

# Spring Interview Questions and Answers

## Spring Core, IoC, and Dependency Injection

### 1. What is Spring Framework

Spring 是一个开源 Java 框架，用来简化企业级开发。它通过 IoC/DI 把对象创建和依赖管理交给容器处理，让我们更专注业务逻辑，同时提供 AOP、事务、数据访问、MVC 等模块化能力，提升可维护性、可测试性和扩展性。

记忆钩子

Spring = 管基础设施，让你写业务。

### 6. Inversion of Control(IoC)

10岁小孩版

以前你要喝水：**你自己去厨房找杯子、找水。**

IoC 后：你说“我要水”，**管家（容器）把水递给你。**

控制权从“你自己做”变成“容器帮你做”，所以叫“控制反转”。

IoC 是把对象创建、组装依赖、生命周期管理等控制权从代码里转移到 Spring 容器。这样组件之间更松耦合，更容易替换实现、写单测和扩展。

IoC 容器：BeanFactory vs ApplicationContext（好背版）

- BeanFactory：最基础，主要管“创建对象 + 注入”
- ApplicationContext：更高级，除了上面，还支持  
事件机制、国际化 i18n、资源加载、更多扩展点、BeanPostProcessor 等（面试就说“功能更全”）

记忆钩子

IoC：我不自己 new，让容器管。

常见坑

- IoC 是**理念/机制**，DI 是**实现方式之一**（别说反了）。

### 7. Dependency Injection(DI)

你要做汉堡（类 A），需要肉饼（类 B）。

你不要自己养牛做肉饼（自己 new B），而是**厨房直接把肉饼放到你手里**（注入）。

面试 30 秒标准回答

DI 是 IoC 的具体实现方式：由容器在运行时把依赖对象注入到组件中，而不是组件内部自己创建依赖。好处是松耦合、可替换实现、易测试。

## 三种注入（面试最推荐怎么说）

### 1. Constructor injection（最推荐）

- 依赖必需、不可变、利于测试

```
@Service
public class UserService {

    private final UserRepository userRepository;

    // 构造器注入
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public void doSomething() {
        userRepository.save();
    }
}
```

### 2. Setter injection

- 依赖可选、可后期修改

### 3. Field injection（不推荐）

- **反射**注入，测试/可见性差，不利于不可变设计

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public void doSomething() {
        userRepository.save();
    }
}
```

记忆钩子

DI：容器把“需要的零件”塞进来。

常见追问

- “为什么推荐构造器注入？”（必需依赖明确、可 final、单测方便）
- “多个实现怎么注入？”（@Qualifier / @Primary）

## 8. Metadata Types

### 10岁小孩版

你要让班主任知道“谁是班干部、谁负责什么”，就要给名单和规则。  
这些“名单/规则”就是 metadata（元数据）。

Spring 通过元数据描述 Bean 的定义与装配方式，常见形式包括 **注解**、XML、Java **配置类**、以及 **properties/yml** 等属性源，用来控制组件扫描、Bean 创建和配置注入。

四类（你写的都对，面试怎么讲更像“体系化”）

- **Annotations**：@Component / @Service / @Repository / @Controller
- **XML**：传统方式（现在少但仍可能遗留系统用）
- **Java Config**：@Configuration + @Bean
- **Property sources**：application.properties / application.yml + 环境变量等

记忆钩子

**元数据** = “告诉容器怎么装配”的说明书。

## 9. Spring Beans and scopes

### 10岁小孩版

“玩具”是大家共用一份，还是每个人发一个？这就是 scope。

Spring Bean 的 scope 决定同一个 Bean 在容器里创建多少个实例。最常用的是 **singleton**（默认）和 **prototype**。在 Web 环境还有 request/session 等作用域。

#### 1. Singleton (单例 - 默认)

- **定义**：在每个 Spring IoC 容器中，该 Bean 只有一个共享的实例。
- **记忆钩子**：全班共用一本书。
- **例子**：

```
@Service // 默认就是 Singleton
public class LibraryService {
    // 所有人拿到的都是同一个 LibraryService 对象
}
```

#### 2. Prototype (原型)

- **定义**：每次向容器请求（注入或 `getBean`）该 Bean 时，都会创建一个新的实例。
- **重点纠正**：它不代表“每次 HTTP 请求”，而是“**每次调用**”。
- **记忆钩子**：每次借书都给你一本印好的新书。
- **例子**：

