课程目标

1、介绍Spring5新特性、重点介绍WebFlux。
2、分析Spring 中经典的高频面试题。

内容定位

本课主要目的是对Spring进行一次总体回顾，了解Spring的发展趋势以及近期更新版本的新特性。

1. Spring5新特性简介

Spring5于2017年9月发布了通用版本（GA），它标志着自2013年12月以来第一个主要Spring Framework 版本。它提供了一些人们期待已久的改进，还采用了一种全新的编程范例，以反应式描述中陈述的反应式原则为基础。

这个版本是很长时间以来最令人激动的Spring Framework 版本。Spring 5兼容Java™8和 JDK 9，它集成了反应式流，以方便后续提供一种颠覆性方法来实现端点和Web应用程序开发。

当然，反应式编程不仅是此版本的主题，还是令许多程序员激动不已的重大特性。人们对能够针对负载波动进行无缝扩展的容灾和响应式服务的需求在不断增加，Spring5很好地满足了这一需求。

我们将介绍JavaSE 8 和 JavaEE 7 API升级的基本内容、Spring5的新反应式编程模型、对HTTP/2支持，以及Spring通过Kotlin对函数式编程的全面支持。我还会简要介绍测试和性能增强，最后介绍对Spring核心和容器的一般性修订。

1.1. 升级到Java SE 8 和 Java EE 7

以前的Spring Framework 中一直在支持一些弃用的Java版本，而Spring5已从旧包袱中解放出来。为了充分利用Java8特性，它的代码库已进行了改进，而且该框架要求将Java8作为最低的JDK版本。

Spring 5在类路径（和模块路径）上完全兼容Java9，而且它通过了JDK9测试套件的测试。对Java 9爱好者而言，这是一条好消息，因为在Java9发布后，Spring 能立即使用它。

在API级别上，Spring5兼容JavaEE8技术，满足对Servlet 4.0、Bean Validation2.0和全新的JSON Binding API 的需求。对JavaEEAPI的最低要求为V7，该版本引入了针对Servlet、JPA和Bean

Validation API的次要版本。

1.2. 反应式编程模型

Spring 5最令人兴奋的新特性是它的反应式编程模型。Spring 5 Framework基于一种反应式基础而构建，而且是完全异步和非阻塞的。只需少量的线程，新的事件循环执行模型就可以垂直扩展。

该框架采用反应式流来提供在反应式组件中传播负压的机制。负压是一个确保来自多个生产者的数据不会让使用者不堪重负的概念。

Spring WebFlux是Spring5的反应式核心，它为开发人员提供了两种为Spring Web编程而设计的编程模型：一种基于注解的模型和 Functional Web Framework（WebFlux.fn）。

基于注解的模型是Spring WebMVC的现代替代方案，该模型基于反应式基础而构建，而Functional Web Framework 是基于@Controller 注解的编程模型的替代方案。这些模型都通过同一种反应式基础来运行，后者调整非阻塞HTTP来适应反应式流API。

1.3. 使用注解进行编程

Web MVC程序员应该对Spring5的基于注解的编程模型非常熟悉。Spring 5调整了Web MVC的@Controller 编程模型，采用了相同的注解。

在下面的代码中BookController类提供了两个方法，分别响应针对某个图书列表的HTTP请求，以及针对具有给定id的图书的HTTP 请求。请注意 resource方法返回的对象（Mono和Flux）。这些对象是实现反应式流规范中的Publisher接口的反应式类型。它们的职责是处理数据流。Mono对象处理一个仅含1个元素的流，而Flux表示一个包含N个元素的流。

反应式控制器

```
@RestController
public class BookController {

    @GetMapping("/book")
    Flux<Book> list() {
        return this.repository.findAll();
    }

    @GetMapping("/book/{id}")
    Mono<Book> findById(@PathVariable String id) {
        return this.repository.findOne(id);
    }
}
```

这是针对Spring Web编程的注解。现在我们使用函数式Web框架来解决同一个问题。

1.4. 支持函数式编程

Spring5的新函数式方法将请求委托给处理函数，这些函数接受一个服务器请求实例并返回一种反应式类型。来看一段代码，创建 BookHandler类，其中listBook()和getBook()方法相当于Controller中的功能。

```
public class BookHandler {

    public Mono<ServerResponse> listBooks(ServerRequest request) {
        return ServerResponse.ok()
                .contentType(APPLICATION_JSON)
                .body(repository.allPeople(), Book.class);
    }

    public Mono<ServerResponse> getBook(ServerRequest request) {
        return repository.getBook(request.pathVariable("id"))
                .then(book -> ServerResponse.ok()
                        .contentType(APPLICATION_JSON)
                        .body(fromObject(book)))
                .otherwiseIfEmpty(ServerResponse.notFound().build());
    }
}
```

