软件（结构）设计说明

Shandong University

May 5, 2020

CODE & NOTE

Contents

1 说明

1. 《软件 (结构) 设计说明》(SDD) 描述了计算机软件配置项 (CSCI 的设计。它描述了 CSCI 级设计决策、CSCI 体系结构设计 (概要设计) 和实现该软件所需的详细设计。SDD 可用接口设计说明 IDD 和数据库 (顶层) 设计说明 DBDD 加以补充。
2. SDD 连同相关的 IDD 和 DBDD 是实现该软件的基础。向需方提供了设计的可视性，为软件支持提供了所需要的信息。
3. IDD 和 DBDD 是否单独成册抑或与 SDD 合为一份资料视情况繁简而定。

2 引言

2.1 标识

本条应包含本文档适用的系统和软件的完整标识。(若适用) 包括标识号、标题、缩略词语、版本号、发行号。

2.2 系统概述

关键说明：

- 系统名称：基于深度学习的视频配乐自动剪辑系统
- 任务提出者：陈若🄵同学小组
- 本系统将是独立的系统，不产生第三方 API 或 SDK。
- 本系统将使用 Vue 作为前端框架，后端使用 Python 语言；数据库采用 Redis 以及 MongoDB。深度学习框架采用 PyTorch。
- 需要租用的软硬件包括：GPU 型云服务器，测试用途带宽

  本系统为演示系统，没有高带宽以及硬件要求，开发测试阶段采用本地开发以及云服务器测试，支出主要用于 GPU 型云服务器进行深度学习。

2.3 文档概述

软件工程的原则说明“文档先行”的重要性。软件开发本质是人与人的合作，其主体是活生生的“人”。而不同的人自然会有不同的开发理念、习惯。为了在规约上达成一致，形成量化的默契，“文档先行”可以最小化团队的沟通成本。

3 CSCI 级设计决策

本章应根据需要分条给出 CSCI 级设计决策，即 CSCI 行为的设计决策 (忽略其内部实现，从用户的角度看，它如何满足用户的需求) 和其他影响组成该 CSCI 的软件配置项的选择与设计的决策。

如果所有这些决策在 CSCI 需求中均是明确的，或者要推迟到 CSCI 的软件配置项设计时指出，本章应如实陈述。为响应指定为关键性的需求 (如安全性、保密性、私密性需求) 而作出的设计决策，应在单独的条中加以描述。如果设计决策依赖于系统状态或方式，则应指出这种依赖性。应给出或引用理解这些设计所需的设计约定。CSCI 级设计决策的例子如下：

1. 关于 CSCI 应接受的输入和产生的输出的设计决策，包括与其他系统、HWCI, CSCI 和用户的接口 (本文的 4.5.x 标识了本说明要考虑的主题)。如果该信息的部分或全部已在接口设计说明 (IDD) 中给出，此处可引用。
2. 有关响应每个输入或条件的 CSCI 行为的设计决策，包括该 CSCI 要执行的动作、响应时间及其他性能特性、被模式化的物理系统的说明、所选择的方程式/算法/规则和对不允许的输入或条件的处理。
3. 有关数据库/数据文件如何呈现给用户的设计决策 (本文的 4.5.x 标识了本说明要考虑的主题)。如果该信息的部分或全部已在数据库 (顶层) 设计说明 (DBDD) 中给出，此处可引用。
4. 为满足安全性、保密性、私密性需求而选择的方法。
5. 对应需求所做的其他 CSCI 级设计决策，例如为提供所需的灵活性、可用性和可维护性所选择的方法。

4 CSCI 体系结构设计

本章应分条描述 CSCI 体系结构设计。如果设计的部分或全部依赖于系统状态或方式，则应指出这种依赖性。如果设计信息在多条中出现，则可只描述一次，而在其他条引用。应给出或引用为理解这些设计所需的设计约定。

4.1 体系结构

4.1.1 程序（模块）划分

React Ducks 项目文件结构

- 组件分为视图组件 components，逻辑组件 containers
- 视图组件只负责渲染，逻辑组件负责实际的业务逻辑
- 所有组件状态统一使用 Redux 集中到唯一的 Store 管理
- 异步状态全部集中到 Redux-Saga 中间件统一管理
- 各组件使用 index.js 统一暴露接口

React/Redux 文件结构如下

```
components                          全局公用视图组件
    component_1
        component_1.js              视图组件 _1
        index.js
    component_2
containers                          逻辑组件
     container_1                    逻辑组件 _1
        components                  逻辑组件所依赖的视图组件
            component_1.js          视图组件 _1
            component_2.js          视图组件 _2
            index.js
        container_1.js              组件1
        index.js
    container_2
    container_3
```