```
@Component
@Scope("prototype")
public class TaskToken {
    // 每次在其他地方 @Autowired 这个 Bean，都会得到不同的实例
}
```

#### 3. Request (请求 - 仅限 Web 环境)

- **定义**：在一个 HTTP 请求内，该 Bean 有一个实例。请求结束后实例销毁。

- **记忆钩子：**每次进教室（发起请求）发一份讲义。
- **例子：**

```
@Component
@WebScope // 或者 @Scope(WebApplicationContext.SCOPE_REQUEST)
public class UserRequestLogger {
    // 记录单次 HTTP 请求的信息
}
```

#### 4. Session (会话 - 仅限 Web 环境)

- **定义：**在一个 HTTP Session 生命周期内，该 Bean 有一个实例。
- **记忆钩子：**同一个同学这学期（Session）领一份学生证。
- **例子：**

```
@Component
@SessionScope
public class UserCart {
    // 存储该用户当前登录会话的购物车
}
```

## 10. Bean LifeCycle:

10岁小孩版

一个“员工（bean）”进公司：

入职（创建）→培训（初始化）→上班（使用）→离职（销毁）

Bean 生命周期一般是：**实例化** → **属性注入** →（各种 Aware 回调可选）→ **BeanPostProcessor 前置处理** → **初始化**（@PostConstruct / InitializingBean / init-method）→ **BeanPostProcessor 后置处理** → **可用** → 容器关闭时 **销毁**（@PreDestroy / DisposableBean / destroy-method）。

**你写的版本 OK，但面试更“标准顺序”是上面这个**

你可以记 4 个关键词：

Instantiate → Populate → Initialize → Destroy

## 11. Autowiring

Autowiring 是 Spring 根据规则自动把依赖注入进来，常见按 **类型（by type）**、**名称（by name）**、**构造器（constructor）** 等方式。现代 Spring 主要用注解方式按类型注入，遇到多个候选时用 @Qualifier 或 @Primary 来消歧。

# Spring Boot

## 12. What is Spring Boot

Spring 是“零件仓库 + 说明书”，你要自己把电脑组起来。

Spring Boot 是“**整机**”：零件自动配好，还自带电源和开机按钮（内置服务器），你按一下就跑。

Spring Boot 是基于 Spring 的快速开发框架，通过 **自动配置**（auto-configuration）、**starter 依赖** 和 **内置服务器**（如 Tomcat），让我们用更少配置更快启动应用，特别适合微服务和云原生部署。

## 13. Spring vs Spring Boot

Spring 是一个完整的生态框架，提供 IoC、AOP、MVC、事务等能力；Spring Boot 是在 Spring 之上做的“脚手架”，核心是 **自动配置** + **starter** + **内置容器**，目标是让 Spring 应用更快搭建、更少样板配置。

## 14. Common Spring Boot Annotations

“Spring Boot 最核心的是 **@SpringBootApplication**，它是一个复合注解，主要包含三个部分：

- **@Configuration**：标记当前类为配置类；

- 一句话版

👉 告诉 Spring：这个类里定义的是“怎么创建 Bean”

它具体干啥？

- 把当前类当成 Spring 的配置类
- 允许你在里面写：

```
@Bean
public DataSource dataSource() {
    return new HikariDataSource();
}
```

- Spring 启动时会：
  - 扫描这个类
  - 执行这些 **@Bean** 方法
  - 把返回值注册进 Spring 容器

- **@EnableAutoConfiguration**：开启自动配置，根据类路径下的 jar 包自动注入默认配置（Spring Boot 的灵魂）；

- 👉 根据你“引了什么依赖”，自动帮你把 Bean 配好

- 它到底自动了啥？

举几个真实例子 👉

你引了：

```
spring-boot-starter-web
```

它自动：

- 配 `DispatcherServlet`
- 配 `Tomcat`
- 配 `Jackson`
- 配 `HandlerMapping`
- `@ComponentScan`：自动扫描并加载符合条件的组件（如 `@Service`, `@RestController`）。”
  - 默认扫描：

```
@SpringBootApplication
public class App {}
```

👉 App 所在包 + 子包

能扫到的包括：

- `@Component`
- `@Service`
- `@Repository`
- `@Controller`
- `@RestController`

扫描后干啥？

- 实例化这些类
- 放进 Spring 容器
- 你就可以：