通过路由函数来匹配HTTP请求参数与媒体类型，将客户端请求路由到处理函数。下面的代码展示了图书资源端点URI将调用委托给合适的处理函数：

```
BookHandler handler = new BookHandler();

RouterFunction<ServerResponse> personRoute =
        route(
                GET("/books/{id}")
                        .and(accept(APPLICATION_JSON)), handler::getBook)
                .andRoute(
                        GET("/books")
                                .and(accept(APPLICATION_JSON)), handler::listBooks);
```

这些示例背后的数据存储库也支持完整的反应式体验，该体验是通过Spring Data 对反应式Couchbase、Reactive MongoDB和Cassandra的支持来实现的。

1.5. 使用REST端点执行反应式编程

新的编程模型脱离了传统的Spring WebMVC模型，引入了一些很不错的新特性。

举例来说，WebFlux模块为RestTemplate提供了一种完全非阻塞、反应式的替代方案，名为WebClient。下面创建一个WebClient，并调用books 端点来请求一本给定id为1234的图书。

通过WebClient 调用REST 端点

```
Mono<Book> book = WebClient.create("http://localhost:8080")
        .get()
        .url("/books/{id}", 1234)
        .accept(APPLICATION_JSON)
        .exchange(request)
        .then(response -> response.bodyToMono(Book.class));
```

1.6. 对HTTP/2支持

HTTP/2幕后原理：要了解HTTP/2如何提高传输性能，减少延迟，并帮助提高应用程序吞吐量，从而提供经过改进的丰富Web体验。

Spring Framework 5.0提供专门的HTTP/2特性支持，还支持人们期望出现在JDK9中的新HTTP客户端。尽管HTTP/2的服务器推送功能已通过Jetty Servlet 引擎的ServerPushFilter类向Spring 开发人员公开了很长一段时间，但如果发现Spring5中开箱即用地提供了HTTP/2性能增强，Web优化者们一定会为此欢呼雀跃。

Servlet 4.0支持在Spring5.1中提供。到那时，HTTP/2新特性将由Tomcat 9.0、Jetty 9.3和Undertow 1.4原生提供。

1.7. Kotlin 和 Spring WebFlux

Kotlin 是一种来自JetBrains的面向对象的语言，它支持函数式编程。它的主要优势之一是与Java有非常高的互操作性。通过引入对Kotlin的专门支持，Spring在V5中全面吸纳了这一优势。它的函数式编程风格与Spring WebFlux模块完美匹配，它的新路由DSL利用了函数式Web框架以及干净且符合语言习惯的代码。可以像下面代码中这样简单地表达端点路由：

Kotlin的用于定义端点的路由 DSL

```
@Bean
fun apiRouter() = router {
    (accept(APPLICATION_JSON) and "/api").nest {
        "/book".nest {
            GET("/", bookHandler::findAll)
            GET("/{id}", bookHandler::findOne)
        }
        "/video".nest {
            GET("/", videoHandler::findAll)
            GET("/{genre}", videoHandler::findByGenre)
        }
    }
}
```

使用Kotlin 1.1.4+时，还添加了对Kotlin的不可变类的支持（通过带默认值的可选参数），以及对完全支持null的API的支持。

1.8. 使用 Lambda 表达式注册Bean

作为传统XML和JavaConfig的替代方案，现在可以使用lambda 表达式注册 Spring bean，使bean可以实际注册为提供者。下面代码中使用lambda 表达式注册了一个Book bean。

将Bean 注册为提供者

```
GenericApplicationContext context = new GenericApplicationContext();
context.registerBean(Book.class, () -> new
       Book(context.getBean(Author.class))
);
```

1.9. Spring Web MVC支持最新的API

全新的WebFlux模块提供了许多新的、令人兴奋的功能，但Spring 5也迎合了愿意继续使用Spring MVC的开发人员的需求。Spring5中更新了模型-视图-控制器框架，以兼容WebFlux和最新版的Jackson 2.9和Protobuf3.0，甚至包括对新的Java EE 8 JSON-Binding API的支持。

除了HTTP/2特性的基础服务器实现之外，Spring WebMVC还通过MVC控制器方法的一个参数来支持Servlet 4.0的PushBuilder。最后，WebMVC 全面支持Reactor3.1的Flux和Mono对象，以及RxJava 1.3和2.1，它们被视为来自MVC控制器方法的返回值。这项支持的最终目的是支持Spring Data中的新的反应式WebClient 和反应式存储库。

1.10. 使用Spring5 执行条件和并发测试

JUnit和Spring 5：Spring 5全面接纳了函数式范例，并支持JUnit5及其新的函数式测试风格。还提供了对JUnit4的向后兼容性，以确保不会破坏旧代码。

Spring5的测试套件通过多种方式得到了增强，但最明显的是它对JUnit5的支持。现在可以在您的单元测试中利用Java8中提供的函数式编程特性。以下代码演示了这一支持：

JUnit 5全面接纳了Java8流和lambda表达式

```
@Test
void givenStreamOfInts_SumShouldBeMoreThanFive() {
   assertTrue(Stream.of(20, 40, 50)
           .stream()
           .mapToInt(i -> i)
           .sum() > 110, () -> "Total should be more than 100");
}
```

