摘要300+项目背景500+ 正文1200 + 结尾500

1. 微服务架构风格

2. 调用/返回架构风格中的面向对象和层次架构风格（通用核心层，基本应用层，业务管理层，扩展层）

【摘要】：

本文讨论了我所在公司EDI管理系统项目的系统架构。该项目在2020年5月启动，在2021年12月正式上线并维护至今，我在该项目中承担系统架构设计师职务，主要负责需求的总体分析以及应用系统架构的设计。该系统以EDI（电子数据交换）为信息载体，完成公司和客户之间的商业数据的交流，整个业务流程十分复杂，牵涉面相当广泛。

本文以EDI管理系统为例，讨论了软件架构的选择和应用。整个系统采用具有四层的层次式软件架构的设计思想，在业务管理层的设计中，采用了微服务架构风格，将系统划分为应用层，中间层和数据层，其中数据层采用虚拟机风格中的解释器来满足多种数据协议的兼容性需求；中间层采用了数据流、独立构件等多种架构风格以减低系统间耦合度、简化软件架构，提高系统灵活性、可修改性等方面的架构属性；应用层采用了MVC设计模式实现了数据、显示和处理分离。说明架构开发过程中遇到的实际问题和解决方案，最后提出了该系统开发过程的不足和改进方案。

项目经过一年半时间的设计开发并集成到公司的云端中间件解决方案，通过近两年的用户反馈，在不断融合新需求的情况下，整个系统运行稳定，达到了预期的效果和要求。

【正文】：

-项目背景-：

随着全球经济的发展与信息技术的进步，制造业对于高效、准确的业务流程管理需求日益增加。传统的纸质文档处理方式已经无法满足制造业快速发展的需要。因此，许多制造业企业寻求更加高效的电子化、自动化的技术来提高业务流程的效率和准确性。EDI（Electronic Data Interchange，电子数据交换）在制造业中得到广泛应用。EDI通过电子方式在不同的企业之间以自动化流程的方式交换商业文档和数据，其中包括订单、发货通知与收获确认、物流与运输管理、发票与支付处理以及库存管理等方面，提高了供应链的效率、准确性和可靠性。但由于EDI种类众多，即使上下游公司都使用统一的企业资源计划系统，也有很大可能因为EDI不匹配而无法正常交易，因此产生了EDI管理系统的需求，该系统主要功能包括统筹传输数据协议、校验数据、转换数据，数据传输以及发送MDN和返回EDI997 (功能性确认/FA)等功能模块。

我作为部门技术骨干之一，参与了项目的整体架构设计与技术选型以及部分编码工作。下面我将首先详细介绍本系统的分析与设计过程中所采用微服务架构风格的原因。

软件架构风格是描述某一特定应用领域软件系统组织方式的惯用模式。

项目启动后，在架构设计工作的开始阶段，我们便意识到，软件架构风格可以为我们的项目提供架构级的通用解决方案。这种架构级的软件重用可以极大提高我们的系统建设进程。由于toB系统对于安全性、可靠性、可用性和扩展性要求很高，性能等常规系统极为看重的质量属性的重要性反而不高，在与公司技术顾问的商讨后，我们采用微服务架构作为该系统的软件架构体系。最终，我们将系统分为应用层，中间层，数据层。应用层负责具体业务和视图展示，如公司配置、客户配置、模板配置、合同配置以及应用流配置。其又分为视图层与业务逻辑层，采用 MVC 设计模式实现了数据、显示和处理分离，中间层除负责为应用层提供通用服务支持如系统管理服务，session 管理服务等之外，还提供kafka日志、数据推送等中间件支持，其又细分为中间件层、逻辑处理层与数据接口层。而数据层负责提供数据存储访问服务，如数据库服务，缓存服务，文件服务，搜索服务等。接下来，我将分层次详细介绍三层层次体系结构的设计过程

“

应用层：主要采用SpringMVC这一基于J2EE平台的MVC框架，主要利用公司自研基于HTML5的UI开发组件，封装了OpenUI5,jQuery等的前端JS框架。我们将系统根据业务功能进行水平划分，这有助于代码管理与维护。

中间层：采用Spring和Shiro等服务框架实现，随着服务器规模的扩大，开发人员的增多，每个应用都变得复杂臃肿，存在大量代码重复，另一方面随着客户的增多以及各种服务的增加，数据库的连接数压力变大，因为我们采用了服务化方案，即应用层和数据层中增加一个服务层。。从结构上来看，系统结构更为清晰明了，更为立体。稳定性上来看，许多散落的代码成为了通用服务，并交付专门的团队负责运维。出于对成本与技术成熟度的考虑，我们采用了开源的 Spring 框架。通过 Spring 框架的使用，缩短了开发周期，减少了开发费用和维护费用，提高了开发的成功率。另外基于安全性方面考虑，我们选择Apache

Shiro，它是一个功能强大且易于使用的 Java 安全框架，为开发人员提供了一个直观而全面的解决方案，用于身份验证、认证中心、加密、借助于Elasticsearch实现全文索引功能的ELK日志中心。该日志中心不但记录系统的人员登入登出以及对于数据的改动，也记录了UI操作时API的调用并附带有时间戳，极大地方便了对于异常问题地追踪以及回归测试。

通过使用Spring+Apache Shiro的分层设计，实现了各层次低耦合，高内聚,并严格遵循了 web 安全的相关规范

微服务中间层：实现服务的统一管理，为服务间的相互协作运行提供技术保障，是架构实现的技术支撑。包括： 服务注册中心，服务负载均衡。

数据层：数据库采用公司自研的内存列式关系型数据库来负责数据库的持久化存储，。数据层涉及缓存，文件系统，数据库，数据通知服务等模块。我们选择了 MyBatis 作为持久层框架，实现了业务逻辑和数据访问逻辑分离，使系统的设计更清晰，更易维护，更易单元测试，极大提高了系统的可维护性。由于用户对数据的访问具有集中性，为防止频繁读取数据库导致系统性能下降问题，我们选择采用开源的J2Cache 两级缓存框架，即一级采用 Ehcache，二级采用 Redis 缓存技术，由于大量的缓存读取会导致 L2 的网络成为整个系统的瓶颈，因此 L1 的目标是降低对 L2 的读取次数。该缓存框架主要用于集群环境中。单机也可使用，用于避免应用重启导致的 Ehcache 缓存数据丢失。通过该框架的使用，提高了系统性能和访问效率。

具体实施步骤/方法论：

1. 设计阶段：服务划分，按业务独立性和处理共性将服务划分。
2. 应用开发实现：①基于公司原有的构件库，快速搭建②对接现有业务③定制服务，微服务提高系统的灵活性和可扩展性。

在业务需求逐步清晰的情况下，实现功能的快速迭代，开发过程应用了敏捷开发agile/devops。

【结尾】：在\*\*系统架构设计中，我们引入微服务架构的设计思想。根据研究分析把系统设计成应用层，中间层和数据层，重点放在应用层并采用MVC设计模式，负责具体业务处理和视图展示。以上设计有效降低了维护成本，提高系统的开放性。

最终项目成功上线，稳定运行了近一年，达到预期的目标和要求，微服务架构也体现了其良好的可用性，可扩展性以及可重用性，配合敏捷开发来加速项目进度。为后续采用微服务架构开发奠定了基础。在项目开发过程中我们也遇到了一些问题，比如迭代版本后地接口匹配问题，简单交叉变化会改变众多组件，需要协调发布，因此需要服务划分尽量合理，采用公共模板的方式提供底层服务。