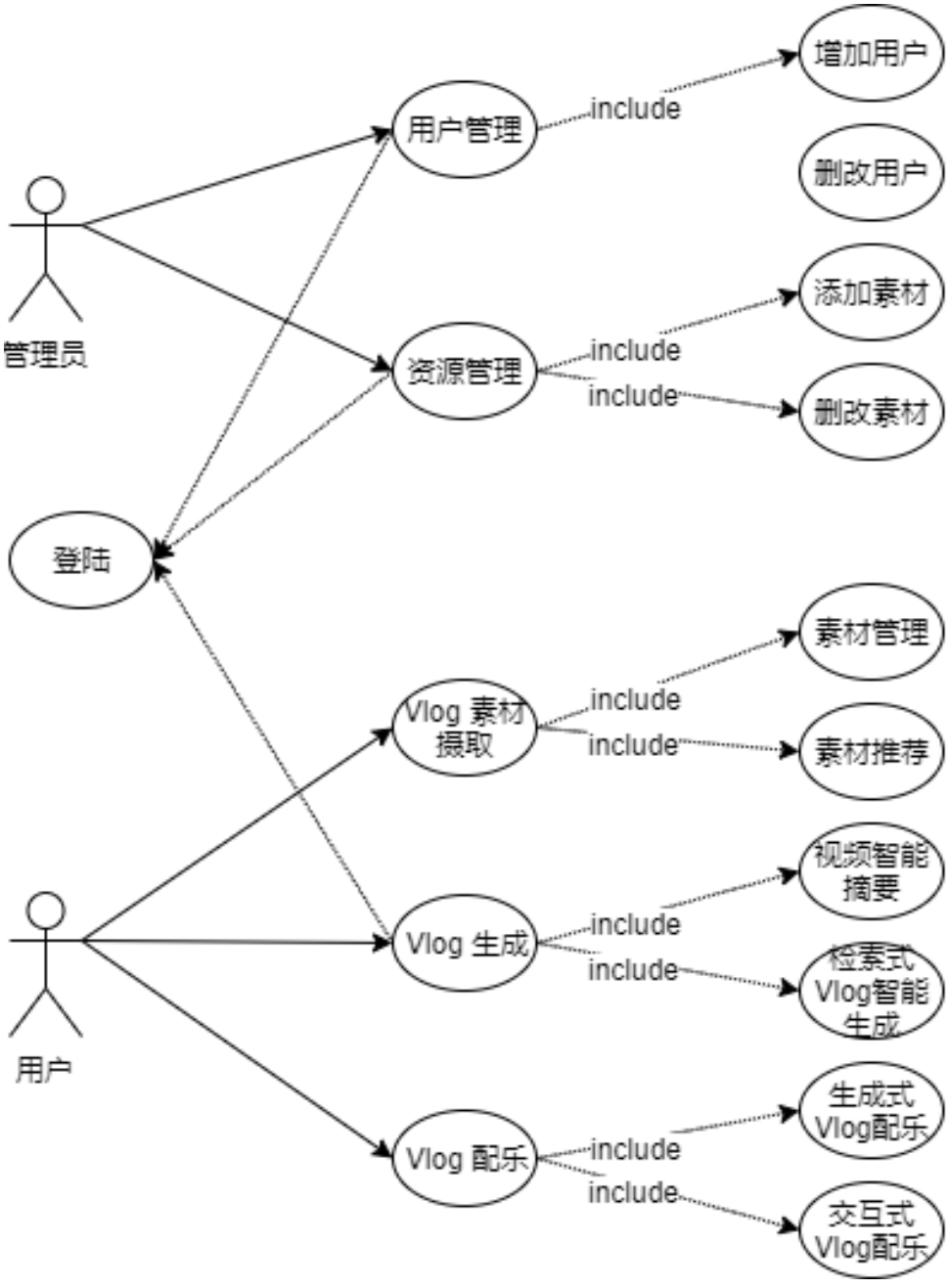
管理员
用户管理
include
增加用户
删改用户
资源管理
include
include
添加素材
删改素材
登陆
用户
Vlog 素材
摄取
include
include
素材管理
素材推荐
Vlog 生成
include
include
视频智能
摘要
检索式
Vlog智能
生成
Vlog 配乐
include
include
生成式
Vlog配乐
交互式
Vlog配乐

图 **1:** 用例图

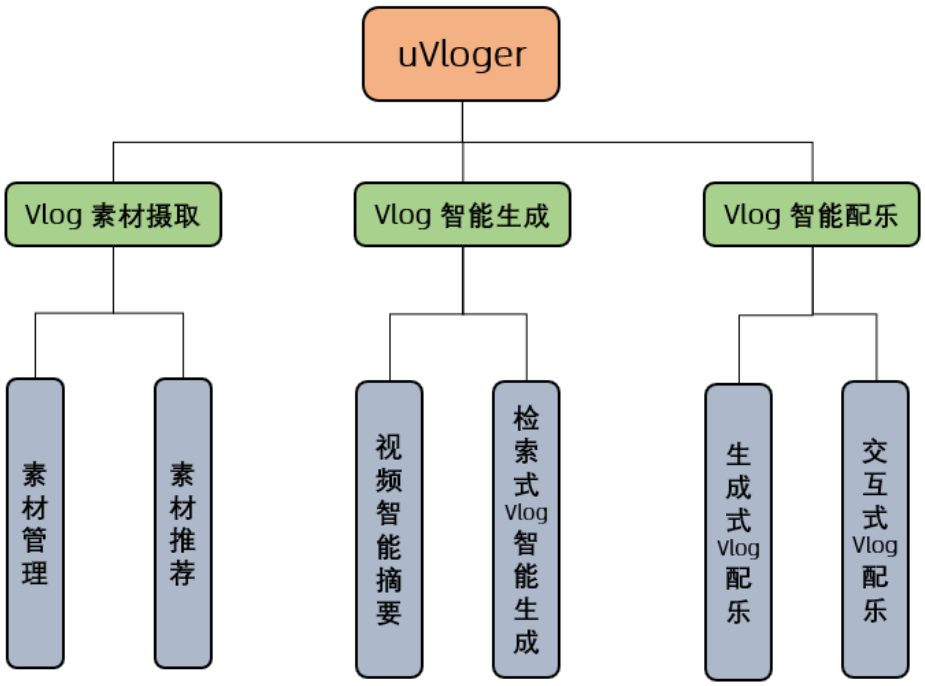
uVloger
Vlog 素材摄取
Vlog 智能生成
Vlog 智能配乐
素材管理
素材推荐
视频智能摘要
检索式Vlog智能生成
生成式Vlog配乐
交互式Vlog配乐