```
@Autowired
UserService userService;
```

## 常用注解速记

- **依赖注入**：用 `@Autowired` 或构造器注入，将 Bean 注入容器。
- **控制层**：用 `@RestController`，它是 `@Controller` 和 `@ResponseBody` 的组合，默认返回 JSON。
- **手动定义 Bean**：在 `@Configuration` 类中使用 `@Bean` 方法。

## 15. Spring Boot Configuration Types

Spring Boot 配置主要来自 `application.properties/yml`、环境变量、命令行参数以及外部配置中心等。默认配置来自 auto-configuration；当我们在本地配置文件或环境里提供同名配置时，会覆盖默认值。生产环境常通过环境变量、Config Server、K8s ConfigMap/Secret 等方式做外部化配置。

## 16. Tomcat Role in Spring Boot Application

Tomcat 就像**餐厅门口的服务员 + 前台**：

负责接客人（HTTP request），把单子交给厨房（你的 Spring MVC/Controller），再把菜端回去（HTTP response）。

Spring Boot 默认带 embedded Tomcat，它本质是一个 Servlet Container/Web Server。主要负责：监听端口、接收 HTTP 请求、管理连接/线程、把请求交给 Spring 的 `DispatcherServlet`（再路由到 Controller），并把响应返回给客户端。生产中也可以换成 Jetty/Undertow。

Tomcat = **接收请求 + 转交 Spring + 返回响应**。

## 17) Profiles in Spring Boot

同一款游戏有**新手模式/困难模式**：规则不一样，但游戏还是同一个。

Profile 就是让你的程序在 dev/staging/prod 用不同配置。

Profile 用来按环境切换配置，比如 dev/staging/prod。常见做法是 `application-dev.yml`、`application-prod.yml` 分文件配置，并用 `spring.profiles.active` 激活（可通过环境变量、命令行参数、配置文件）。代码层也能用 `@Profile("dev")` 控制哪些 Bean 在某个环境才生效。

Profile = **同一套代码，不同环境不同配置**。

## 18) Actuator and its usage in Spring Boot

Actuator 就像给汽车装了**仪表盘 + 体检接口**：你能看到油量、速度、发动机温度（健康、指标、线程、环境等）。

Spring Boot Actuator 提供一组运维/监控端点（通常在 `/actuator/*`），用于查看应用健康状态、指标、环境配置、线程 dump、日志级别等。常用端点有 `/actuator/health`、`/actuator/metrics`、`/actuator/info`。生产环境一般会配合 Micrometer/Prometheus，并用安全策略限制暴露端点。

Actuator = **监控/运维端点集合**。

## AOP, Hibernate, JDBC

### 19. Spring AOP and proxy pattern

#### 1. 为什么需要 AOP？（痛点）

如果你要在每个功能执行前都检查一下：“**用户登录了吗？**”，执行后记录一下：“**操作耗时多久？**”。

如果没有 AOP，你的代码长这样：

- **下单**：检查登录 + **下单逻辑** + 记录日志
- **取消**：检查登录 + **取消逻辑** + 记录日志
- **查看**：检查登录 + **查看逻辑** + 记录日志

