WORK LOG 12

Shandong University

May 24, 2020

高德琛

Contents

1 前言

《Work Log 11》将从原理、实施两方面开展对设计方法的讨论。以软件工程原理为原材料，将其运用在 $\mu Vloger$ 项目当中，这是本次报告的整体逻辑。

其中章节 3将概述本课程中设计的设计方法知识，而章节 2.1则是理论在 $\mu Vloger$ 项目上的具现。

2 *μVloger* 设计方法实施

2.1 设计方法

μVloger 系统具有一定的复杂性，对整体的系统逻辑首先需要大致的掌握。为了进行初步的理解，我们从数据流程（图 1）以及业务流程（图 2）开始。

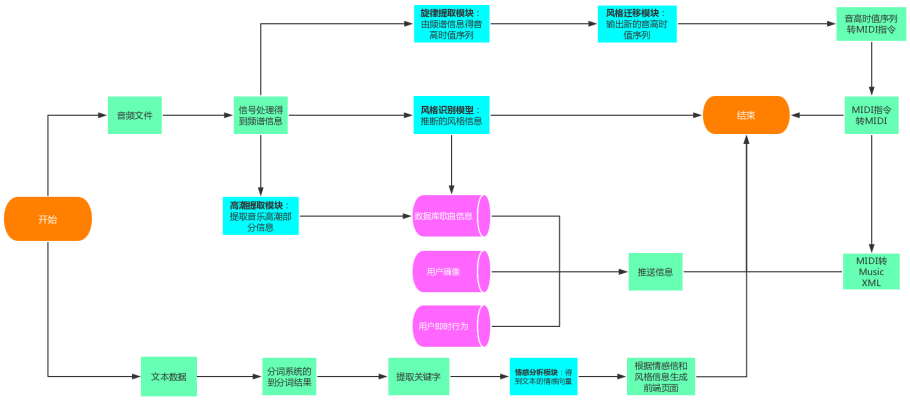

图 **1:** 数据流程

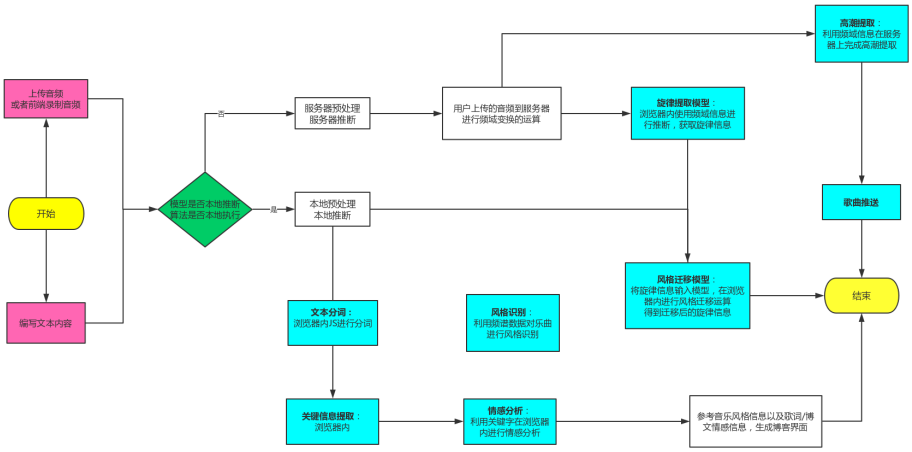

图 **2:** 业务流程

所谓“一图胜千言”，流程图皆为 ProcessOn 绘制而成，相比之前使用的 Visio 而言，ProcessOn 即开即用，便捷可共享。适合快速制图。

我的理解当中，面向对象符合我们思考问题的过程，而面向用户的逻辑本质上也是面向过程的，因为用户的操作是天然具有时序的。利用这种时序的建模也就是业务流程

的来源。

然而，时序的建模并不一定适用于软件工程的设计方法。例如说对于内聚性、耦合性而言，依赖时序的做法显然不够明智。

接下来我们从面向对象的角度来进行设计。

2.2 面向对象设计

3 设计方法笔记

μVloger 是 Web 前端应用，尽管我们采取了前后端分离的开发模型，但是通过后端采用 Node.JS 构建服务器，我们可以进一步降低不同技术、语言带来的学习成本、沟通成本。

不过我们没有全然采纳 JavaScript，深度学习技术的推断、音频处理算法在 Python 下的开发成本更低，而且社区也更为活跃，为我们团队避免了许多潜在的问题。

首先图 3展示了 *μVloger* 整体架构。Node Server、Django Server 分别处理网络业务逻辑、算法及深度学习业务逻辑。由 Nginx 实现反向代理、分布式。

