc tính và đọc nó trong file cấu hình spring bean.
- Đối với các ứng dụng nhỏ hơn, chú thích rất hữu ích nhưng đối với các ứng dụng lớn hơn, chú thích có thể trở thành một vấn đề. Nếu chúng ta có tất cả cấu hình trong file XML, việc duy trì nó sẽ dễ dàng hơn.
- Sử dụng chú thích chính xác cho các thành phần để hiểu mục đích một cách dễ dàng. Đối với các dịch vụ, hãy sử dụng `@Service` và đối với DAO bean, hãy sử dụng `@Repository`.
- Spring framework có rất nhiều module, hãy sử dụng những gì bạn cần. Loại bỏ tất cả các dependency bổ sung thường được thêm vào khi bạn tạo dự án thông qua các mẫu Spring Tool Suite.
- Nếu bạn đang sử dụng Aspects, hãy đảm bảo rằng join-point càng hẹp càng tốt để tránh advice về các phương thức không mong muốn. Xem xét các chú thích tùy chỉnh dễ sử dụng hơn và tránh mọi vấn đề.
- Sử dụng dependency injection khi có lợi ích thực tế, chỉ vì lợi ích của sự liên kết chặt chẽ chứ không sử dụng nó vì khó bảo trì hơn.

49. Core container là gì?

Đây là module cơ bản của Spring, cái mà sẽ cung cấp những tính năng cơ bản của Spring framework. `BeanFactory` chính là trái tim của bất kỳ Spring-based application nào, Spring framework được xây dựng trên module này và nó cũng chính là Spring container.

50. Hạn chế của autowiring

Các hạn chế của việc autowiring:

- **Overriding**: Vẫn có thể định nghĩa các dependencies và việc này sẽ luôn luôn override autowiring.
- **Data types**: Không thể autowire những thuộc tính đơn giản như primitive, String và Classes.
- **Confusing**: Autowiring thì không tường minh, vì thế sử dụng khai báo tường minh có thể là cách khôn ngoan hơn.