**麻烦在哪？** 1. 你的业务逻辑（下单）被这些“杂事”（登录、日志）淹没了。 2. 万一哪天要改日志格式，你要改 100 个地方。

## 2. AOP 是怎么做的？（上帝视角）

AOP 的核心思想是：“**横向抽取**”。

它就像一个“**拦截器**”或者“**安检口**”。你只需要写好核心业务（下单、取消），然后告诉 Spring：

“在所有 **Service** 方法执行前，帮我统一跑一下『检查登录』；执行完后，统一跑一下『记录日志』。”

Spring AOP 用来处理 **cross-cutting concerns**（日志、事务、安全、缓存等）。Spring AOP 的实现主要基于 **proxy**：通过 JDK dynamic proxy 或 CGLIB 在运行时创建代理对象，拦截方法调用，在调用前/后/环绕插入切面逻辑。

## 20. Key components of AOP

- Aspect：一套“规矩”（比如“每次下单都要记日志”）
- Advice：规矩具体怎么做
- Pointcut：哪些地方要执行规矩
- Join Point：具体的那个“时刻/地点”（比如某个方法被调用）

AOP 关键概念：

- Aspect：切面，封装横切逻辑
- Advice：通知（Before/After/AfterReturning/AfterThrowing/Around）
- Pointcut：切点表达式，决定拦截哪些 join points
- Join Point：可被拦截的执行点（在 Spring AOP 里主要是方法执行）
- Weaving：织入，把切面应用到目标对象（Spring AOP 多为运行时通过代理织入）

## 23. Hibernate ORM

**Hibernate** 是一个 ORM 框架，把 Java 对象映射到数据库表（entity ↔ table）。在 Spring 里最常见是通过 **JPA** 使用：Hibernate 常作为 JPA 的实现，配合 Spring Data JPA 进行 CRUD、查询、事务管理等。

Hibernate = 对象 ↔ 表的翻译官（ORM）。

## 24) Hibernate Validator Framework & HibernateTemplate

10岁小孩版

Validator 就像“入场检查”：不合格（空、太短、邮箱格式不对）就不让进。

Hibernate Validator 是 Bean Validation (Jakarta Validation) 的参考实现，通过注解如 `@NotNull`、`@Size`、`@Email` 定义约束，并配合 `@Valid` / `@Validated` 在 Controller/service 层自动校验，防止非法数据进入业务逻辑。

`HibernateTemplate` 是早期封装 Hibernate 操作的模板类，但现在更推荐 Spring Data JPA 或直接用 `EntityManager`。

## 25) Spring JDBC API and its classes

JDBC 原本像“自己拧很多螺丝”：开连接、关连接、处理异常、读结果集。

Spring JDBC 像“给你电动螺丝刀”，把重复步骤都帮你做了。

Spring JDBC 提供对 JDBC 的抽象，减少样板代码（连接管理、异常转换、资源关闭）。核心是 `JdbcTemplate`（执行 SQL、查询映射），常配合 `DataSource`（连接池/连接来源）。另外还有 `NamedParameterJdbcTemplate`（命名参数更易读）、`SimpleJdbcCall`（调用存储过程）等。

`JdbcTemplate` = 少写样板 JDBC。

## 27) Fetching records using Spring `JdbcTemplate`

你问数据库“给我用户表”，它给你一堆行。你需要一个“翻译器”把每一行翻译成 `User` 对象。

用 `JdbcTemplate` 查询数据通常用 `query()`：传入 SQL + 参数（可选）+ `RowMapper`（把每行变成对象）。例如：