迁移到JUnit5：如果您对升级到JUnit5持观望态度，Steve Perry的分两部分的深入剖析教程将说服您冒险尝试。

Spring 5继承了JUnit 5在Spring TestContext Framework 内实现多个扩展API的灵活性。举例而言，开发人员可以使用JUnit5的条件测试执行注解@EnabledIf 和@DisabledIf来自动计算一个SpEL（Spring Expression Language）表达式，并适当地启用或禁用测试。借助这些注解，Spring5支持以前很难实现的复杂的条件测试方案。Spring TextContext Framework 现在能够并发执行测试。

使用Spring WebFlux执行集成测试

Spring Test 现在包含一个WebTestClient，后者支持对Spring WebFlux服务器端点执行集成测试。

WebTestClient 使用模拟请求和响应来避免耗尽服务器资源，并能直接绑定到WebFlux 服务器基础架构。

WebTestClient 可绑定到真实的服务器，或者使用控制器或函数。在下面的代码中，WebTestClient被绑定到 localhost：

绑定到 localhost的 WebTestClient

```
WebTestClient testClient = WebTestClient
        .bindToServer()
        .baseUrl("http://localhost:8080")
        .build();
```

将WebTestClient 绑定到RouterFunction

```
RouterFunction bookRouter = RouterFunctions.route(
        RequestPredicates.GET("/books"),
        request -> ServerResponse.ok().build()
);
WebTestClient
        .bindToRouterFunction(bookRouter)
        .build().get().uri("/books")
        .exchange()
        .expectStatus().isOk()
        .expectBody().isEmpty();
```

1.11. 包清理和弃用

Spring 5中止了对一些过时API的支持。遭此厄运的还有Hibernate3和4，为了支持Hibernate5，它们遭到了弃用。另外，对Portlet、Velocity、JasperReports、XMLBeans、JDO和Guava的支持也已中止。

包级别上的清理工作仍在继续：Spring 5不再支持 beans.factory.access、jdbc.support.nativejdbc、mock.staticmock（来自spring-aspects模块）或web.view.tiles2M。Tiles3现在是Spring的最低要求。

1.12. Spring 核心和容器的一般更新

Spring Framework 5改进了扫描和识别组件的方法，使大型项目的性能得到提升。目前，扫描是在编译时执行的，而且向META-INF/spring.components文件中的索引文件添加了组件坐标。该索引是通过一个为项目定义的特定于平台的应用程序构建任务来生成的。

标有来自javax包的注解的组件会添加到索引中，任何带@Index 注解的类或接口都会添加到索引中。Spring的传统类路径扫描方式没有删除，而是保留为一种后备选择。有许多针对大型代码库的

明显性能优势，而托管许多Spring项目的服务器也会缩短启动时间。

Spring 5还添加了对@Nullable的支持，后者可用于指示可选的注入点。使用者现在必须准备接受null值。此外，还可以使用此注解来标记可以为null的参数、字段和返回值。@Nullable 主要用于IntelliJ IDEA等IDE，但也可用于Eclipse和FindBugs，它使得在编译时处理null值变得更方便，而无需在运行时发送NullPointerExceptions。

Spring Logging 还提升了性能，自带开箱即用的Commons Logging 桥接器。现在已通过资源抽象支持防御性编程，为getFile 访问提供了isFile 指示器。

1.13. 如何看Spring5

Spring5的首要特性是新的反应式编程模型，这代表着对提供可无缝扩展、基于Spring的响应式服务的重大保障。随着人们对Spring5的采用，开发人员有望看到反应式编程将会成为使用Java 语言的Web和企业应用程序开发的未来发展道路。

未来的Spring Framework 版本将继续体现这一承诺，因为Spring Security、Spring Data和Spring Integration 有望采用反应式编程的特征和优势。

总之，Spring5代表着一次大受Spring 开发人员欢迎的华丽转变，同时也为其他框架指出了一条发展之路。Spring5的升级也为Spring Boot、Spring Cloud 提供非常丰富的经验，Spring不仅仅只是一个框架，已然成为了一种编程生态。

2. Spring中经典的高频面试题

2.1. 什么是Spring框架?Spring框架有哪些主要模块?

Spring 框架是一个为Java应用程序的开发提供了综合、广泛的基础性支持的Java平台。Spring 帮助开发者解决了开发中基础性的问题，使得开发人员可以专注于应用程序的开发。Spring框架本身亦是按照设计模式精心打造，这使得我们可以在开发环境中安心的集成Spring框架，不必担心Spring是如何在后台进行工作的。

Spring 框架至今已集成了20多个模块。这些模块主要被分如下图所示的核心容器、数据访问/集成、Web、AOP（面向切面编程）、工具、消息和测试模块。

2.2. 使用Spring框架能带来哪些好处?

