### 依赖注入技术

依赖注入是一种软件设计模式，通过抽象解耦组件间的依赖关系，实现代码松耦合。本质是将依赖项的创建责任从使用方转移到外部，使代码仅依赖接口而非具体实现。这一设计支持两大原则：依赖倒置原则强调高层模块与底层实现通过抽象交互；开闭原则支持通过扩展而非修改增强功能。松耦合设计带来多重优势：可扩展性支持轻松替换组件；可测试性允许通过模拟依赖隔离单元测试；延迟绑定实现运行时期动态配置依赖；并行开发保障团队独立协作；可维护性确保功能模块清晰隔离。

依赖注入通过特定模式动态注入依赖：

构造函数注入作为首选方式，通过构造函数强制传入依赖项，确保对象初始状态完整

属性注入通过可写属性实现，适用于可选依赖或测试场景

方法注入与环境上下文适用于临时依赖场景

依赖注入主要适用于中大型项目。核心实践原则包括：优先采用构造函数注入保证依赖关系显式可见；谨慎使用属性注入并限定于可选依赖场景；将对象组装逻辑严格限定在应用入口的组合根中；单元测试时通过模拟框架注入虚拟依赖。配置方式需权衡利弊：代码配置提供编译时期安全性，配置文件支持运行时期灵活性但缺乏编译验证。

### 利斯科夫替换原则：

#### 单一职责原则：

强调一个类只应负担一项明确的职责。在文档管理系统中，若将文件上传、物理存储、数据库操作、格式转换、安全扫描等多重功能堆积在一个文档类里，会导致代码混乱且难以维护。应用此原则后，需进行职责划分：单独的文档元数据类仅负责核心属性；专门的仓库类处理数据库交互；独立的文件服务管理磁盘存储；单独的验证类执行安全检查；转换类处理文档格式变换。

#### 开闭原则：

要求软件实体的功能可扩展但不能修改已有代码。在系统的日志记录中，若上传文档服务等方法直接调用具体的文件日志写入功能，日后添加数据库日志或云端日志就需要改动所有业务代码，违背原则。正确做法是定义日志接口规定基本记录方法，并由文件日志器、数据库日志器、云日志器等具体实现。核心的业务服务仅依赖此日志接口。当需新增日志方式时，只需提供新接口实现注入系统，已有业务服务无需任何变更，实现了对新日志渠道的无缝扩展，同时保障了业务逻辑的稳定。

#### 里氏替换原则：

保证派生类可安全替换基类而不影响程序行为。在系统用户权限设计中，基础用户类获取权限方法应返回浏览等基本权限。管理员类若重写此方法，不应遗漏基础权限仅返回管理权限。遵循原则的实现应是管理员类方法先调用基类方法获取基础权限，再添加自身特有权限。这样，在任何使用基础用户对象进行权限校验的地方，传入管理员对象也能正确工作，确保继承关系的合理性以及多态机制的可靠性。

#### 最少知识法则：

约束对象应尽量减少与其他对象的交互深度，只与直接合作的模块通信。在用户界面操作中，若保存按钮处理类直接调用文件存储服务的保存方法甚至操作数据库连接，则暴露了过多底层细节，界面控制器知晓了本不该关心的实现。应用法则后，引入文档管理服务作为中间层，封装保存的所有环节：验证、存储、持久化、日志。界面控制器只与文档管理服务交互。调整保存流程仅需在服务内部修改，界面层完全不受影响，有效降低了模块间的耦合度。

#### 依赖倒转原则：

提出高层逻辑应依赖抽象，具体实现也应依赖抽象。在文档搜索场景，若搜索服务直接使用特定的解析器提取文件内容，将紧密绑定文件格式。遵循原则需定义文本提取接口规定核心方法。搜索服务仅依赖此接口。不同的解析器实现各自格式的处理。当需支持新格式或替换算法时，只需提供新接口实现注入搜索服务，搜索核心逻辑无需修改。这解耦了高层搜索策略与低层解析细节，增强了系统灵活性和可测试性。

#### 合成复用原则：

倡导优先使用组合关系而非继承实现复用。在通知系统中，若通过继承定义基础通知器及其邮件通知器、站内通知器等子类，难以满足灵活组合发送的需求。应用原则应定义通知通道接口规定发送方法，并提供不同通道的实现如邮件通道、站内信通道等。构建通知服务类组合多个通知通道。发布文档的服务仅与通知服务通信。通知服务自身负责通过组合的通道发送消息。这允许运行时自由添加通道，动态配置组合发送方式，结构简洁灵活，避免了继承体系的僵硬和紧耦合。