```
jdbcTemplate.query("select * from users", new BeanPropertyRowMapper<>(User.class));
```

如果只要一条记录用 `queryForObject()`；更复杂映射可以用自定义 `RowMapper` 或 `ResultSetExtractor`。

更“面试靠谱”的代码（建议你背这一版）

```
List<User> users = jdbcTemplate.query("SELECT id, name FROM users WHERE status=?", (rs, rowNum) -> new User(rs.getLong("id"), rs.getString("name")), "ACTIVE");
```

`query` = 多行；`queryForObject` = 单行；`RowMapper` = 行→对象。

## Spring MVC

### 28) Spring MVC and its components

Spring MVC 是 Spring 的 Web 框架，采用 MVC 思想把请求处理分层。核心组件：

`DispatcherServlet` 作为前端控制器接收所有请求；根据 `HandlerMapping` 找到对应 Controller；Controller 处理业务并返回 `Model + View`（或直接返回 JSON）；再由 `ViewResolver` 解析视图并渲染响应。

（注意：面试说到 `HandlerMapping` / `ViewResolver` 会显得你更懂“内部流程”。）

MVC: DispatcherServlet 统一入口, Controller 处理, Model 传数据, View 渲染。

## 29) DispatcherServlet and Request Flow in Spring MVC

DispatcherServlet 是 Spring MVC 的前端控制器。请求流程一般是：

1. 客户端请求到达 DispatcherServlet
2. 它通过 HandlerMapping 找到匹配的 Controller/Handler
3. 通过 HandlerAdapter 调用 Controller 方法
4. Controller 返回 ModelAndView 或返回数据对象
5. 若是视图：用 ViewResolver 找 View 并渲染；若是 REST：通过 HttpMessageConverter（如 Jackson）把对象序列化成 JSON
6. 返回响应给客户端

DS → Mapping → Adapter → Controller → (ViewResolver 或 MessageConverter) → Response

## 30) Interceptors (Spring MVC Interceptor)

进电影院前有安检：

可以在你进门前检查（pre），看完出来时统计（post/after）。这就是 interceptor。

Spring MVC 的 Interceptor 用来在请求进入 Controller 前后做统一处理（例如鉴权、日志、限流、埋点）。它通常实现 HandlerInterceptor，提供 preHandle（进 Controller 前）、postHandle（Controller 执行后、渲染前）、afterCompletion（请求完成后，用于清理资源/记录耗时）。它作用于 Controller 调用链，比 Servlet Filter 更贴近 Spring MVC。

## 32. Most important Spring MVC annotations

Spring MVC 常用注解：

- @Controller / @RestController：声明控制器（后者默认返回 JSON）
- @RequestMapping / @GetMapping / @PostMapping：映射路径和 HTTP 方法
- @RequestParam：取 query 参数
- @PathVariable：取路径参数
- @RequestBody：取 JSON body 并反序列化
- @ModelAttribute：绑定表单数据到对象/把对象放入 Model
- @ExceptionHandler / @ControllerAdvice：异常处理

### 33) Importance of session scope

Session 就像“你在网站里的一张临时会员卡”。同一个人多次操作都能认出你是谁。

Session scope 用于在同一个用户会话期间共享状态数据（如登录态、购物车）。它能避免每个请求重复创建/加载同一份会话数据。但要注意：在现代系统里更推崇 **无状态（stateless）**，登录态通常用 JWT/Token + Redis 等方式管理，尽量减少把大量状态塞进 session 造成内存压力和集群粘性問題。

### 35) Data validation

Spring 常用校验是 Bean Validation（Jakarta Validation）：在 DTO 上写 `@NotNull/@Size/@Email`，Controller 参数加 `@Valid` 或 `@Validated` 触发校验，失败会抛异常（如 `MethodArgumentNotValidException`）。复杂场景可自定义 `Validator` 或自定义约束注解，并配合全局异常处理返回统一错误格式。

### 36) BeanFactory vs ApplicationContext

BeanFactory 像“只会发工具的仓库”。

ApplicationContext 像“仓库 + 管理系统 + 广播系统 + 国际化 + 资源中心”。

`BeanFactory` 是最基础的 IoC 容器能力（创建 Bean、依赖注入）。`ApplicationContext` 是更完整的容器：提供事件机制、国际化、资源加载、更多扩展点，并且在常见场景下默认更常用。实际开发一般直接使用 `ApplicationContext`。

BeanFactory=基础；ApplicationContext=全家桶。

### 38) Exception Handling in Spring MVC

出了错不要让页面爆炸，要有“统一的客服中心”把错误翻译成用户看得懂的提示。

Spring MVC 异常处理常用三层：

1. 控制器内 `@ExceptionHandler` 处理特定异常
2. 全局 `@ControllerAdvice` + `@ExceptionHandler` 统一处理（最常用）
3. 结合 Spring Boot 的 error 机制（`/error`、`ErrorController`）或自定义错误响应。REST 项目通常返回统一的 JSON 错误结构和合适的 HTTP 状态码。