图 **2:** 功能模块图

4.1.2 程序 (模块) 层次结构关系

功能模块	功能	功能描述	优先级
用户登录	用户登录	对用户输入的用户名，密码进行验证，验证通过后，该用户可以使用 vlog 系统中自己拥有权限的那部分功能，否则拒绝使用。	10
维护基本资料	普通用户资料维护	用户的修改删除新增或者查询，系统根据用户的操作，对用户资料进行更新或显示。	10
	视频片段资料维护	用户上传、修改、删除、查询视频资料，对视频片段增加修改删除 tag	10
Vlog 素材摄取	素材管理	将用户本地的多模态碎片信息，通过低级特征（时序、地理位置、传感器信息），以及高级特征（基于深度学习的多模块信息理解）归类整理	10
	素材推荐	根据用户画像、已有素材特征，从分享素材当中预测可以用于改良用户 Vlog 的优质、专业素材，并通过快用卡片推送给用户	8
vlog 生成	视频智能摘要	利用深度学习的“智能摘要”技术，提取出冗长视频素材当中的关键部分	9
	素材剪辑生成	将视频的摘要片段，进行构图优化，加入转场，剪辑成完整的 Vlog 作品	8
vlog 智能配乐	生成式 vlog 配乐	根据用户输入的视频，智能生成契合该视频的背景音乐	10
	交互式 vlog 配乐	通过交互形式，获取用户对某些多义性视频片段的理解，并将该语义作为音乐生成的条件，以影响局部乐句的生成	8

4.2 全局数据结构说明

4.2.1 常量

前端服务器配置:

名称	类型	值	定义
.NODE_ENV	string	“production”or “development”	当前运行环境，开发环境或生产环境

FORWARDING_HOST	String	production FORWARDING_HOST development FORWARDING_HOST	Nginx 转发机地址（开发与生产模式不同地址）
STATIC_HOST	String	production FORWARDING_HOST development FORWARDING_HOSTs	静态资源服务器地址（开发与生产模式不同地址）
STATIC_FOLDERS	JSON	AVATAR_FOLDER, MUSIC_FOLDER, MODEL_FOLDER …	不同类静态资源地址（图片，音乐，模型文件等）
Config: webpack Config: wepack_overrides	JSON		脚手架内封装的 Webpack 打包配置，以及外部的 wepack 的配置覆写
Config: Sagas	JSON	{…sagas}	使用 Redux-Saga 管理异步流，统一配置异步流入口
RECOMMEND_TYPES	JSON	ARTICLE, MUSIC, USER	支持推荐的资源类型，如音乐，用户，以及歌单
USER_TYPES	JSON	ADMIN, USER	用户权限类型，管理员，普通用户，游客
FETCH_TYPES	JSON	FETCH_END FETCH_START	异步请求的处理状态，包括正在请求，请求结束
MSG_CODE	Array		与后端接口约定的状态码类型，采用传统的 HTTP 状态码
Routes	Array[JSON]	[{Path, Component, Auth}, …]	路由表，每个字段包含路由的地址信息，渲染的组件，以及是否需要权限保护

Node Server 常量：

名称	类型	值	定义
Config MONGODB	JSON	{ uri, username, password }	MongoDB 接口访问的配置，包括 MongDB 服务器地址，登陆验证信息
Config REDIS	JSON	{RDS_PORT, RDS_HOST, RDS_OPTS }	Redis 接口访问配置
Config SESSION_STORAGE	JSON	{port, host, db}	用于存储 Session 的 Redis 数据库配置
Config TEST_HOST	JSON	production TEST_HOST development TEST_HOST	系统测试机地址，用于跨域访问的配置，分为开发与生产模式
Config STATIC_HOST	String	production STATIC_HOST development TEST_HOST	生产与开发模式下的静态资源服务器地址
Config HTTP_CODE	JSON	status code of http response	响应状态码
Config HTTP_MSG	JSON	SUCCESS: {GET: ‘获取成功，…} NOT_FOUND: ‘资源不存在’ …	各类状态码对应的响应信息
Config UPLOAD_MODE	JSON	‘singe’‘array’‘any’	文件上传模式
Config FORWARDING_HOST	String	production FORWARDING_HOST development FORWARDING_HOST	Nginx 转发机地址，用于跨域访问的配置，分为生产模式和开发模式
Config RECOMMENDER	JSON	{ClassName, NearestNeightbors}	推荐系统设置，推荐的类型名（推荐用户还是单曲/歌单）最近邻居数量