下面列举了一些使用Spring 框架带来的主要好处：
1、Dependency Injection（DI）方法使得构造器和JavaBean properties文件中的依赖关系一目了然。

2、与EJB容器相比较，IOC容器更加趋向于轻量级。这样一来IOC容器在有限的内存和CPU资源的情况下进行应用程序的开发和发布就变得十分有利。

3、Spring 并没有闭门造车，Spring 利用了已有的技术比如ORM框架、logging框架、J2EE、Quartz和JDK Timer，以及其他视图技术。

4、Spring框架是按照模块的形式来组织的。由包和类的编号就可以看出其所属的模块，开发者仅仅需要选用他们需要的模块即可。

5、要测试一项用Spring 开发的应用程序十分简单，因为测试相关的环境代码都已经囊括在框架中了。

更加简单的是，利用JavaBean形式的POJO类，可以很方便的利用依赖注入来写入测试数据。

6、Spring的Web框架亦是一个精心设计的Web MVC框架，为开发者们在web框架的选择上提供了一个除了主流框架比如Struts、过度设计的、不流行web框架的以外的有力选项。

7、Spring 提供了一个便捷的事务管理接口，适用于小型的本地事务处理（比如在单DB的环境下）和复杂的共同事务处理（比如利用JTA的复杂DB环境）。

2.3. 什么是控制反转(IOC)? 什么是依赖注入(DI)?

1、控制反转是应用于软件工程领域中的，在运行时被装配器对象来绑定耦合对象的一种编程技巧，对象之间耦合关系在编译时通常是未知的。在传统的编程方式中，业务逻辑的流程是由应用程序中的早已被设定好关联关系的对象来决定的。在使用控制反转的情况下，业务逻辑的流程是由对象关系图来决定的，该对象关系图由装配器负责实例化，这种实现方式还可以将对象之间的关联关系的定义抽象化。而绑定的过程是通过“依赖注入”实现的。

2、控制反转是一种以给予应用程序中目标组件更多控制为目的设计范式，并在我们的实际工作中起到了有效的作用。

3、依赖注入是在编译阶段尚未知所需的功能是来自哪个的类的情况下，将其他对象所依赖的功能对象实例化的模式。这就需要一种机制用来激活相应的组件以提供特定的功能，所以依赖注入是控制反转的基础。否则如果在组件不受框架控制的情况下，框架又怎么知道要创建哪个组件?

2.4. 在Java中依赖注入有哪些方式?

1.构造器注入
2.Setter 方法注入
3.接口注入

2.5. BeanFactory和ApplicationContext有哪些区别?

BeanFactory 可以理解为含有bean集合的工厂类。BeanFactory包含了种bean的定义，以便在接收到客户端请求时将对应的bean实例化。

BeanFactory 还能在实例化对象的时生成协作类之间的关系。此举将bean 自身与bean客户端的配置中解放出来。BeanFactory 还包含了bean生命周期的控制，调用客户端的初始化方法（initialization Methods）和销毁方法（destruction Methods）。

从表面上看，ApplicationContext 如同 bean factory一样具有bean定义、bean关联关系的设置，根据请求分发bean的功能。但ApplicationContext在此基础上还提供了其他的功能。

1.提供了支持国际化的文本消息
2.统一的资源文件读取方式
3.已在监听器中注册的bean的事件

以下是三种较常见的ApplicationContext 实现方式：
1、ClassPathXmlApplicationContext：从classpath的XML配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。

```
ApplicationContext context = new  ClassPathXmlApplicationContext("application.xml");
```

2、FileSystemXmlApplicationContext：由文件系统中的XML配置文件读取上下文。

```
ApplicationContext context = new FileSystemXmlApplicationContext("application.xml");
```

3、XmlWebApplicationContext：由Web应用的XML文件读取上下文。

2.6. Spring提供几种配置方式来设置元数据?

将Spring配置到应用开发中有以下三种方式：
1.基于XML的配置
2.基于注解的配置
3.基于Java的配置

2.7. 如何使用XML配置的方式配置Spring?

在Spring框架中，依赖和服务需要在专门的配置文件来实现。这些配置文件的格式通常用开头，然后一系列的bean定义和专门的应用配置选项组成。

SpringXML配置的主要目的时候是使所有的Spring 组件都可以用xml文件的形式来进行配置。这意味着不会出现其他的Spring 配置类型（比如声明的方式或基于Java Class的配置方式）

Spring的XML配置方式是使用被Spring命名空间的所支持的一系列的XML标签来实现的。Spring有以下主要的命名空间：context、beans、jdbc、tx、aop、mvc和aso。

```
    <bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHa
        <property name="messageConverters">
            <list>
                <ref bean="mappingJacksonHttpMessageConverter" />
            </list>
        </property>
```

```
    </bean>

    <bean id="mappingJacksonHttpMessageConverter"
          class="org.springframework.http.converter.json.MappingJackson2HttpMessageConv
        <property name = "supportedMediaTypes">
            <list>
                <bean class="org.springframework.http.MediaType">
                    <constructor-arg index="0" value="text"/>
                    <constructor-arg index="1" value="plain"/>
                    <constructor-arg index="2" value="UTF-8"/>
                </bean>
                <bean class="org.springframework.http.MediaType">
                    <constructor-arg index="0" value="*"/>
                    <constructor-arg index="1" value="*"/>
                    <constructor-arg index="2" value="UTF-8"/>
                </bean>
                <bean class="org.springframework.http.MediaType">
                    <constructor-arg index="0" value="text"/>
                    <constructor-arg index="1" value="*"/>
                    <constructor-arg index="2" value="UTF-8"/>
                </bean>
                <bean class="org.springframework.http.MediaType">
                    <constructor-arg index="0" value="application"/>
                    <constructor-arg index="1" value="json"/>
                    <constructor-arg index="2" value="UTF-8"/>
                </bean>
            </list>
        </property>
    </bean>


</beans>
```

