- Spring
  - Spring 基础
    - Spring 是一款开源的轻量级 Java 开发框架，旨在提高开发人员的开发效率以及系统的可维护性。
    - IOC&AOP
      - IOC控制反转
        - IoC 是一种设计思想，而不是一个具体的技术实现，解决的是对象管理和对象依赖的问题
        - IoC 的思想就是将原本在程序中手动创建对象的控制权，交由 Spring 框架来管理。
        - 控制 ：指的是对象创建（实例化、管理）的权力；反转 ：控制权交给外部环境（Spring 框架、IoC 容器）
        - 在 Spring 中， IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个 Map（key，value），Map 中存放的是各种对象。
        - DI依赖注入是控制反转的实现方式，对象无需自行创建或者管理它的依赖关系，依赖关系会被自动注入到它们的对象当中去
        - 将对象集中统一管理，工厂模式，降低耦合度
      - AOP面向切面编程
        - AOP 是使用动态代理实现的，解决的是非业务代码抽取的问题
        - AOP能够将与业务无关却为业务模块所共同调用的逻辑或责任（例如事务处理、日志管理、权限控制等）封装起来
        - 面向切面编程其实就是在业务方法前后增加非业务代码
        - 便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可拓展性和可维护性。
        - Spring AOP 属于运行时增强，而 AspectJ 是编译时增强。 Spring AOP 基于代理(Proxying)，而 AspectJ 基于字节码操作(Bytecode Manipulation)。
        - AspectJ 相比于 Spring AOP 功能更加强大，但是 Spring AOP 相对来说更简单
    - Spring Bean
      - bean 代指的就是那些被 IoC 容器所管理的对象。
      - 定义Bean的几种方式 ⊖ xml文件、注解、JavaConfig、基于Groovy DSL配置
      - bean的作用域
        - singleton：唯一 bean 实例，Spring 中的 bean 默认都是单例的，对单例设计模式的应用。
        - prototype：每次请求都会创建一个新的 bean 实例。
        - request：每一次 HTTP 请求都会产生一个新的 bean，该 bean 仅在当前 HTTP request 内有效。
        - session：每一次来自新 session 的 HTTP 请求都会产生一个新的 bean，该 bean 仅在当前 HTTP session 内有效。
        - 配置bean作用域
          - xml文件 ⊖ <bean id="..." class="..." scope="singleton"></bean>
          - 注解 ⊖ @Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
      - 单例bean的线程安全问题
        - 单例 bean 存在线程问题，主要是因为当多个线程操作同一个对象的时候是存在资源竞争的
        - 解决方法
          - 在 bean 中尽量避免定义可变的成员变量。
          - 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
      - 将类声明为bean的注解
        - 一般使用 @Autowired 注解自动装配 bean
        - @Component ：通用的注解，可标注任意类为 Spring 组件。如果一个 Bean 不知道属于哪个层，可以使用@Component 注解标注。
        - @Repository：对应持久层即 Dao 层，主要用于数据库相关操作。
        - @Service：对应服务层，主要涉及一些复杂的逻辑，需要用到 Dao 层。
        - @Controller：对应 Spring MVC 控制层，主要用户接受用户请求并调用 Service 层返回数据给前端页面。
      - @Component 和 @Bean 的区别
        - @Component 注解作用于类，而@Bean注解作用于方法
        - @Component通常是通过类路径扫描来自动侦测以及自动装配到 Spring 容器中，@Bean 注解通常是在标有该注解的方法中定义产生这个 bean
        - @Bean 注解比 @Component 注解的自定义性更强，很多地方我们只能通过 @Bean 注解来注册 bean。比如当我们引用第三方库中的类需要装配到 Spring容器时
      - @Autowired和@Resources
        - 「@Autowired 是spring 的注解」，是 spring2.5 版本引入的，Autowired 只根据 type 进行注入，「不会去匹配 name」。