### 39) ViewResolver

Controller 说“去 主页”，ViewResolver 负责把“主页”翻译成真正的页面文件/模板。

`ViewResolver` 负责把 Controller 返回的 **逻辑视图名**（如 `"home"`）解析成具体 View（JSP/Thymeleaf/Freemarker 等），然后由 View 去渲染 Model。REST API（`@RestController`）通常不需要 ViewResolver，因为直接走消息转换返回 JSON。

👉 Model 是“Controller 往 View 里塞数据用的容器”

👉 View 负责“怎么展示 Model 里的数据”

# Spring Boot Interview Questions and Answers

## Spring Boot Interview Questions for Freshers

---

### 1. What is Spring Boot?

Spring Boot 是基于 Spring 的快速开发框架，用来更快地构建可运行的 Java 应用。它通过 **auto-configuration**、**starter dependencies** 和 **embedded server**（如 Tomcat/Jetty/Undertow）减少繁琐配置，让应用可以一键启动，并提供 **production-ready** 能力（例如 Actuator、外部化配置等）。

### 2. What are the Features of Spring Boot?

Spring Boot 的核心特性包括：

1. **Auto-configuration**：根据 classpath 依赖自动装配默认 Bean
2. **Starter dependencies**：用 starter 一键引入一整套常用依赖
3. **Embedded server**：默认可用 Tomcat 直接启动
4. **Externalized configuration**：`application.yml/properties`、env、命令行等统一配置
5. **Actuator**：监控与运维端点
- 6) (可选) **CLI**：命令行快速跑 demo（现在企业里用得相对少）

### 4. Define the Key Components of Spring Boot.

Spring Boot 的核心组件可以概括为：

- **Starter dependencies**（依赖组合包）
- **Auto-configuration**（自动装配）
- **SpringApplication**（启动引导）
- **Embedded server**（内置容器）
- **Actuator**（监控运维）
- **Externalized configuration**（外部化配置）

### 5. Why do we prefer Spring Boot over Spring?

相比“纯 Spring”，Spring Boot 更受欢迎主要因为它：

- 默认约定好大量配置（**convention over configuration**）
- 提供 **auto-configuration + starters**，减少手动 XML/Java 配置
- 默认内置 web server，打包即可运行
- 提供 Actuator 等生产能力，方便上线运维

## 7. What are the Spring Boot Starter Dependencies?

Starter 是一组常用依赖的集合，帮助快速引入某类能力。常见 starter：

- `spring-boot-starter-web` (Spring MVC + embedded Tomcat 默认等)
- `spring-boot-starter-data-jpa` (JPA/Hibernate + 数据访问)
- `spring-boot-starter-security` (Spring Security)
- `spring-boot-starter-test` (测试框架集合)
- `spring-boot-starter-thymeleaf` (模板引擎)

## 8. How does a spring application get started?

Spring Boot 通过 `main()` 调用 `SpringApplication.run()` 启动。它会：

1. 创建并准备 `ApplicationContext`
2. 加载配置 (properties/yml、env、命令行)
3. 扫描组件并完成 Bean 初始化 (IoC/DI)
4. 刷新 context，启动 `embedded server` (Web 应用场景)
5. 应用开始对外提供服务

## 9. What does the @SpringBootApplication annotation do internally?

`@SpringBootApplication` 等价于：

`@Configuration + @EnableAutoConfiguration + @ComponentScan`

它让当前类成为配置入口，开启自动装配，并从当前包及子包扫描组件，完成应用引导启动。

## 10. What is Spring Initializr?

Spring Initializr (常用 [start.spring.io](https://start.spring.io)) 是用来生成 Spring Boot 项目骨架的工具。你选择构建工具 (Maven/Gradle)、Java 版本、依赖 (web/jpa/security...)，它会生成项目结构和配置文件，帮助快速开始开发。

Initializr = 勾选依赖 → 生成骨架工程。

# Spring Boot Intermediate Interview Questions

## 12. What are the basic Spring Boot Annotations?

常用 Spring Boot/MVC 注解：

- `@SpringBootApplication`：启动入口 (`@Configuration + @EnableAutoConfiguration + @ComponentScan`)
- `@Configuration` / `@Bean`：声明配置类和手动注册 Bean
- `@Component` / `@Service` / `@Repository`：组件扫描注册 Bean (语义更清晰)

- `@RestController` : REST 控制器 (默认 `@ResponseBody` 返回 JSON)
- `@RequestMapping` + `@GetMapping/@PostMapping...` : 请求路径与方法映射
- `@RequestParam/@PathVariable/@RequestBody` : 参数绑定 (面试很常问)