Routes	Array[JSON]	[{Path, Method Auth}, …]	路由表，元素信息包括路由地址，请求方式以及是否需要权限保护
SchemaMeta	JSON	{字段 1: {Type, Default, …}}	MongoDB 表定义的元数据，其中每个字段包含类型，默认值等信息
AllowedAudioTypes	Array	[‘mp3’, ‘flac’, ‘wav’, ‘mid’]	允许的音频格式类型

4.2.2 数据结构

4.3 CSCI 部件

本条应：

a. 标识构成该 CSCI 的所有软件配置项。应赋予每个软件配置项一个项目唯一标识符。

注: 软件配置项是 CSCI 设计中的一个元素，如 CSCI 的一个主要的分支、该分支的一个组成部分、一个类、对象、模块、函数、例程或数据库. 软件配置项可以出现在一个层次结构的不同层次上，并且可以由其他软件配置项组成. 设计中的软件配置项与实现它们的代码和数据实体 (例程、过程、数据库、数据文件等) 或包含这些实体的计算机文件之间，可以有也可以没有一对一的关系。一个数据库可以被处理为一个 CSCI，也可被处理为一个软件配置项。SDD 可以通过与所采用的设计方法学一致的名字来引用软件配置项。

b. 给出软件配置项的静态关系 (如“组成”)。根据所选择的软件设计方法学可以给出多种关系 (例如，采用面向对象的设计方法时，本条既可以给出类和对象结构，也可以给出 CSCI 的模块和过程结构)。

c. 陈述每个软件配置项的用途，并标识分配给它的 CSCI 需求与 CSCI 级设计决策 (需求的分配也可在 6.a 中提供)。

d. 标识每个软件配置项的开发状态/类型 (如新开发的软件配置项、重用已有设计或软件的软件配置项、再工程的已有设计或软件、为重用而开发的软件等)。对于已有设计或软件，本说明应提供标识信息，如名称、版本、文档引用、库等。

e. 描述 CSCI(若适用，每个软件配置项) 计划使用的计算机硬件资源 (例如处理器能力、内存容量、输入/输出设备能力、辅存容量和通信/网络设备能力)。这些描述应覆盖该 CSCI 的资源使用需求中提及的、影响该 cscl 的系统级资源分配中提及的、以及在软件开发计划的资源使用度量计划中提及的所有计算机硬件资源。如果一给定的计算机硬件资源的所有使用数据出现在同一个地方，如在一个 SDD 中，则本条可以引用它。针对每一计算机硬件资源应包括如下信息:

1) 得到满足的 CSCI 需求或系统级资源分配;

2) 使用数据所基于的假设和条件 (例如，典型用法、最坏情况用法、特定事件的假设);

3) 影响使用的特殊考虑 (例如虚存的使用、覆盖的使用、多处理器的使用或操作系统开销、库软件或其他的实现开销的影响);

4) 所使用的度量单位 (例如处理器能力百分比、每秒周期、内存字节数、每秒千字节);

5) 进行评估或度量的级别 (例如软件配置项,CSCI 或可执行程序)。

f. 指出实现每个软件配置项的软件放置在哪个程序库中。

4.4 执行概念

4.5 接口设计

4.5.1 视频智能摘要

用例名称	视频智能摘要
功能简述	利用深度学习的“智能摘要”技术，提取出冗长视频素材当中的关键部分
用例编号	W003
执行者	Tornado Gateway
前置条件	素材完整：用户 Vlog 草稿涉及素材已上传至服务器
后置条件	内存存储视频的摘要信息
涉众利益	用户：希望得到正确的视频摘要信息
基本路径	用户选中 Vlog 草稿涉及素材 用户点击请求 Vlog 生成服务 后台分析素材，提取出冗长视频素材中的关键时间区间
扩展路径	后台素材缺失，提示用户，启用素材上传 本地素材缺失，提示用户
字段列表	视频二进制数据、素材唯一标志列表、视频摘要区间信息
设计规则	单独模块
未解决的问题	快应用对多视频拼接播放支持

4.5.2 检索式 **Vlog** 生成

用例名称	检索式 Vlog 智能生成
功能简述	将视频摘要优化剪辑成完整的 Vlog 作品
用例编号	W004
执行者	Tornado Gateway、Video Server
前置条件	摘要序列完整：用户 Vlog 草稿智能摘要已生成并调整完毕
后置条件	内存存储无音乐的 Vlog 视频文件
涉众利益	用户：希望得到满意的 Vlog 视频文件
基本路径	用户调整好摘要序列 用户请求 Vlog 生成，传输摘要序列到后台 后台将业务请求转发给视频处理服务器的消息队列 视频处理服务器将视频摘要剪辑返回链接
扩展路径	后台视频素材缺失，提示用户，启用视频素材上传
字段列表	视频二进制数据、素材唯一标志列表、视频摘要区间信息
设计规则	单独模块