        - 「@Resource 是 Java 自己的注解」，两个属性是比较重要的，name 和 type；Spring 将 @Resource 注解的 name 属性解析为 bean 的名字，而 type 属性则解析为 bean 的类型
      - Bean的生命周期
        - Spring在启动的时候需要「扫描」在XML/注解/JavaConfig 中需要被Spring管理的Bean信息，将这些信息封装成BeanDefinition，最后放到一个beanDefinitionMap中
        - 实例化 ⊖ 实例化Bean实例，就是遍历BeanDefinitionMap
        - 属性赋值 ⊖ 对象属性依赖注入
        - 初始化
          - 检查Aware相关接口并设置相关依赖
          - BeanPostProcessor前置处理
          - 是否实现InitializingBean接口
          - 是否配置自定义的init-method
          - BeanPostProcessor后置处理
        - 销毁
          - 注册Destruction相关回调接口
          - 使用中
          - 是否实现DisposableBean接口
          - 是否配置自定义的destory-method
      - 循环依赖
        - 主要通过三级的缓存来解决，核心逻辑：把实例化和初始化的步骤分开，放入缓存中
        - A在注入属性时，发现需要依赖B，就会走B的实例化过程，B属性注入依赖A，从三级缓存找到A；然后把三级缓存的A删掉放到二级缓存，B初始化完毕放入一级缓存
        - 接着回来创建A，直接从一级缓存拿到B然后完成创建，放入一级缓存
        - 三级缓存(三个Map)
          - 一级缓存singletonObjects，存放创建好的成品Bean
          - 二级缓存earlySingletonObjects，存放半成品Bean，还没进行属性依赖注入
          - 三级缓存singletonFactories，存的是Bean工厂对象
          - 三级缓存考虑代理，二级缓存考虑性能
    - Spring事务
      - 事务管理方式
        - 编程式事务管理 ⊖ 通过 TransactionTemplate或者TransactionManager手动管理事务
        - 声明式事务管理 ⊖ 实际是通过 AOP 实现（基于@Transactional 的注解方式使用最多）
      - @Transactional 的使用注意事项
        - @Transactional 注解只有作用到 public 方法上事务才生效，不推荐在接口上使用；
        - 避免同一个类中调用 @Transactional 注解的方法，这样会导致事务失效；
        - 正确的设置 @Transactional 的 rollbackFor 和 propagation 属性，否则事务可能会回滚失败
        - 被 @Transactional 注解的方法所在的类必须被 Spring 管理，否则不生效
      - 隔离级别
        - Read Uncommited
        - Read Committed
        - Repeatable Read
        - Serializable
  - SpringMVC
    - MVC 是模型(Model)、视图(View)、控制器(Controller)的简写，其核心思想是通过将业务逻辑、数据、显示分离来组织代码。
    - MVC 是一种设计模式，Spring MVC 是一款很优秀的 MVC 框架。Spring MVC 可以帮助我们进行更简洁的 Web 层的开发，并且它天生与 Spring 框架集成。
    - 工作原理
      - 客户端（浏览器）发送请求，直接请求到 DispatcherServlet
      - DispatcherServlet 根据请求信息调用 HandlerMapping，解析请求对应的Controller。
      - 解析到对应的Controller后，由HandlerAdapter适配器处理，HandlerAdapter会根据Controller来调用处理业务
      - Controller处理完业务后，会返回一个 ModelAndView 对象，Model 是返回的数据对象，View 是个逻辑上的 View
      - ViewResolver 会根据逻辑 View 查找实际的 View
      - DispaterServlet 把返回的 Model 传给 View（视图渲染）
      - 把 View 返回给请求者（浏览器）
  - Spring&SpringBoot注解
    - @SpringBootApplication
      - 可以把 @SpringBootApplication看作是 @Configuration、@EnableAutoConfiguration、@ComponentScan 注解的集合。
      - @Configuration ⊖ 允许在 Spring 上下文中注册额外的 bean 或导入其他配置类