下面这个web.xml仅仅配置了DispatcherServlet，这件最简单的配置便能满足应用程序配置运行时组件的需求。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xm
         version="3.1">
```

```
6
7       <display-name>Sitedev Web Application</display-name>
8
9       <!-- loading spring context start -->
10      <context-param>
11          <param-name>contextConfigLocation</param-name>
12          <param-value>classpath:application-web.xml</param-value>
13      </context-param>
14      <listener>
15          <listener-class>org.springframework.web.context.ContextLoaderListener</listener
16      </listener>
17      <!-- loading spring context end -->
18
19      <!-- springmvc config start -->
20      <servlet>
21          <servlet-name>dispatcher</servlet-name>
22          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class
23          <init-param>
24              <param-name>contextClass</param-name>
25              <param-value>org.springframework.web.context.support.AnnotationConfigWebApp
26          </init-param>
27          <load-on-startup>1</load-on-startup>
28          <async-supported>true</async-supported>
29      </servlet>
30      <servlet-mapping>
31          <servlet-name>dispatcher</servlet-name>
32          <url-pattern>/*</url-pattern>
33      </servlet-mapping>
34      <!-- springmvc config end -->
35
36
37  </web-app>
```

2.8. Spring提供哪些配置形式?

Spring 对Java配置的支持是由@Configuration 注解和@Bean 注解来实现的。由@Bean注解的方法将会实例化、配置和初始化一个新对象，这个对象将由Spring的IOC容器来管理。@Bean 声明所起到的作用与元素类似。被@Configuration所注解的类则表示这个类的主要目的是作为bean定义的资源。被@Configuration 声明的类可以通过在同一个类的内部调用@bean方法来设置嵌入bean的依赖关系。

最简单的@Configuration声明类请参考下面的代码：

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

对于上面的@Beans配置文件相同的XML配置文件如下：

```
<beans>
   <bean id="myService" class="com.gupaoedu.services.MyServiceImpl"/>
</beans>
```

上述配置方式的实例化方式如下：利用AnnotationConfigApplicationContext类进行实例化

```
public static void main(String[] args) {
   ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
   MyService myService = ctx.getBean(MyService.class);
   myService.doStuff();
}
```

要使用组件组建扫描，仅需用@ComponentScan进行注解即可：

```
@Configuration
@ComponentScan(basePackages = "com.gupaoedu")
public class AppConfig  {
}
```

在上面的例子中，com.gupaoedu包首先会被扫到，然后再容器内查找被@Component声明的类，找到后将这些类按照Spring bean 定义进行注册。

如果你要在你的web应用开发中选用上述的配置的方式的话，需要用AnnotationConfigWebApplicationContext 类来读取配置文件，可以用来配置Spring的Servlet监听器 ContextLoaderListener或者Spring MVC的DispatcherServlet。

```
<web-app>
  <context-param>
    <param-name>contextClass</param-name>
    <param-value>
      org.springframework.web.context.support.AnnotationConfigWebApplicationContext
    </param-value>
  </context-param>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>com.gupaoedu.AppConfig</param-value>
  </context-param>
  <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>
    <init-param>
      <param-name>contextClass</param-name>
      <param-value>
        org.springframework.web.context.support.AnnotationConfigWebApplicationContext
      </param-value>
    </init-param>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>com.gupaoedu.web.MVCConfig</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/web/*</url-pattern>
  </servlet-mapping>
</web-app>
```

2.9. 怎样用注解的方式配置Spring?

Spring在2.5版本以后开始支持用注解的方式来配置依赖注入。可以用注解的方式来替代XML方式的bean描述，可以将bean 描述转移到组件类的内部，只需要在相关类上、方法上或者字段声明上使用注解即可。注解注入将会被容器在XML注入之前被处理，所以后者会覆盖掉前者对于同一个属性的处理结果。

注解装配在Spring中是默认关闭的。所以需要在Spring文件中配置一下才能使用基于注解的装配模式。

如果你想要在你的应用程序中使用关于注解的方法的话，请参考如下的配置。

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

在标签配置完成以后，就可以用注解的方式在Spring 中向属性、方法和构造方法中自动装配变量。

下面是几种比较重要的注解类型：
1.@Required：该注解应用于设值方法。
2.@Autowired：该注解应用于有值设值方法、非设值方法、构造方法和变量。
3.@Qualifier：该注解和@Autowired 注解搭配使用，用于消除特定bean自动装配的歧义。

4.JSR-250 Annotations:Spring 支持基于JSR-250注解的以下注解，@Resource、@PostConstruct和@PreDestroy。

2.10. 清解释Spring Bean 的生命周期?

Spring Bean的生命周期简单易懂。在一个bean实例被初始化时，需要执行一系列的初始化操作以达到可用的状态。同样的，当一个bean不在被调用时需要进行相关的析构操作，并从bean 容器中移除。

Spring bean factory 负责管理在spring 容器中被创建的bean的生命周期。Bean的生命周期由两组回调（call back）方法组成。
1.初始化之后调用的回调方法。
2.销毁之前调用的回调方法。

Spring 框架提供了以下四种方式来管理bean的生命周期事件：

1、InitializingBean 和DisposableBean回调接口
2、针对特殊行为的其他Aware接口
3、Bean 配置文件中的Custom init()方法和destroy()方法
4、@PostConstruct 和@PreDestroy 注解方式

使用customInit()和customDestroy()方法管理bean生命周期的代码样例如下：