概括而言，我们解耦了前后端逻辑，算法、网络逻辑，数据层、应用层逻辑。

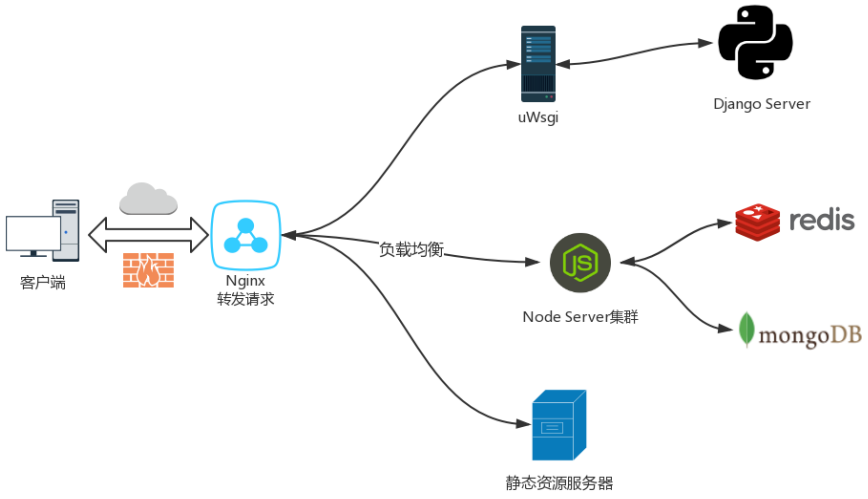

图 **3:** Server 总体架构

对于 Node 服务器的设计，我首先试图引入层次结构。业务逻辑划分为多种微服务，而前端的业务入口可以直接定义为路由。在微服务基础上，我们可以进一步细分得

到多种中间件，中间件按照高内聚、低耦合的设计原则进行划分，其中复用最广泛的，我们称之为“核心中间件”。

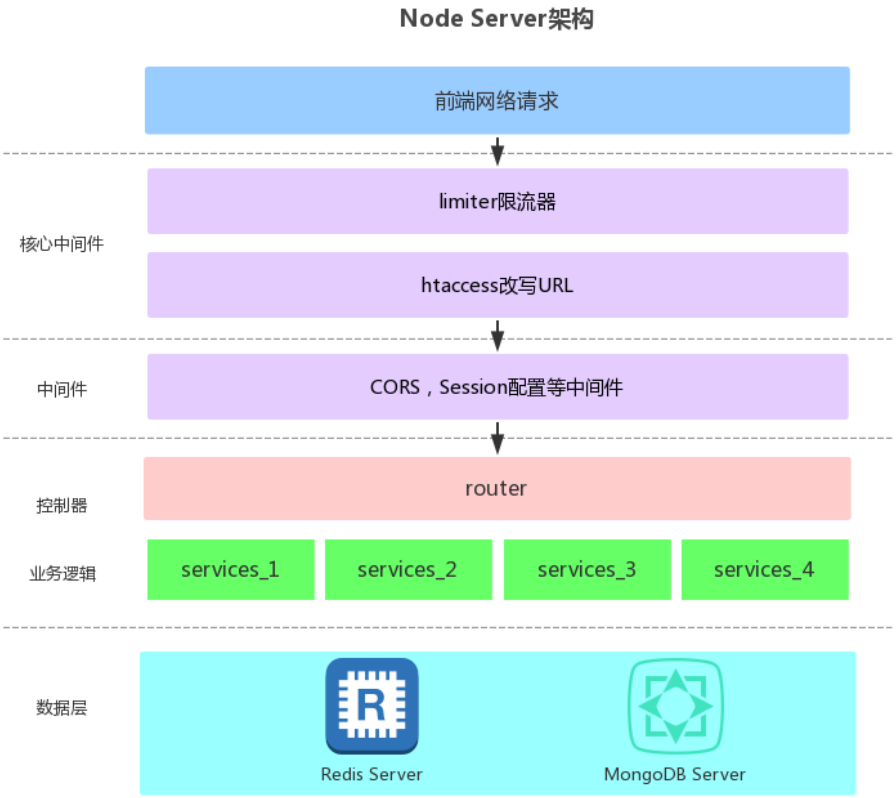

图 **4:** Node Server 架构

3.1 反映系统整体的组织结构和基本特征

1. 软件的层次结构
2. 模块相互作用的方式
3. 全局的、重要的数据变量和数据结构
4. 数据库的逻辑结构
5. 接口

3.2 详细设计

1. 模块逻辑的详细设计
2. 系统数据结构的设计
3. 系统数据库结构的设计
4. 系统-人机接口的设计

3.3 分治

架构设计：将系统分解为一组子系统或模块，以及子系统或模块之间的接口。
详细设计：子系统或模块被再次细分为子模块

3.4 基于抽象的设计原则

1. 里氏替换原则：子类可以替换父类，可以出现在父类能出现的任何地方。
2. 开闭原则：一个软件实体应当对扩展开放，对修改关闭。拓展时不对模块的源码更改，只对拓展出去的部分编写新的代码。
3. 依赖倒转原则：要依赖于抽象，不要依赖于具体类。高级业务逻辑依赖于抽象类，具体类从抽象类泛化而来。
4. 传统软件工程的中的抽象：信息屏蔽、数据局部化、只是对模块细节的封装，没有继承的概念。
5. 面对对象软件工程中的抽象：抽象类、接口、只有方法的特征，没有方法的实现。
6. 接口隔离原则：使用多个专门接口比使用单一的总接口要好很多。

3.5 内聚

度量模块内部各部件间紧密程度、分解时将相关的内容放到一起、确定模块是否合理（部件联系越紧密越好）。内聚的分类如表 1所示。

功能内聚	所有部件处理同一组数据，共同完成单一的功能	最理想的内聚
顺序内聚	各部件之间既有数据联系又有控制联系	较为理想
通信内聚	所有的部件都访问同一组数据，各部件之间只有数据关系，没有控制关系	较为理想
过程内聚	各部件之间只有控制联系，而没有数据联系	较弱内聚
时间内聚	具有时间关系，无数据联系，也无控制联系	内聚更弱
实用程序内聚	部件常常是一些相关的、可重用的实用程序	内聚很弱
偶然内聚	各部件之间没有任何关系	最差的内聚

表 **1:** 内聚分类