      - @EnableAutoConfiguration ⊖ 启用 SpringBoot 的自动配置机制
      - @ComponentScan ⊖ 扫描被@Component (@Service,@Controller)注解的 bean，注解默认会扫描该类所在的包下所有的类。
    - SpringBean相关
      - @Autowired ⊖ 自动导入对象到类中，被注入进的类同样要被 Spring 容器管理
      - 要想把类标识成可用于 @Autowired 注解自动装配的 bean 的类,可以采用以下注解实现：⊖ @Component、@Repository、@Service、@Controller
      - @RestController ⊖ @RestController注解是@Controller和@ResponseBody的合集，表示这是个控制器 bean，并且是将函数的返回值直接填入 HTTP 响应体中，是 REST 风格的控制器。
      - @Scope ⊖ 声明 Spring Bean 的作用域
      - @Configuration ⊖ 一般用来声明配置类，可以使用 @Component注解替代，不过使用@Configuration注解声明配置类更加语义化。
    - 处理常见的HTTP请求类型
      - @GetMapping("/users") 等价于@RequestMapping(value="/users",method=RequestMethod.GET)
      - @PostMapping("users") 等价于@RequestMapping(value="/users",method=RequestMethod.POST)
      - @PutMapping("/users/{userId}") 等价于@RequestMapping(value="/users/{userId}",method=RequestMethod.PUT)
      - @DeleteMapping("/users/{userId}")等价于@RequestMapping(value="/users/{userId}",method=RequestMethod.DELETE)
    - 前后端传值
      - @PathVariable用于获取路径参数，@RequestParam用于获取查询参数。
      - @RequestBody ⊖ 用于读取 Request 请求（可能是 POST,PUT,DELETE,GET 请求）的 body 部分并且Content-Type 为 application/json 格式的数据，接收到数据之后会自动将数据绑定到 Java 对象上去。
      - 一个请求方法只可以有一个@RequestBody，但是可以有多个@RequestParam和@PathVariable
    - 读取application.yml配置信息
      - 使用 @Value("${property}") 读取比较简单的配置信息：
      - 通过@ConfigurationProperties读取配置信息并与 bean 绑定。
    - 参数校验
      - @NotEmpty 被注释的字符串的不能为 null 也不能为空
      - 常用字段验证
        - @NotBlank 被注释的字符串非 null，并且必须包含一个非空白字符
      - 验证请求体 ⊖ 在需要验证的参数上加上了@Valid注解 @RequestBody @Valid Person person
      - 验证请求参数Path Variables 和 Request Parameters ⊖ Controller类上加上@Validated注解
  - Spring设计模式
    - 工厂设计模式
      - Spring使用工厂模式可以通过 BeanFactory 或 ApplicationContext 创建 bean 对象。
      - 延迟注入(使用到某个 bean 的时候才会注入),相比于ApplicationContext 来说会占用更少的内存，程序启动速度更快。 ⊖ BeanFactory
      - 容器启动的时候，一次性创建所有 bean ，ApplicationContext 扩展了 BeanFactory
      - ApplicationContext
        - ClassPathXmlApplication：把上下文文件当成类路径资源。
        - FileSystemXmlApplication：从文件系统中的 XML 文件载入上下文定义信息。
        - XmlWebApplicationContext：从Web系统中的XML文件载入上下文定义信息。
      - 三个实现类
    - 单例设计模式
      - Spring 通过 ConcurrentHashMap 实现单例注册表的特殊方式实现单例模式
    - 代理模式
      - Spring AOP 把业务模块所共同调用的逻辑封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可拓展性和可维护性。
      - 业务类「必须要实现某个接口」，它是「基于反射的机制实现的」，生成一个实现同样接口的一个代理类，然后通过重写方法的方式，实现对代码的增强。 ⊖ JDK动态代理
