代码管理平台Code Cube

软件架构文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <13/May/2024> | <1.0> | 最初的发布，增加简介、用例视图、逻辑视图、进程视图、部署视图 、实现视图、技术视图、数据视图、核心算法设计、质量属性的设计 | 王晨丞 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 5

3.1 概述 5

3.2 在构架方面具有重要意义的设计包 6

3.2.1 Spring Boot包说明 6

3.2.2 Hugging Face Transformers包说明 6

4. 进程视图 7

5. 部署视图 7

6. 实现视图 7

7. 技术视图 7

8. 数据视图 8

9. 质量属性的设计 8

软件架构文档

# 简介

## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

本文档分为用例视图、逻辑视图、进程视图、部署视图、实现视图、技术视图、数据视图、核心算法设计、质量属性的设计九个方面，对系统的构架进行说明。本文档面向开发人员、架构师、项目经理、客户和合作伙伴。

**1.1.1开发人员**：

开发人员应该详细阅读软件架构文档，以理解系统的整体结构和各个组件之间的关系。他们应该特别关注接口定义、数据流图、通信协议和关键组件的功能描述。通过遵循架构文档中定义的标准和约定，开发人员可以确保他们的工作与整个系统的架构保持一致。

**1.1.2架构师**：

架构师应该使用软件架构文档来评估现有系统的架构是否合理，并根据需要进行修改或改进。他们应该分析架构的可扩展性、可维护性、性能和安全性等方面，并与团队成员讨论如何优化架构以满足项目需求。架构师还应该确保架构文档与项目的其他文档（如需求文档、设计文档等）保持一致。

**1.1.3项目经理**：

项目经理应该通过软件架构文档来了解项目的技术复杂度、资源需求和开发周期等信息。有助于项目经理制定合理的项目计划、分配资源和控制进度。关注架构文档中的风险和挑战部分，以便提前采取措施进行风险管理和问题解决。

