1、Spring 包含的模块有哪些？

1）核心容器：

1>spring-core 包括*控制反转*和*依赖注入*；

2> spring-beans 提供了 *BeanFactory*；

3> spring-context 提供*框架式对象访问方式*；

2）AOP：

1>spring-aop 模块以*动态代理技术*实现*面向切面的编程框架*；

2>spring-aspects 模块提供了 *AspectJ(早期实现)的集成*功能；

3）数据访问：

1>spring-jdbc 模块提供 JDBC 抽象层*简化 JDBC 编程*；

2>spring-orm 模块*集成 JPA 和 Hibernate*；

3>spring-tx 模块：提供*声明式事务*；

4）web：spring-webmvc 模块/spring-web 模块

2、Spring,Spring MVC,Spring Boot 间啥关系？

Spring 中有很多功能模块，Spring-Core 是 IOC 依赖注入功能，是**其他功能的基础依赖**；Spring MVC 是 Spring 中模块之一，帮助 Spring 快速**构建** MVC **架构**(模型/视图/控制器)的 **Web 程序**；**Spring Boot 简化 Spring 开发**，仅仅只是减少配置文件。

3、谈谈自己对于 Spring IoC 的了解

IoC 控制反转：是一种设计思想并非技术实现。主要思想就是 1）将**对象创建的控制权交给** *IoC 容器来管理*（原在程序中手动创建对象）；2）将**对象之间的相互依赖关系**也**交给 IoC 容器来管理**，并由 **IoC 容器完成对象的注入**。

>>这样我们**需要创建对象**的时候，只需要**配置***注解/配置文件*即可，无需考虑对象怎么被创建以及对象之间的依赖关系。

【IoC 容器】负责实例化、配置和组装 Bean 组件的核心容器。通过配置**元数据（XML 文件/注解/Java 配置类）**来定义需要管理的对象(即 Bean)

4、什么是 Spring Bean？

Bean 指的就是那些**被 IoC 容器所管理的对象**。通过配置**元数据（XML 文件/注解/Java 配置类）**来定义需要管理的对象。

5、将一个类声明为 Bean 的注解有哪些?

1）@Component：通用的注解，可**标注任意类为 Spring 组件**。不知道属于哪层的时候用。

2）@Repository：持久层*用于数据库相关操作*。

3）@Service：服务层*用 Dao 层实现业务逻辑*。

4）@Controller：控制层*接受用户请求*并*调用 Service 层返回数据*给前端页面。

6、@Component 和@Bean 的区别是什么？

1)@Component *类级别*注解：作用在类上；

@Bean *方法级别*注解：作用在方法上；

2) @Component 通过@ComponentScan *类路径扫描自动装配*到 Spring 容器中；

@Bean 在@Configuration 类中*手动配置返回某个类的实例对象*，装配到 Spring 容器中；

>>因此，@Component 适用于**无需额外初始化**操作更简单；@Bean *自定义性更强*，**在创建 Bean 对象时可进行额外操作**（初始化等逻辑）。

7、注入 Bean 的注解有哪些？有啥区别

@Autowired / @Resource / @Inject(基本不用)

@Autowired：

1>*Spring 内置*的注解；

2>默认注入方式：*类型匹配*；nofind→byName

3>多个实现类时，*@Qualifier* 显示指定名称；

4>可用在*属性*、*setter 方法*、*构造方法*、*构造方法参数*上；

@Resource：

1>*JDK 内置*的注解；（JDK 扩展包中）

2>默认注入方式：*名称匹配*；nofind →byType

3>多个实现类时，*(name)属性*显示指定名称；

4>可用在*属性*、*setter 方法*上。

8、Bean 的作用域有哪些？（@Scope）

1）singleton：IoC 容器中*只有唯一的 bean 实例*。Spring 中的 bean **默认都是单例**的。

2）prototype：*每次获取都会创建一个新的 bean 实例*。即连续 getBean()两次，是不同 Bean。

（下面 3-6 仅在 Web 应用中可用，可用于用一个 Web 应用程序中多个组件之间的访问）

3）request：*每一次 HTTP 请求*都会产生一个新 bean**（请求 bean）**，该 bean 仅在当前 HTTP request 内有效。

4）session：每一次*来自新 session 的 HTTP 请求*都会产生一个新 bean**（会话 bean）**，该 bean 仅在当前 HTTP session 内有效。

5）application/global-session：每个 *Web 应用在启动时*创建一个 bean**（应用 bean）**，该 bean 仅在当前应用启动时间内有效。