```
<beans>
   <bean id="demoBean" class="com.gupaoedu.task.DemoBean"
        init-Method="customInit" destroy-Method="customDestroy">
   </bean>
</beans>
```


2.11. Spring Bean作用域之间的区别?

Spring 容器中的bean可以分为5个范围。所有范围的名称都是自说明的，但是为了避免混淆，还是让我们来解释一下：

1.singleton：这种bean范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个bean的实例，单例的模式由 bean factory 自身来维护。

2.prototype：原形范围与单例范围相反，为每一个bean 请求提供一个实例。

3.request：在请求bean范围内会每一个来自客户端的网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。

4.Session：与请求范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

5.global-session: global-session和Portlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多portlet。如果你想要声明让所有的portlet 共用全局的存储变量的话，那么这全局变量需要存储在global-session中。

全局作用域与Servlet中的session作用域效果相同。

2.12. 什么是Spring inner beans?

在Spring框架中，无论何时bean被使用时，当仅被调用了一个属性。一个明智的做法是将这个bean声明为内部 bean。内部bean可以用 setter注入“属性”和构造方法注入“构造参数”的方式来实现。比如在我们的应用程序中；一个Customer 类引用了一个Person类我们的要做的是创建一个Person的实例，然后在Customer内部使用。

```
public class Customer{
   private Person person;
}
public class Person{
   private String name;
   private String address;
   private int age;
}
```

内部bean的声明方式如下：

```
<bean id="CustomerBean" class="com.gupaoedu.common.Customer">
   <property name="person">
      <bean class="com.gupaoedu.common.Person">
         <property name="name" value="lokesh" />
         <property name="address" value="India" />
         <property name="age" value="34" />
      </bean>
   </property>
</bean>
```

2.13. Spring框架中的单例Beans是线程安全的吗?

Spring框架并没有对单例bean进行任何多线程的封装处理。关于单例bean的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的Spring bean 并没有可变的状态（比如Service类和DAO类），所以在某种程度上说Spring的单例bean是线程安全的。如果你的bean有多种状态的话（比如View Model对象），就需要自行保证线程安全。

最浅显的解决办法就是将多态bean的作用域由"singleton"变更为"prototype"。

2.14. 请举例说明如何在Spring 中注入一个Java集合?

Spring 提供了以下四种集合类的配置元素：

1、`<list>`：该标签用来装配可重复的list值。

2、`<set>`：该标签用来装配没有重复的set值。

3、`<map>`：该标签可用来注入键和值可以为任何类型的键值对。

4、`<props>`：该标签支持注入键和值都是字符串类型的键值对。

下面看一下具体的例子：

```
<beans>
   <bean id="javaCollection" class="com.gupaoedu.JavaCollection">
      <property name="customList">
         <list>
            <value>INDIA</value>
            <value>Pakistan</value>
            <value>USA</value>
            <value>UK</value>
         </list>
      </property>
      <property name="customSet">
         <set>
            <value>INDIA</value>
            <value>Pakistan</value>
            <value>USA</value>
            <value>UK</value>
         </set>
      </property>
      <property name="customMap">
         <map>
            <entry key="1" value="INDIA"/>
            <entry key="2" value="Pakistan"/>
            <entry key="3" value="USA"/>
            <entry key="4" value="UK"/>
         </map>
      </property>
      <property name="customProperies">
         <props>
            <prop key="admin">admin@gupaoedu.com</prop>
            <prop key="support">support@gupaoedu.com</prop>
         </props>
      </property>
   </bean>
</beans>
```

2.15. 如何向Spring Bean 中注入java.util.Properties?

第一种方法是使用如下面代码所示的标签：