4.5.3 生成式 **Vlog** 配乐

用例名称	生成式 Vlog 配乐
功能简述	根据用户输入的视频，智能生成契合该视频的背景音乐
用例编号	W005
执行者	Tornado Gateway、Video Server
前置条件	视频素材完整：用户 Vlog 视频在服务器数据库存在
后置条件	内存储存配乐后的 Vlog 视频文件
涉众利益	用户：希望得到配乐后的 Vlog 视频
基本路径	用户请求为 Vlog 配乐 后台将业务请求转发给视频处理服务器的消息队列 根据用户输入的视频生成契合该视频的背景音乐
扩展路径	后台视频素材缺失，提示用户，启用视频素材上传
字段列表	视频唯一标志、视频二进制数据
设计规则	单独模块

4.5.4 素材管理

用例名称	素材管理
功能简述	将用户本地的多模态碎片信息，通过低级特征（时序、地理位置、传感器信息），以及高级特征（基于深度学习的多模块信息理解）归类整理
用例编号	W001
执行者	Tornado Gateway、快应用前端
前置条件	文件完整：文件没有损坏 文件格式：符合常用多媒体素材的格式 特征完整：存有相对应的低级特征信息
后置条件	内存存储各种特征信息
涉众利益	用户：希望得到正确的素材管理结果 后台：得到用户可分享的优质素材
基本路径	用户关联系统存储的多模态素材集合 用户同意产品联网分析相关素材 前端本地运算得到素材低级特征，传送给后台 后台对低级特征进行信息理解，返回高级特征 前端利用高级特征渲染快应用卡片
扩展路径	文件损坏，提示用户 文件格式不符合规范，提示用户 网络异常，进行重试，提示用户 网络缺失，提示用户
字段列表	文件合法，素材特征信息
设计规则	单独模块

4.5.5 推荐系统

用例名称	素材管理
功能简述	将用户本地的多模态碎片信息，通过低级特征（时序、地理位置、传感器信息），以及高级特征（基于深度学习的多模块信息理解）归类整理
用例编号	W001
执行者	Tornado Gateway、快应用前端
前置条件	文件完整：文件没有损坏 文件格式：符合常用多媒体素材的格式 特征完整：存有相对应的低级特征信息
后置条件	内存存储各种特征信息
涉众利益	用户：希望得到正确的素材管理结果 后台：得到用户可分享的优质素材
基本路径	用户关联系统存储的多模态素材集合 用户同意产品联网分析相关素材 前端本地运算得到素材低级特征，传送给后台 后台对低级特征进行信息理解，返回高级特征 前端利用高级特征渲染快应用卡片
扩展路径	文件损坏，提示用户 文件格式不符合规范，提示用户 网络异常，进行重试，提示用户 网络缺失，提示用户
字段列表	文件合法，素材特征信息
设计规则	单独模块

4.5.6 交互式 **Vlog** 配乐

用例名称	交互式 Vlog 配乐
功能简述	通过交互形式，获取用户对某些多义性视频片段的理解，并将该语义作为音乐生成的条件，以影响局部乐句的生成
用例编号	W006
执行者	Tornado Gateway、Video Server
前置条件	视频素材完整：用户 Vlog 视频在服务器数据库存在
后置条件	内存储存配乐后的 Vlog 视频文件
涉众利益	用户：希望得到交互配乐后的 Vlog 视频
基本路径	用户请求为 Vlog 配乐 后台将业务请求转发给视频处理服务器的消息队列 利用交互影响局部语句的生成 生成契合该视频的背景音乐，将音乐与视频合成
扩展路径	后台视频素材缺失，提示用户，启用视频素材上传
字段列表	视频唯一标志、视频二进制数据、视频摘要区间信息
设计规则	单独模块

4.5.7 用户登录

- 用例名称：

  中文名称：用户登录

  功能：验证用户的身份

- 简要说明：本用例的功能主要是用于确保用户在提供正确的验证信息之后，可以进一步使用本系统。

- 事件流

  * 基本流

    · 用户请求使用本系统。

    · 系统显示用户登录信息输入界面。

    · 用户输入登录名，密码并确认操作。

    · 系统验证用户登录信息，如果登录信息验证没有通过，系统显示提醒信息，并转向基本流 2，如果验证通过，系统显示系统操作主界面。

  * 备选流

    · 备选流 1

· 客户可以在没有登录成功之前的任意时候要求放弃登录。
· 系统结束用户登录信息输入界面的显示。
· 退出系统。

- 特殊需求：无
- 前置条件：请求使用本系统
- 后置条件：用户登录成功，可以使用系统提供的功能
- 附加说明：无

4.5.8 维护基本资料

- 用例名称：

  中文名称：维护普通用户资料

  功能：用于维护普通用户的资料信息

- 简要说明：本用例的功能主要是用户的修改删除新增或者查询，系统根据用户的操作，对用户资料进行更新或显示。

- 事件流

  * 基本流
    · 用户请求维护用户资料。
    · 系统显示用户资料。
    · 根据用户的操作执行以下相应操作。
    · 用户修改已经存在的用户资料信息，系统执行修改用户资料信息子流。
    · 用户选择增加用户资料操作，系统执行增加用户资料信息子流。
    · 用户选择删除用户资料操作，系统执行删除用户资料信息子流。
    · 用户选择查询符合指定条件的用户资料的信息，系统执行查询用户资料子流。
    · 用户要求保存操作结果。
    · 系统保存用户操作结果。
    · 用户要求结束用户资料信息的维护。
    · 系统结束用户资料的显示。