6）websocket：*每次 WebSocket 会话*产新 bean。

9、Bean 是线程安全的吗？

*有没有资源竞争*取决于其*作用域*和*状态*。

1）**prototype 中线程安全：**每次获取都会创建一个新 bean 实例，*不存在资源竞争*问题；

2）**singleton** 中只有唯一 bean 实例，*可能有资源竞争*；**→看有无状态：**

>有状态 bean（**有可变成员变量**）**→不安全**；

>无状态 bean（如 Dao/Service）**→**安全；

【解决】1>*尽量避免*可变成员变量；2>在类中*定义一个 ThreadLocal 成员变量*，将需要的可变成员变量*保存在 ThreadLocal 中*。

10、Bean 的生命周期了解么？

1）创建 bean 实例：IoC 容器→配置文件中 bean 的定义→**Java 反射 API** 创建 bean 实例；

2）bean 属性赋值：设置相关属性和依赖；如@Value 注入的值，@Autowired/@Resource 注入的对象和资源；

3）bean 初始化前 1：检查是否实现 **Aware** 的相关接口并设置相关依赖：如

>BeanNameAware 接口→传入 Bean 的名字

>BeanClassLoaderAware → ClassLoader 对象

>BeanFactoryAware 接口→ BeanFactory 对象

4）bean 初始化前 2：BeanPostProcessor **前置处理方法** postProcessBeforeInitialization()；

5）bean 初始化：

> **InitializingBean 接口→** afterPropertiesSet()；

>**配置文件**定义 **init-method 属性方法**；

6）bean 初始化后：BeanPostProcessor **后置处理方法** postProcessAfterInitialization()；

7）使用前：注册销毁的相关调用接口；

8）使用；

9）使用后销毁：

bean→实现 **DisposableBean 接口→**destroy()；

bean→**配置文件**定义 **destroy-method 属性方法**或者**@PreDestroy 标记的方法**。

11、谈谈自己对于 AOP 的了解

1）面向切面编程→***【统一提取】***业务无关的附加功能（分散、重复不易管理）→***【动态插入】***到每个业务方法中→**减少重复代码**，解耦；可以在**不修改原来代码**的基础上**加新功能**。

2）**基于动态代理**的方法实现：

1>目标类**实现了接口**→***JDK 动态代理***：目标类的*接口实现*→动态生成代理对象；（有共同接口，拜把子）

2>目标类**没实现接口**→***cglib***：*继承*目标类→子类作为代理对象；（继承关系，认干爹）

3>也可以使用 *AspectJ*（早期实现框架）；

12、Spring AOP 和 AspectJ AOP 有什么区别？

Spring AOP 属于*运行时*增强，基于*代理实现*；而 AspectJ 是*编译时*增强，基于*字节码操作*。

13、AspectJ 定义的通知类型有哪些？

14、多个切面的执行顺序如何控制？

1）注解@Order(数值)：数值越小→优先级高；

2）实现 Ordered 接口重写 getOrder()方法：return 数值；

15、Spring MVC 的核心组件有哪些？

1）DispatcherServlet：核心中央处理器→负责**请求接收/分发**，并**返回客户端响应**。

2）HandlerMapping：处理器映射器→根据*URL* 去匹配*查找 Handler*，并会将请求涉及到的*拦截器和 Handler* 一起封装。

3）HandlerAdapter：处理器适配器→根据 HandlerMapping 找到的 Handler，适配*执行对应的 Handler*；

4）Handler：请求处理器→处理请求的处理器。

5）ViewResolver：视图解析器→根据 *Handler 返回的逻辑视图/视图*，解析并*渲染真正的视图*，并*传递给 DispatcherServlet 响应*客户端。

16、SpringMVC 工作原理了解吗？

17、全局异常处理怎么做？

18、什么是 SpringBoot 自动装配？

传统 Spring 开发：需要**手动配置每个 bean** 和**大量配置文件**；

而 SpringBoot 自动装配机制：**扫描类路径（找到需要的组件）→条件注解/自动配置类（完成自动配置 bean）**。

19、SpringBoot 是如何实现自动装配的？

【前提】@SpringBootApplication 组合注解：

>@Configuration：允许*注册额外的 bean* 或*导入其他配置类*；

>@EnableAutoConfiguration：启用 SpringBoot *自动配置机制*；

>@ComponentScan：扫描*被@Component 注解*的 bean（**默认扫描启动类所在包下所有类**）

【@EnableAutoConfiguration】自动装配：

@EnableAutoConfiguration 只是一个简单的注解，核心 ***AutoConfigurationImportSelector 类***实现 *ImportSelector 接口*中的 ***selectImports 方法***，通过*获取所有需要装配的全限定类名*，调用 ***getAutoConfigurationEntry()***方法加载自动装配类；

【getAutoConfigurationEntry()方法内部】

1>判断自动装配*是否打开*；

2>去除 EnableAutoConfiguration 中*排除的类*；

3>获取需*要自动装配*的所有配置类文件 ***META-INF/spring.factories***；

4>通过*条件注解@ConditionalOnClass* 等所有条件都满足才会加载。（并非一启动全加载）

20、Spring 循环依赖了解吗，怎么解决？

>***Bean 对象循环引用***，如两个或多个 Bean 间相互持有对方引用。单对象自我依赖。

>解决：*三级缓存→循环依赖下*未初始化完的 bean 对象*提前暴露*出去*供后者属性注入*。

1）一级缓存<BeanName, Bean>：

初始化好的 bean；（平常的单例 bean 都是在这获取的）

2）二级缓存<BeanName, Bean>：

属性未填充的 bean；（与三级缓存配合使用，可以防止 AOP 情况下，每次调用都产生新的代理对象）

3）三级缓存<BeanName, ObjectFactory>：

存放对象工厂；（ObjectFactory 生成原始对象或代理对象）

【解决循环依赖的流程】

1）A 和 B 循环依赖；

2）spring 创建 A，A 依赖 B；

3）spring 创建 B，B 依赖 A；

4）这时找一/二级缓存都没有 A；

5）从*三级缓存*中用 ***ObjectFactory 创建 A 的前期暴露对象*** bean 放到二级缓存中，***删除三级缓存***中的这个 ObjectFactory；

6）***B 从二级缓存中注入***属性未填充的 A；

7）B 完成初始化后放到***一级缓存，A 注入 B***；

21、@Lazy 能解决循环依赖吗？

@Lazy 用来标识*类*是否需要*懒加载/延迟加载*。

> 被标记的 bean 在 IoC 容器启动时***不会立即实例化***，而是***在第一次被请求时***才创建；

>A 和 B，在 **A 的构造器上加@Lazy**；延迟 B

> spring **创建 A 的 bean 时**，需要注入 B；

> 由于 A 上*有@Lazy*，spring ***创建 B 的代理对象***，并***注入 A 中***；

>**创建 B 的 bean 时**，***A 已创建***，**注入 A**。