      - 使用字节码处理框架 ASM，其原理是通过字节码技术为一个类「创建子类，然后重写父类的方法」，实现对代码的增强。 ⊖ CGLIB代理
    - 模板方法
      - 定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。
      - Spring 中 jdbcTemplate、hibernateTemplate 等以 Template 结尾的对数据库操作的类，它们就使用到了模板方法。
      - 一般情况使用继承的方式来实现模板方法，但是 Spring 使用Callback 模式与模板方法模式配合，既达到了代码复用的效果，同时增加了灵活性。
    - 观察者模式
      - Spring 事件驱动模型就是观察者模式很经典的一个应用。
      - 事件角色
      - 事件驱动模型三种角色
        - 事件监听者角色
        - 事件发布者角色
      - 事件流程
        - 定义一个事件：实现一个继承自ApplicationEvent，并写相应的构造函数
        - 定义一个事件监听者：实现ApplicationListener接口，重写onApplicationEvent方法
        - 使用事件发布者发布消息：通过ApplicationEventPublisher的publishEvent()方法发布消息
    - 适配器模式
      - 适配器模式(Adapter Pattern) 将一个接口转换成客户希望的另一个接口，适配器模式使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。
      - Spring AOP 的增强或通知(Advice)使用到了适配器模式，与之相关的接口是AdvisorAdapter ⊖ Spring AOP中的适配器模式
      - DispatcherServlet 根据请求信息调用 HandlerMapping，解析到请求对应的 Controller后，开始由HandlerAdapter 适配器处理。
      - HandlerAdapter 作为期望接口，具体的适配器实现类用于对目标类Controller进行适配。 ⊖ Spring MVC中的适配器模式
      - 不同类型的 Controller 通过不同的方法来对请求进行处理，如果不利用适配器模式的话，DispatcherServlet 直接获取对应类型的 Controller，需要自行来判断
    - 装饰者模式
      - 装饰者模式可以以动态地给对象添加一些额外的属性或行为。
      - JDK 中比如InputStream 类下有 FileInputStream (读取文件)、BufferedInputStream (增加缓存,使读取文件速度大大提升)等子类都在不修改InputStream 代码的情况下扩展了它的功能。
  - SpringBoot
    - 自动装配
      - SpringBoot 定义了一套接口规范，即：SpringBoot 在启动时会扫描外部引用 jar 包中的META-INF/spring.factories文件，将文件中配置的类型信息加载到 Spring 容器，并执行类中定义的各种操作。
      - 对于外部 jar 来说，只需要按照 SpringBoot 定义的标准，就能将自己的功能装置进 SpringBoot。
      - @SpringBootApplication可以看作是 @Configuration、@EnableAutoConfiguration、@ComponentScan 注解的集合。
    - 如何实现自动装配
      - 启用 SpringBoot 的自动配置机制，是实现自动装配的核心注解
      - @EnableAutoConfiguration
        - AutoConfigurationImportSelector 类实现了 ImportSelector接口，也就实现了这个接口中的 selectImports方法
        - 自动装配核心功能的实现实际是通过 AutoConfigurationImportSelector类
        - selectImports该方法主要用于获取所有符合条件的类的全限定类名，这些类需要被加载到 IoC 容器中。
        - getAutoConfigurationEntry()方法负责加载自动配置类
      - getAutoConfigurationEntry()方法
        - 判断自动装配开关是否打开
        - 获取EnableAutoConfiguration注解中的 exclude 和 excludeName。
        - 获取需要自动装配的所有配置类，读取META-INF/spring.factories
        - @ConditionalOnXXX 中的所有条件都满足，该类才会生效，按需加载而不是每次启动全部加载
      - 允许在上下文中注册额外的 bean 或导入其他配置类 ⊖ @Configuration
      - 扫描被@Component (@Service,@Controller)注解的 bean，注解默认会扫描启动类所在的包下所有的类 ，可以自定义不扫描某些 bean ⊖ @ComponentScan