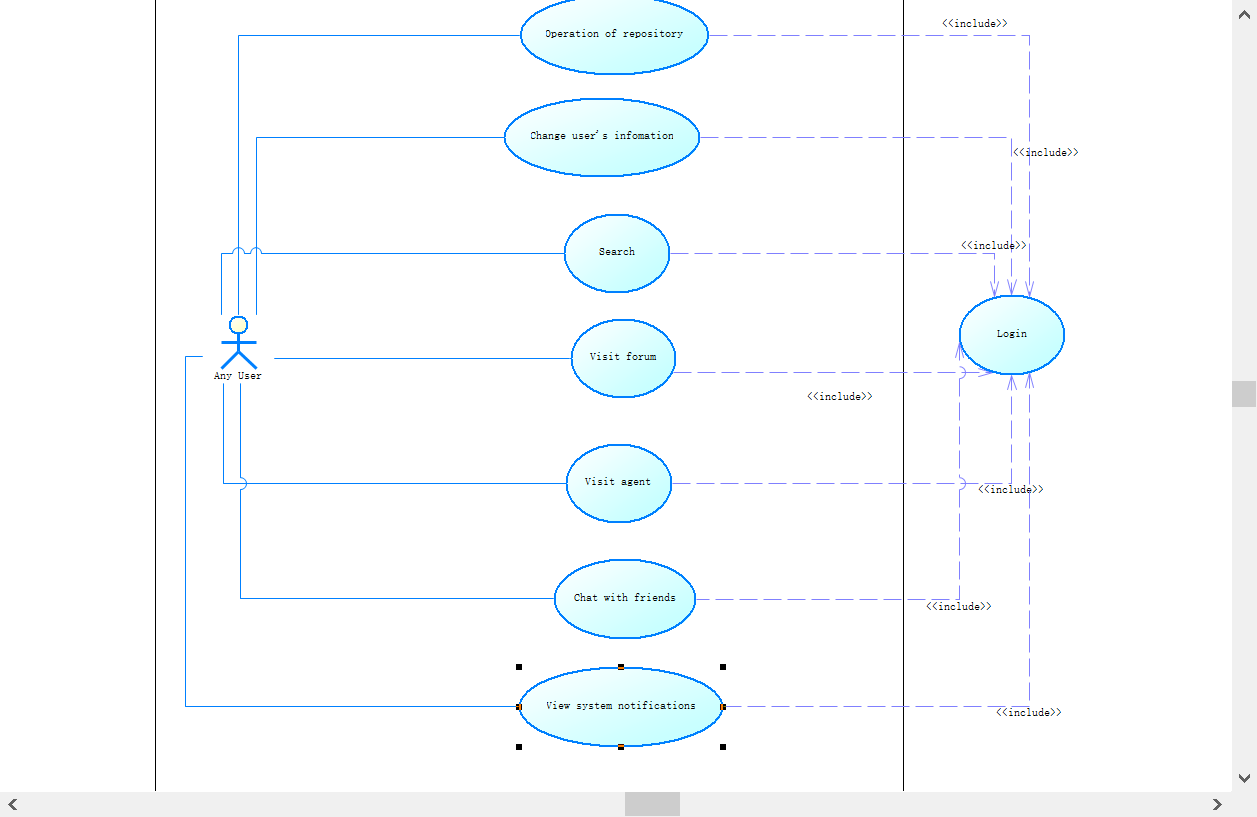
**1.1.4客户和合作伙伴**：

利益相关者可以通过软件架构文档了解系统的整体情况、技术选型和关键设计决策等信息。利用这些信息来评估项目的价值、风险和潜在收益，并与其他利益相关者进行有效的沟通和协作。理解架构文档中的关键概念和术语，以便做出合适的决策。