3.6 耦合

度量模块间相互相互联系的强弱，耦合越松散越好。耦合态分类如表 2所示。

内容耦合	一个模块直接进入另一个模块中存取其数据或使用其服务（直接在一个类中创建另一个类的对象）	最差的耦合
公共耦合	两个模块间通过一个公共环境进行数据交换	较差的耦合
外部耦合	模块对外部系统有依赖关系	尽量减少使用
控制耦合	两个模块之间通过接口的参数表交换开关数据，旨在控制另一个模块的执行逻辑	较差的耦合
印记耦合	指两个模块之间通过参数交换方式产生耦合，并且交换的是数据结构而不是数据元素（调用另一个模块的公共方法和变量）	允许使用
数据耦合	两个模块之间通过接口的参数表交换信息数据，并且这些信息数据的类型是基本数据类型	允许使用

表 **2:** 耦合分类

3.7 复用

1. 软件复用：重复使用为了复用目的而设计的软件的过程
2. 黑盒复用：不用修改直接复用（无须知道源码直接能使用）
3. 白盒复用：修改后才能使用
4. 代码复制：共享公共函数和子例程

5. 面向对象复用
6. 基于组件复用：组件 (Component) 是指有定义完备接口的、明确规定了上下文依赖关系的合成单元
7. 基于 Web 服务的功能复用

3.8 设计描述方法

1. 层次图：即组成系统的程序模块及其调用关系
2. 结构图：表明模块之间的相互作用关系
3. 盒式图
4. PDL 伪码
5. 程序流程图
6. E-R 图

3.9 面向对象方法

面向对象的分析方法是利用面向对象的信息建模概念，如实体、关系、属性等，同时运用封装、继承、多态等机制来构造模拟现实系统的方法。

传统的结构化设计方法的基本点是面向过程，系统被分解成若干个过程。而面向对象的方法是采用构造模型的观点，在系统的开发过程中，各个步骤的共同的目标是建造一个问题域 zd 的模型。在面向对象的设计中，初始元素是对象，然后将具有共同特征的对象归纳成类，组织类之间的等级关系，构造类库。在应用时，在类库中选择相应的类。

面向来对象方法模型包括六个要素：封装、抽象、模块化、层次结构、类型、并发、持久。

面向对象方法是一种基于对象模型的程序设计方法，包括面向对象分析、面向对象设计、面向对象编程，是目前应用范围最广的设计方法。

面向对象是软件开发方法。面向对象的概念和应用已超越了程序设计和软件开发，扩展到如数据库系统、自交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD 技术、人工智能等领域。面向对象是一种对现实世界理解和抽象的方法，是计算机编程技术发展到一定阶段后的产物。