* 备选流
    - 备选流 1：如果在用户请求保存操作结果的时候，由于网络、数据库管理系统等外部原因造成操作结果不能保存，系统保证以恰当的方式通知用户，并维护用户的操作状态，在外部原因消除之后，用户仍能继续操作。
    - 备选流 2：如果用户要求结束用户信息维护的时候，仍有未保存的信息，系统提醒用户。

– 特殊需求：无

– 前置条件：用户登入系统

– 后置条件：系统保存修改过的用户资料

– 附加说明：无

4.6 接口标识与接口图

本条应陈述赋予每个接口的项目唯一标识符，(若适用) 并用名字、编号、版本和文档引用等标识接口实体 (软件配置项、系统、配置项、用户等)。接口标识应说明哪些实体具有固定接口特性 (从而把接口需求强加给接口实体)，哪些实体正在开发或修改 (因而已把接口需求分配给它们)。(若适用) 应该提供一个或多个接口图以描述这些接口。

4.5.x(接口的项目唯一标识符)

本条 (从 4.5.2 开始编号) 应用项目唯一标识符标识接口，应简要标识接口实体，并且应根据需要划分为几条描述接口实体的单方或双方的接口特性。如果一给定的接口实体本文没有提到 (例如，一个外部系统)，但是其接口特性需要在本 SDD 描述的接口实体时提到，则这些特性应以假设、或“当 [未提到实体] 这样做时，[提到的实体] 将……”的形式描述。本条可引用其他文档 (例如数据字典、协议标准、用户接口标准) 代替本条的描述信息。本设计说明应包括以下内容，(若适用) 它们可按适合于要提供的信息的任何次序给出，并且应从接口实体角度指出这些特性之间的区别 (例如数据元素的大小、频率或其他特性的不同期望)。

a. 由接口实体分配给接口的优先级；

b. 要实现的接口的类型 (例如实时数据传输、数据的存储与检索等)；

c. 接口实体将提供、存储、发送、访问、接收的单个数据元素的特性，例如：

1) 名称/标识符；

a) 项目唯一标识符；

b) 非技术 (自然语言) 名称；

c) 标准数据元素名称；

d) 缩写名或同义名；

2) 数据类型 (字母数字、整数等)；

3) 大小与格式 (例如字符串的长度与标点符号)；

4) 计量单位 (如米、元、纳秒等)；

5) 范围或可能值的枚举 (如 0−99)

6) 准确度 (正确程度) 与精度 (有效数位数)；

7) 优先级、时序、频率、容量、序列和其他约束，如数据元素是否可被更新，业务规则是否适用；

8) 保密性与私密性约束；

9) 来源 (设置/发送实体) 与接收者 (使用/接收实体)。

d. 接口实体将提供、存储、发送、访问、接收的数据元素集合体 (记录、消息、文件、数组、显示、报表等) 的特性，例如：

1) 名称/标识符；

a) 项目唯一标识符；

b) 非技术 (自然语言) 名称；

c) 技术名称 (如代码或数据库中的记录或数据结构名)；

d) 缩写名或同义名；

2) 数据元素集合体中的数据元素及其结构 (编号、次序、分组)；

3) 媒体 (如盘) 及媒体上数据元素/集合体的结构；

4) 显示和其他输出的视听特性 (如颜色、布局、字体、图标及其他显示元素、蜂鸣声、亮度等)；

5 软件架构风格原理

5.1 定义

软件体系结构风格（Architectural Styles）是描述特定系统组织方式的惯用范例，强调了软件系统中通用的组织结构。

一个软件体系结构风格定义了构件和连接件类型的符号集，及规定了它们怎样组合起来的约束集合。

架构风格是一组原则。你可以把它看成是一组为系统家族提供抽象框架的粗粒度模式。架构风格能改进分块，还能为频繁出现的问题提供解决方案，以此促进设计重用。

5.2 涉及问题

- 设计词汇表是什么？或者构件和连接器的类型是什么?
- 可容许的结构模式是什么?
- 基本的计算模型是什么?
- 风格的基本不变性是什么?
- 其使用的常见例子是什么?
- 使用此风格的优缺点是什么?
- 其常见特例是什么?