## 参考资料

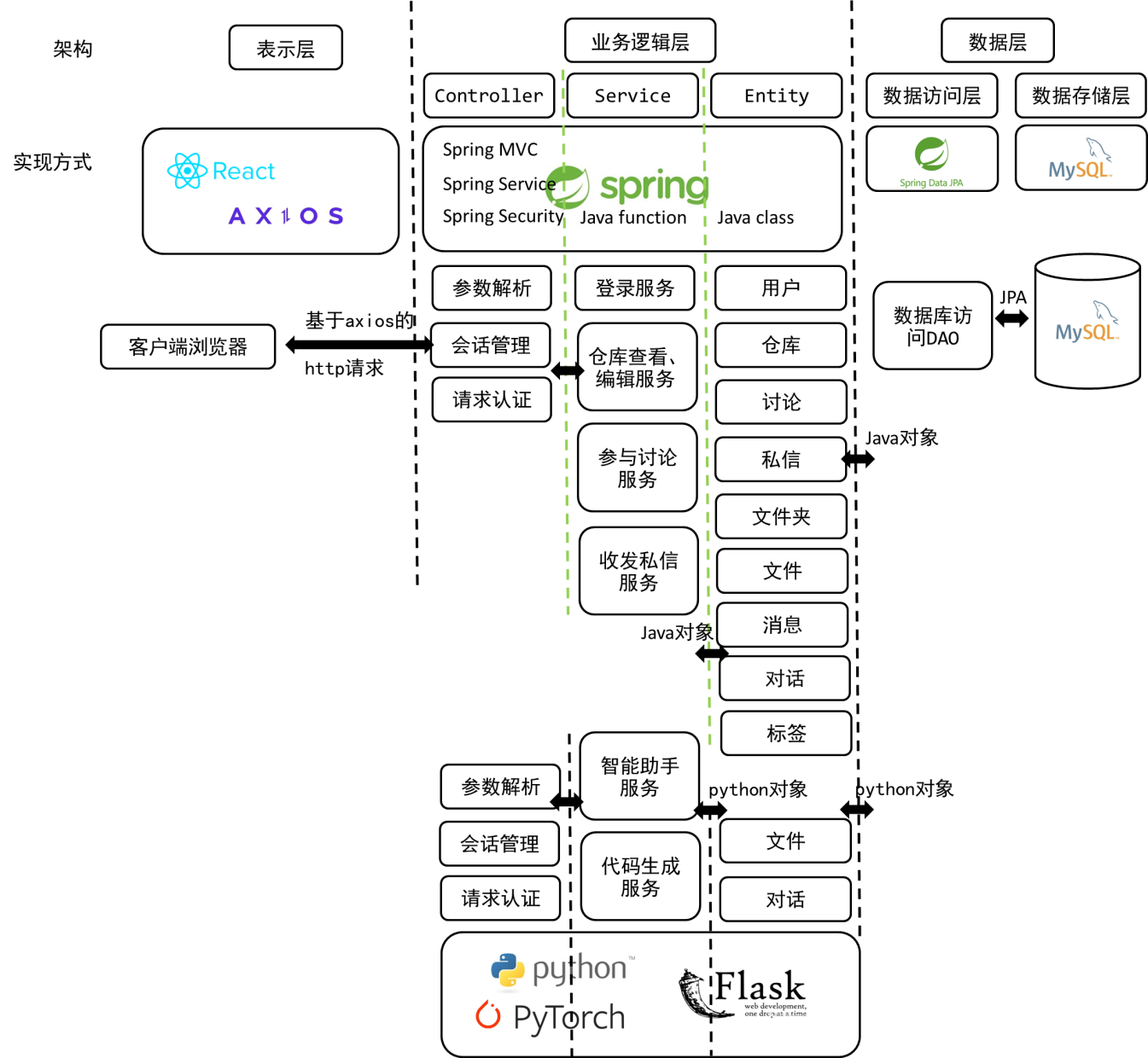
无

# 用例视图



# 逻辑视图

## 概述



对于我们的逻辑架构，我们采用了3Tiers的三层式架构风格，主要分为表示层、业务逻辑层和数据层。

在表示层中，我们基于react框架进行前端的实现，而客户端浏览器通过基于axios的http请求与业务逻辑层进行连接。

在业务逻辑层中，我们基于java的springboot框架和python的pytorch和flask框架进行开发。我们主要分为三个子层：Controller，Service和Entity。其中Controller负责和前端进行直接的信息交互，Service层按照特定逻辑进行业务过程，Entity负责与数据库建立联系。我们的springboot后端负责大部分的业务逻辑，智能助手和代码生成服务则依赖于python的flask后端进行。

在数据层中，我们主要分为数据访问层和数据存储层。数据访问层采用springboot jpa进行数据库的操作。使用关系型数据库mysql进行数据的存储。

## 在构架方面具有重要意义的设计包

## 3.2.1 Spring Boot包说明

**包名称**: org.springframework.boot

**简要说明**:  
Spring Boot是一个开源的Java框架，用于简化Spring应用的初始搭建和开发过程。通过约定大于配置（Convention Over Configuration）的方式，Spring Boot可以帮助开发者快速搭建一个独立的、生产级别的基于Spring的应用。

1. **重要的类和包**

Spring Boot的核心功能分散在多个子包中，以下是一些重要的类和包：

1. **org.springframework.boot.autoconfigure**:
   * 包含大量自动配置类，这些类在类路径上检测到特定库时会自动配置Spring应用。
   * 例如：DataSourceAutoConfiguration（数据源自动配置）、ServletWebServerFactoryAutoConfiguration（Servlet Web服务器自动配置）等。
2. **org.springframework.boot.SpringApplication**:
   * 主要的启动类，负责启动Spring应用。
   * 提供静态方法来方便地启动Spring应用，如SpringApplication.run(MyClass.class, args);。
3. **org.springframework.boot.web.servlet.support.SpringBootServletInitializer**:
   * 用于在Servlet容器（如Tomcat）中部署Spring Boot应用的类。
   * 继承此类的应用可以作为WAR包部署到外部Servlet容器中。
4. **org.springframework.boot.context.properties.ConfigurationProperties**:
   * 注解类，用于将application.properties或application.yml中的属性绑定到Java Bean上。
5. **org.springframework.boot.test**:
   * 包含用于测试Spring Boot应用的类和注解。
   * 例如：SpringBootTest注解用于标记Spring Boot集成测试类。