```
<bean id="adminUser" class="com.gupaoedu.common.Customer">
   <property name="emails">
      <props>
         <prop key="admin">admin@gupaoedu.com</prop>
         <prop key="support">support@gupaoedu.com</prop>
      </props>
   </property>
</bean>
```

也可用"`util:`"命名空间来从properties文件中创建出一个propertiesbean，然后利用 setter 方法注入bean的引用。

2.16. 请解释Spring Bean 的自动装配?

在Spring框架中，在配置文件中设定bean的依赖关系是一个很好的机制，Spring容器还可以自动装配合作关系bean之间的关联关系。这意味着Spring 可以通过向Bean Factory中注入的方式自动搞定bean之间的依赖关系。自动装配可以设置在每个bean上，也可以设定在特定的bean上。
下面的XML配置文件表明了如何根据名称将一个bean设置为自动装配：

```
<bean id="employeeDAO" class="com.gupaoedu.EmployeeDAOImpl" autowire="byName" />
```

除了bean配置文件中提供的自动装配模式，还可以使用@Autowired 注解来自动装配指定的bean。

在使用@Autowired 注解之前需要在按照如下的配置方式在Spring 配置文件进行配置才可以使用。

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

也可以通过在配置文件中配置 AutowiredAnnotationBeanPostProcessor 达到相同的效果。

```
<bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
```

配置好以后就可以使用@Autowired来标注了。

```
@Autowired
public EmployeeDAOImpl ( EmployeeManager manager ) {
     this.manager = manager;
}
```

2.17. 自动装配有哪些局限性?

自动装配有如下局限性：

重写：你仍然需要使用和`<property>`设置指明依赖，这意味着总要重写自动装配。

原生数据类型：你不能自动装配简单的属性，如原生类型、字符串和类。

模糊特性：自动装配总是没有自定义装配精确，因此，如果可能尽量使用自定义装配。

2.18. 请解释各种自动装配模式的区别?

在Spring框架中共有5种自动装配，让我们逐一分析。

1.no：这是Spring框架的默认设置，在该设置下自动装配是关闭的，开发者需要自行在bean定义中用标签明确的设置依赖关系。

2.byName：该选项可以根据 bean名称设置依赖关系。当向一个bean中自动装配一个属性时，容器将根据bean的名称自动在在配置文件中查询一个匹配的bean。如果找到的话，就装配这个属性，如果没找到的话就报错。

3.byType：该选项可以根据bean类型设置依赖关系。当向一个bean中自动装配一个属性时，容器将根据bean的类型自动在在配置文件中查询一个匹配的bean。如果找到的话，就装配这个属性，如果没找到的话就报错。

4.constructor：造器的自动装配和byType模式类似，但是仅仅适用于与有构造器相同参数的bean，如果在容器中没有找到与构造器参数类型一致的bean，那么将会抛出异常。

5.autodetect：该模式自动探测使用构造器自动装配或者byType自动装配。首先，首先会尝试找合适的带参数的构造器，如果找到的话就是用构造器自动装配，如果在bean 内部没有找到相应的构造器或者是无参构造器，容器就会自动选择byType的自动装配方式。

2.19. 请举例解释@Required 注解?

在产品级别的应用中，IOC容器可能声明了数十万了bean，bean与bean之间有着复杂的依赖关系。

设值注解方法的短板之一就是验证所有的属性是否被注解是一项十分困难的操作。可以通过在中设置"dependency-check"来解决这个问题。

在应用程序的生命周期中，你可能不大愿意花时间在验证所有 bean的属性是否按照上下文文件正确配置。或者你宁可验证某个bean的特定属性是否被正确的设置。即使是用“dependency-check”属性也不能很好的解决这个问题，在这种情况下，你需要使用@Required 注解。

需要用如下的方式使用来标明bean的设值方法。

```
public class EmployeeFactoryBean extends AbstractFactoryBean<Object> {
   private String designation;
   public String getDesignation() {
      return designation;
   }
   @Required
   public void setDesignation(String designation) {
      this.designation = designation;
   }
}
```

RequiredAnnotationBeanPostProcessor 是Spring中的后置处理用来验证被@Required 注解的 bean 属性是否被正确的设置了。在使用RequiredAnnotationBeanPostProcessor 来验证 bean 属性之前，首先要在IOC容器中对其进行注册：