### 13. What is Spring Boot dependency management?

Spring Boot 的 dependency management 指的是: Boot 通过 BOM/依赖管理 (Maven 的 `dependencyManagement` 或 Gradle 的平台) 为一大堆常用库提供 **兼容的默认版本**, 避免版本冲突。我们引入 starter 时通常不需要手写版本号, Boot 会用与当前 Boot 版本匹配的依赖版本组合。

### 14. Is it possible to change the port of the embedded Tomcat server in Spring Boot?

可以。最常用是在 `application.properties/yml` 设置: `server.port=8081`。也能通过环境变量或命令行覆盖, 例如 `--server.port=8081`。

### 15. What is the starter dependency of the Spring boot module?

Starter 是为某个功能场景打包的一组依赖集合 (例如 web、jpa、security)。它的价值是 **少配依赖**、版本更稳定, 并且往往会触发相应的 auto-configuration。

### 16. What is the default port of Tomcat in spring boot?

- 8080

### 18. How to disable a specific auto-configuration class?

可以通过 `exclude` 禁用某个 auto-configuration:

- 在启动类: `@SpringBootApplication(exclude = XxxAutoConfiguration.class)`  
或: `@EnableAutoConfiguration(exclude = XxxAutoConfiguration.class)`  
也可以在配置里用 `spring.autoconfigure.exclude=...` (写类全名) 做禁用。

### 20. Describe the flow of HTTPS requests through the Spring Boot application.

HTTPS 请求流程:

1. 客户端与服务器先进行 **TLS 握手** (证书校验/密钥协商), 建立加密通道
2. 加密的 HTTP 请求到达 Tomcat (**Servlet 容器**)
3. Tomcat 把请求交给 Spring MVC 的 `DispatcherServlet`
4. `DispatcherServlet` 通过映射找到对应 `Controller`
5. `Controller` 调用 `Service` (业务逻辑), 必要时调用 `Repository/DAO` 访问数据库 (JPA/JdbcTemplate 等)
6. 返回响应: REST 通常经 `HttpMessageConverter` 序列化为 JSON; 如果是传统网页才会走 `ViewResolver` 渲染模板
7. 响应通过 TLS 通道加密返回客户端

## 21. Explain @RestController annotation in Spring Boot.

`@RestController` = `@Controller` + `@ResponseBody`。它表示这个 Controller 的返回值默认写入 HTTP response body，通常配合 Jackson 自动序列化成 JSON，用于构建 REST API。

## 22. Difference between @Controller and @RestController

`@Controller` 通常配合 ViewResolver 返回页面视图名；如果要返回 JSON，需要方法上加 `@ResponseBody`。  
`@RestController` 默认所有方法都等价于加了 `@ResponseBody`，更适合 REST API（直接返回 JSON/XML）。

Controller 默认页面；RestController 默认数据。

## 23. What is the difference between RequestMapping and GetMapping?

`@RequestMapping` 是通用映射，可指定 method (GET/POST...)；  
`@GetMapping` 是 `@RequestMapping(method = GET)` 的语法糖，更简洁明确。类似还有  
`@PostMapping`/`@PutMapping`/`@DeleteMapping`。

## 24. What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?

`@SpringBootApplication` 是启动类最常用注解，包含 `@EnableAutoConfiguration` 并且还包含  
`@ComponentScan` 与 `@Configuration`。  
`@EnableAutoConfiguration` 只负责开启自动配置，不会自动做组件扫描。一般我们直接用  
`@SpringBootApplication`；只有在更细粒度拆分配置时才单独使用/组合它。

## 25. What are Profiles in Spring?

Profile 用于按环境切换配置 (dev/test/prod)。可以用 `application-dev.yml`、`application-prod.yml` 分环境配置，Bean 层面用 `@Profile("dev")` 控制加载。激活通过 `spring.profiles.active=prod`（配置文件/环境变量/命令行）完成。

## 26. Mention the differences between WAR and embedded containers.

WAR 是部署到外部 Servlet 容器（外部 Tomcat/WebLogic 等）的打包方式；应用依赖容器提供运行环境。

Embedded container（Boot 常见）是把服务器（Tomcat/Jetty/Undertow）和应用一起打包成可执行 jar，启动更简单，适合容器化和微服务部署。

Boot 也可以打 WAR，但主流是可执行 jar + embedded。

# Spring Boot Interview Questions For Experienced

---