## 3.2.2 Hugging Face Transformers包说明

**包名称**Huggingface Transformers

**简要说明**

Hugging Face Transformers是一个自然语言处理（NLP）工具包，旨在简化NLP模型的开发和使 用。它提供了大量预训练的模型和算法，支持多种编程语言，如Python、Java和JavaScript，能 够方便地集成到各种应用中。其核心是基于Transformer的模型，如BERT、GPT、XLNet、RoBERTa 等，这些模型在NLP的多种任务中表现卓越，如问答、文本分类、情感分析、机器翻译等。

重要类介绍

**Model Classes**:

PreTrainedModel: 这是所有预训练模型的基础类。它提供了加载预训练权重、保存和加载模型、 将模型转换为特定设备（如CPU或GPU）等功能。

**Tokenizer Classes:**

BertTokenizer, GPTTokenizer, XLNetTokenizer, RoBERTaTokenizer 等：这些类对应于特定模型 的分词器。分词器是NLP中用于将文本转换为模型可以理解的格式（如token IDs和attention masks）的重要工具。这些类提供了分词、编码和解码等功能。

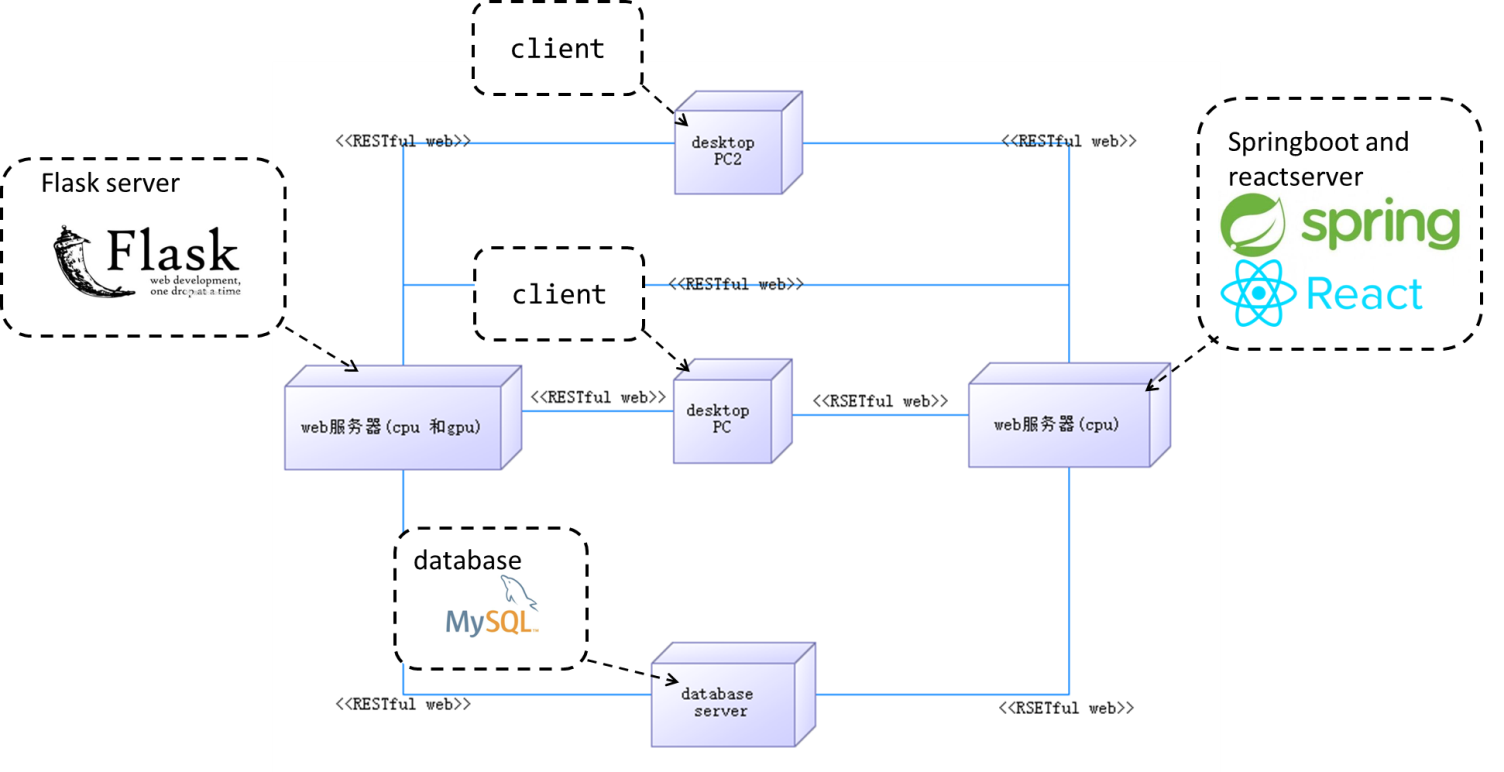
**Configuration Classes:**

BertConfig, GPTConfig, XLNetConfig, RoBERTaConfig 等：这些类对应于特定模型的配置。它们 包含了模型的参数设置，如隐藏层大小、注意力头数等。在加载预训练模型时，这些配置将被用于 初始化模型。