```
<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" />
```

但是如果没有属性被用@Required 注解过的话后置处理器会抛出一个BeanInitializationException 异常。

2.20. 请求举例说明@Qualified注解?

@Qualifier注解意味着可以在被标注bean的字段上可以自动装配。Qualifier注解可以用来取消Spring不能取消的bean应用。

2.21. 构造方法注入和设值注入有什么区别?

请注意以下明显的区别：
1.在设值注入方法支持大部分的依赖注入，如果我们仅需要注入int、string和long型的变量，我们不要用设值的方法注入。对于基本类型，如果我们没有注入的话，可以为基本类型设置默认值。在构造方法注入不支持大部分的依赖注入，因为在调用构造方法中必须传入正确的构造参数，否则的话会报错。

2.设值注入不会重写构造方法的值。如果我们对同一个变量同时使用了构造方法注入又使用了设置方法注入的话，那么构造方法将不能覆盖由设值方法注入的值。很明显，因为构造方法尽在对象被创建时调用。

3.在使用设值注入时有可能还不能保证某种依赖是否已经被注入，也就是说这时对象的依赖关系有可能是不完整的。而在另一种情况下，构造器注入则不允许生成依赖关系不完整的对象。

4.在设值注入时如果对象A和对象B互相依赖，在创建对象A时Spring会抛出BeanCurrentlyInCreationException异常，因为在B对象被创建之前A对象是不能被创建的，反之亦然。所以Spring 用设值注入的方法解决了循环依赖的问题，因对象的设值方法是在对象被创建之前被调用的。

2.22. Spring框架中有哪些不同类型的事件?

Spring的ApplicationContext 提供了支持事件和代码中监听器的功能。

我们可以创建bean用来监听在ApplicationContext中发布的事件。ApplicationEvent类和在ApplicationContext 接口中处理的事件，如果一个bean实现了ApplicationListener接口，当一个ApplicationEvent 被发布以后，bean会自动被通知。

```
public class AllApplicationEventListener implements ApplicationListener<ApplicationEvent> {
   @Override
   public void onApplicationEvent(ApplicationEvent applicationEvent) {
      //process event
   }
}
```

Spring 提供了以下5种标准的事件：

1.上下文更新事件（ContextRefreshedEvent）：该事件会在ApplicationContext 被初始化或者更新时发布。也可以在调用ConfigurableApplicationContext接口中的refresh()方法时被触发。
2.上下文开始事件（ContextStartedEvent）：当容器调用ConfigurableApplicationContext的Start()方法开始/重新开始容器时触发该事件。
3.上下文停止事件（ContextStoppedEvent）：当容器调用ConfigurableApplicationContext的Stop()方法停止容器时触发该事件。

4.上下文关闭事件（ContextClosedEvent）：当ApplicationContext 被关闭时触发该事件。容器被关闭时，其管理的所有单例Bean都被销毁。
5.请求处理事件（RequestHandledEvent）：在Web应用中，当一个http请求（request）结束触发该事件。

除了上面介绍的事件以外，还可以通过扩展ApplicationEvent类来开发自定义的事件。

```
public class CustomApplicationEvent extends ApplicationEvent {
   public CustomApplicationEvent ( Object source, final String msg ){
      super(source);
      System.out.println("Created a Custom event");
   }
}
```

为了监听这个事件，还需要创建一个监听器：

```
public class CustomEventListener implements ApplicationListener < CustomApplicationEvent >{
   @Override
   public void onApplicationEvent(CustomApplicationEvent applicationEvent) {
   }
}
```

之后通过applicationContext 接口的publishEvent()方法来发布自定义事件。

```
1 CustomApplicationEvent customEvent = new CustomApplicationEvent(applicationContext, "Te
2 applicationContext.publishEvent(customEvent);
```

2.23. FileSystemResource和ClassPathResource有何区别?

在FileSystemResource 中需要给出spring-config.xml文件在你项目中的相对路径或者绝对路径。在ClassPathResource 中spring 会在ClassPath中自动搜寻配置文件，所以要把ClassPathResource文件放在ClassPath下。

如果将spring-config.xml保存在了src文件夹下的话，只需给出配置文件的名称即可，因为src文件夹是默认。

简而言之，ClassPathResource 在环境变量中读取配置文件，FileSystemResource在配置文件中读取配置文件。

2.24. Spring 框架中都用到了哪些设计模式?

Spring框架中使用到了大量的设计模式，下面列举了比较有代表性的：
1、代理模式: 在AOP和remoting中被用的比较多。
2、单例模式：在spring 配置文件中定义的bean默认为单例模式。
3、模板模式：用来解决代码重复的问题。比如: RestTemplate，JmsTemplate，JpaTemplate。

4、委派模式：Spring 提供了DispatcherServlet 来对请求进行分发。
5、工厂模式：BeanFactory用来创建对象的实例，贯穿于BeanFactory/ApplicationContext接口的核心理念。
6、代理模式：AOP思想的底层实现技术，Spring中采用JDK Proxy和CgLib类库。

2.25. 在Spring框架中如何更有效的使用JDBC?

使用Spring JDBC框架，资源管理以及错误处理的代价都会减轻。开发人员只需通过 statements和queries 语句从数据库中存取数据。Spring 框架中通过使用模板类能更有效的使用JDBC，也就是所谓的JdbcTemplate。

2.26. 请解释下Spring 框架中 的IOC容器?

Spring中的org.springframework.beans 包和org.springframework.context包构成了Spring框架IOC容器的基础。

BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。

ApplicationContex 接口对BeanFactory（是一个子接口）进行了扩展，在BeanFactory的基础上添加了其他功能，比如与Spring的AOP更容易集成，也提供了处理message resource的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对Web应用的WebApplicationContext。

2.27. 在Spring中可以注入null或空字符串吗?

完全可以。