## 27. What is Spring Boot Actuator?

Spring Boot Actuator 提供一组 production-ready 的监控与运维端点（通常在 `/actuator/*`），用来查看应用健康状态、指标、环境配置、线程 dump、日志级别等。使用时通常引入 `spring-boot-starter-actuator`，并根据需要暴露/保护端点，常结合 Micrometer + Prometheus/Grafana 做监控。

## 29. What is the purpose of using @ComponentScan in the class files?

`@ComponentScan` 用于指定 Spring 扫描哪些包，自动发现并注册组件（`@Component`/`@Service`/`@Repository`/`@Controller` 等）到容器。默认情况下，`@SpringBootApplication` 自带 `@ComponentScan`，会扫描启动类所在包及其子包。需要跨包扫描时才显式指定 `basePackages` 或 `basePackageClasses`。

## 30. What are the @RequestMapping and @RestController annotations in Spring Boot used for?

`@RequestMapping` 用来把 HTTP 请求映射到 Controller 方法，可按路径、HTTP method、参数、header 等匹配；`@GetMapping`/`@PostMapping`... 是它的快捷形式。

`@RestController` = `@Controller` + `@ResponseBody`，表示方法返回值直接写入响应体（通常 JSON），用于 REST API。

Mapping 管路由；RestController 管“返回数据”。

## 34. What is dependency Injection and its types?

DI 是一种把依赖对象由外部注入而不是内部 `new` 的方式，用来实现松耦合、易测试。常见注入方式：

- Constructor injection (**推荐**)：依赖必需、可 `final`、更易测试
- Setter injection：依赖可选、可后期修改
- Field injection (**不推荐**)：反射注入，测试和可维护性差

## 35. What is an IOC container?

IoC 容器是 Spring 管理对象（Bean）的核心组件：负责 Bean 的创建、依赖注入、生命周期回调、作用域等。常见容器接口是 `BeanFactory`，更常用的是 `ApplicationContext`（功能更全，如事件、资源加载、国际化等）。

IoC 容器 = 创建 + 注入 + 生命周期管理。

## 36. What is the difference between Constructor and Setter Injection?

构造器注入适合“必需依赖”：依赖在创建时就完整，利于不可变设计（可 `final`），更便于测试与保证对象可用性；Setter 注入适合“可选依赖”或需要后期调整的场景，但可能导致对象在某个时刻处于“未完全注入”的不安全状态。

# Bonus Spring Boot Interview Questions and Answers

## 1. What is Thymeleaf?

Thymeleaf 是 Java 的服务端模板引擎，常与 Spring MVC/Spring Boot 配合使用，用于把 Model 数据渲染到 HTML 页面。它的特点是 HTML-friendly（模板本身像正常 HTML），支持表达式、条件、循环、表单绑定等，适合传统服务端渲染（SSR）页面。

Thymeleaf = 服务端把数据塞进 HTML 模板。

## 2. Explain Spring Data and What is Data JPA?

Spring Data 是 Spring 的数据访问抽象层，提供统一的 Repository 编程模型，减少 DAO 样板代码，并集成多种数据源（JPA、Mongo、Redis 等）。

Spring Data JPA 是 Spring Data 的一个子项目，基于 JPA（常用 Hibernate 实现），通过接口继承 `JpaRepository` 就能获得 CRUD、分页排序，并支持方法名派生查询和 `@Query` 自定义查询。

## 3. Explain Spring MVC

Spring MVC 是基于 Spring 的 Web 框架，采用 MVC 模式。核心是 `DispatcherServlet` 作为前端控制器接收请求，根据映射找到 Controller 处理业务，然后返回视图（通过 ViewResolver 渲染）或返回数据（REST 场景通过 MessageConverter 序列化为 JSON）。

## 4. What is Spring Bean?

Spring Bean 是由 Spring IoC 容器创建、管理和装配的对象。容器负责它的生命周期、依赖注入、作用域（singleton/prototype 等）以及各种回调扩展。

## 5. What are Inner Beans in Spring?

Inner Bean 是在某个 Bean 的配置内部定义的嵌套 Bean，通常只用于该外层 Bean 的依赖，不单独暴露为可复用的顶层 Bean。传统 XML 配置里可以在 `<property>` 内嵌套 `<bean>` 定义。现代 Spring Boot 项目更常用注解/ `@Bean`，Inner Bean 在面试里更多是“历史/概念”点。

## 6. What is Bean Wiring?

Bean wiring 指 Spring 把 Bean 之间的依赖关系装配起来的过程。可以是自动装配（`@Autowired`、构造器注入、`@Qualifier`）或显式装配（Java Config `@Bean` 方法传参、XML 显式引用）。目标是让组件松耦合、易测试。

## 9. Mention the steps to connect the Spring Boot application to a database using JDBC.

用 JDBC 连接数据库的典型步骤：

1. 添加依赖：`spring-boot-starter-jdbc` + 对应数据库 driver（MySQL/PostgreSQL 等）
2. 配置连接信息：`spring.datasource.url/username/password/driver-class-name`（以及连接池参数）
3. Spring Boot 会自动创建 `DataSource`，并自动提供 `JdbcTemplate`（通常不需要手动 new）
4. 在代码里注入 `JdbcTemplate` 执行 `query/update`，配合 `RowMapper` 映射结果（可选）
- 5) 配置 schema 初始化/迁移：Flyway 或 Liquibase