5.3 意义

软件体系结构设计的一个中心问题是能否重用软件体系结构模式，或者采用某种软件体系结构风格。有原则地使用软件体系结构风格具有如下意义:

- 它促进了设计的复用，使得一些经过实践证实的解决方案能够可靠地解决新问题。
- 它能够带来显著的代码复用，使得体系结构风格中的不变部分可共享同一个解决方案。

- 便于设计者之间的交流与理解。
- 通过对标准风格的使用支持了互操作性，以便于相关工具的集成。
- 在限定了设计空间的情况下，能够对相关风格作出分析。
- 能够对特定的风格提供可视化支持。

5.4 待明确问题

与此同时，人们目前尚不能准确回答的问题是：

- 系统设计的哪个要点可以用风格来描述；
- 能否用系统的特性来比较不同的风格，如何确定用不同的风格设计系统之间的互操作；
- 能否开发出通用的工具来扩展风格；
- 如何为一个给定的问题选择恰当的体系结构风格，或者如何通过组合现有的若干风格来产生一个新的风格。

6 架构风格实例

6.1 系统层次划分

M.Shaw 等人根据此框架给出了管道与过滤器、数据抽象和面向对象组织、基于事件的隐式调用、分层系统、仓库系统及知识库和表格驱动的解释器等一些常见的软件体系结构风格。

- 客户端-服务器：将系统分为两个应用，其中客户端向服务器发送服务请求。
- 基于组件的架构：把应用设计分解为可重用的功能、逻辑组件，这些组件的位置相互透明，只暴露明确定义的通信接口。
- 分层架构：把应用的关注点分割为堆栈组（层）。

- 消息总线：指接收、发送消息的软件系统，消息基于一组已知格式，以便系统无需知道实际接收者就能互相通信。
- N 层/三层架构：用与分层风格差不多一样的方式将功能划分为独立的部分，每个部分是一个层，处于完全独立的计算机上。
- 面向对象：该架构风格是将应用或系统任务分割成单独、可重用、可自给的对象，每个对象包含数据，以及与对象相关的行为。
- 分离表现层：将处理用户界面的逻辑从用户界面（UI）视图和用户操作的数据中分离出来。
- 面向服务架构（SOA）：是指那些利用契约和消息将功能暴露为服务、消费功能服务的应用。

6.2 按领域划分

- 通信：SOA，消息总线，管道和过滤器
- 部署：客户端/服务器，三层架构，N 层架构
- 领域：领域模型，网关
- 交互：分离表现层
- 结构：基于组件的架构，面向对象，分层架构

7 典型风格

7.1 管道/过滤器风格

7.1.1 定义

在管道/过滤器风格的软件体系结构中，每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。因此，这里的构件被称为过滤器，这种风格的连接件就像是数据流传输的管道，将一个过滤器的输出传到另一过

滤器的输入。此风格特别重要的过滤器必须是独立的实体，它不能与其它的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。一个管道/过滤器网络输出的正确性并不依赖于过滤器进行增量计算过程的顺序。

7.1.2 优势

- 使得软构件具有良好的隐蔽性和高内聚、低耦合的特点；
- 允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；
- 支持软件重用。重要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来；
- 系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉；
- 允许对一些如吞吐量、死锁等属性的分析；
- 支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。

7.1.3 劣势

- 通常导致进程成为批处理的结构。这是因为虽然过滤器可增量式地处理数据，但它们是独立的，所以设计者必须将每个过滤器看成一个完整的从输入到输出的转换。
- 不适合处理交互的应用。当需要增量地显示改变时，这个问题尤为严重。
- 因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性

7.2 数据抽象与面向对象风格

7.2.1 定义

抽象数据类型概念对软件系统有着重要作用，目前软件界已普遍转向使用面向对象系统。这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的构件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的构件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。

7.2.2 优势

面向对象的系统有许多的优点，并早已为人所知：

- 因为对象对其它对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其它的对象。
- 设计者可将一些数据存取操作的问题分解成一些交互的代理程序的集合。

7.2.3 劣势

- 为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象。
- 必须修改所有显式调用它的其它对象，并消除由此带来的一些副作用。例如，如果 A 使用了对象 B，C 也使用了对象 B，那么，C 对 B 的使用所造成的对 A 的影响可能是料想不到的。

7.3 基于事件的隐式调用风格

7.3.1 定义

基于事件的隐式调用风格的思想是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

从体系结构上说，这种风格的构件是一些模块，这些模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。

基于事件的隐式调用风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

支持基于事件的隐式调用的应用系统很多。例如，在编程环境中用于集成各种工具，在数据库管理系统中确保数据的一致性约束，在用户界面系统中管理数据，以及在编辑器中支持语法检查。例如在某系统中，编辑器和变量监视器可以登记相应 Debugger 的断点事件。当 Debugger 在断点处停下时，它声明该事件，由系统自动调用处理程序，如编辑程序可以卷屏到断点，变量监视器刷新变量数值。而 Debugger 本身只声明事件，并不关心哪些过程会启动，也不关心这些过程做什么处理。

7.3.2 优势

- 为软件重用提供了强大的支持。当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中。
- 为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其它构件的接口。

7.3.3 劣势

- 构件放弃了对系统计算的控制。一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的构成，它也不能保证这些过程被调用的顺序。
- 数据交换的问题。有时数据可被一个事件传递，但另一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。在这些情况下，全局性能和资源管理便成了问题。
- 既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理存在问题。

7.4 C2 风格

7.4.1 规则

C2 体系结构风格可以概括为：通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2 风格中的系统组织规则如下：

- 系统中的构件和连接件都有一个顶部和一个底部；
- 构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；
- 一个连接件可以和任意数目的其它构件和连接件连接；
- 当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

7.4.2 特点

- 系统中的构件可实现应用需求，并能将任意复杂度的功能封装在一起；
- 所有构件之间的通讯是通过以连接件为中介的异步消息交换机制来实现的；
- 构件相对独立，构件之间依赖性较少。系统中不存在某些构件将在同一地址空间内执行，或某些构件共享特定控制线程之类的相关性假设。

7.5 层次系统风格

7.5.1 定义

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机 (在另一些层次系统中层是部分不透明的)。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

这种风格支持基于可增加抽象层的设计。这样，允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

7.5.2 优势

- 支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解；
- 支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层；
- 支持重用。只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，而允许各种不同的实现方法。

7.5.3 劣势

- 并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来；
- 很难找到一个合适的、正确的层次抽象方法。

7.6 仓库风格

7.6.1 定义

在仓库风格中，有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存贮上执行，仓库与外构件间的相互作用在系统中会有大的变化。

控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统。

7.6.2 组成

- 知识源。知识源中包含独立的、与应用程序相关的知识，知识源之间不直接进行通讯，它们之间的交互只通过黑板来完成。
- 黑板数据结构。黑板数据是按照与应用程序相关的层次来组织的解决问题的数据，知识源通过不断地改变黑板数据来解决问题。

- 控制。控制完全由黑板的状态驱动，黑板状态的改变决定使用的特定知识。

8 架构模式

8.1 分层模式

这种模式可以用来构建能分解为子任务组的结构化程序，每个子任务都处于特定的抽象级别。每层为下一更高层提供服务。一个通用信息系统常见的 4 层如下：呈现层（UI 用户界面层）应用层（service-服务层）业务逻辑层（domain-领域层）数据访问层（persistence-持久化层）用途常用的桌面应用电子商务 web 应用

8.2 CS 模式

这个模式包含两个部分：一个服务端 + 多个客户端。服务端组件提供给多个客户端组件服务。客户端请求服务，服务端提供相应的服务给客户端。除此之外，服务端不间歇地监听来自客户端的服务请求。用途在线应用程序，如电子邮件，文件共享和银行业务

8.3 主从模式

这种模式由两部分组成：主人（master）和奴隶（slaves）。主组件将工作分配给特定的从组件，并根据从组件返回的结果计算最终结果。用途在数据库复制中，主数据库被视为权威来源，并且从属数据库与其同步在计算机系统中连接到总线的外设（主从驱动器）

8.4 管道过滤器模式

这个模式可被用于构建生成和处理数据流的系统。每个处理步骤都包含在一个过滤器组件中。被处理的数据需通过管道进行传递。这些管道可用于缓冲或同步目的。用途编译器。使用连续的过滤器执行词法分析，解析，语义分析和代码生成。生物信息学工作流程

8.5 经纪人模式

这个模式用于使用架构的组件来构建的分布式系统。这些组件可以通过远程服务调用相互交互。代理组件负责协调组件之间的通信。服务端将其能力（服务和特性）发布给代理。客户端向经纪人请求服务，然后经纪人将客户重定向到其注册的对应服务。用途消息代理软件，如 Apache ActiveMQ，Apache Kafka，RabbitMQ，JBoss Messaging

8.6 点对点模式

在这种模式中，单个组件被称为同级点（Peer：身份，级别相同的点）。同级点可以既作为客户端，向其它同级点请求服务，又作为服务器向其它同级点提供服务。一个同级点既可以充当客户端或服务器或两者兼而有之，并且可以随着时间动态地改变其角色。用途文件分享网络，例如 Gnutella,Gnutella2 多媒体协议，例如 P2PTV、PDTP

8.7 事件总线模式

这个模式主要用于处理事件，有 4 个主要的组件：事件源，事件监听器，频道，事件总线。事件源将消息发布到事件总线上的特定频道。监听器订阅特定频道。监听器会收到发布到他们之前订阅的频道的消息。用途 android 开发通知系统

8.8 模型-视图-控制器模式

这个模式又叫 MVC 模式，他把交互式应用程序分成了 3 个部分。模型，包含核心功能和数据视图，把信息呈现给用户（可能有多个视图）控制器，处理用户输入用途使用主流的编程语言架构的 web 应用程序 web 框架如 Django、Ruby on Rails

8.9 黑板模式

这种模式对于没有确定性解决策略的问题是有帮助的。黑板模式由 3 个主要组件组成。黑板-一个包含来自解决方案空间对象的结构化全局内存知识源-具有自我表达的专用模块控制组件-选择，配置，执行模块所有组件都可以访问黑板。组件可能产生添加到黑板的新数据对象。组件在黑板上查找特定类型的数据，并可能通过与现有知识源的模式匹配找到这些数据。用途语音识别车辆识别与跟踪蛋白质结构鉴定声纳信号解释

8.10 解释模式

此模式用于设计解释用专用语言编写的程序的组件。它主要指定如何解释执行程序代码，称为用特定语言编写的句子或表达式。基本思想是为语言的每个符号设置一个类。用途数据库查询语言例如 SQL 用于描述通信协议的语言