# 进程视图

前端使用react框架fetch api，后端使用springboot、flask框架进行通信。故无须进程视图

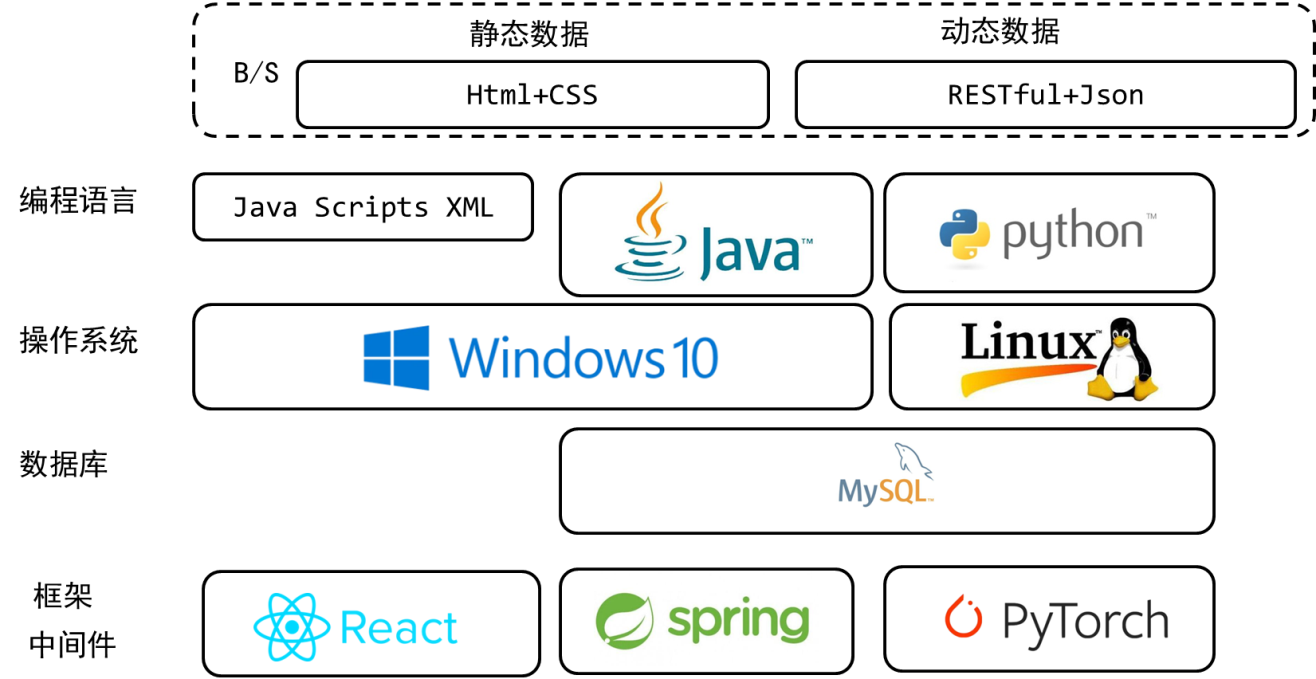
# 部署视图



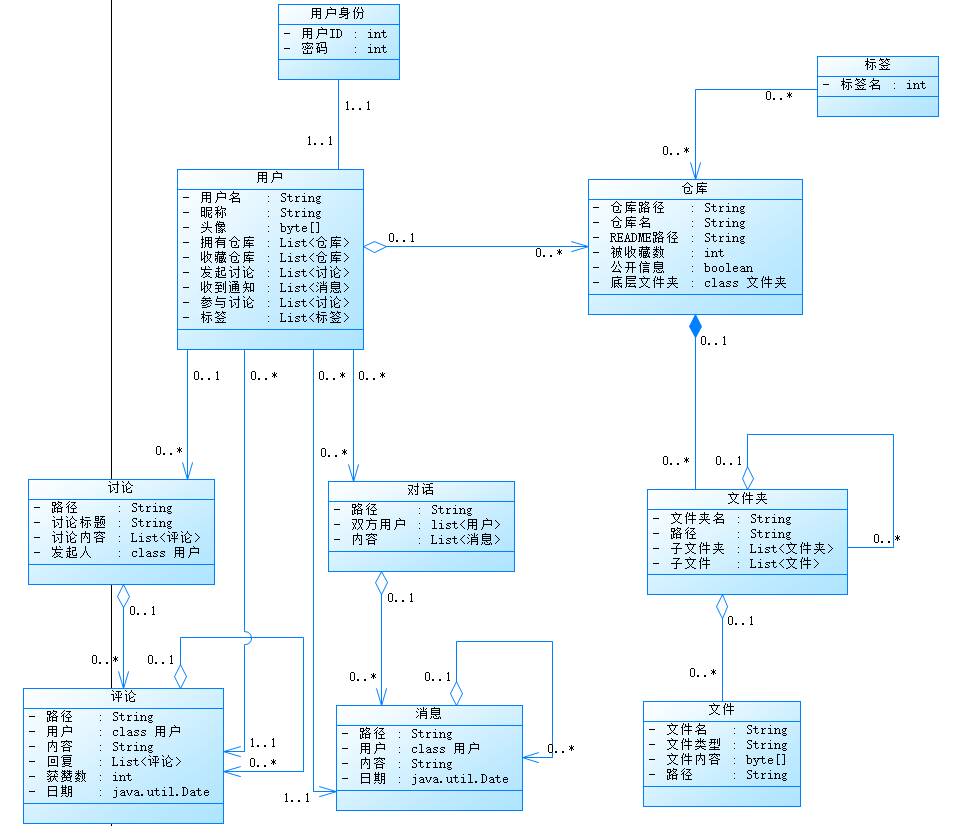
# 实现视图



# 技术视图



# 数据视图



# 质量属性的设计

10.1性能：

软件架构通过优化系统设计和算法选择，减少不必要的计算和存储开销，提高系统的处理速度和响应能力。

缓存策略、并发处理以及负载均衡等技术的合理应用，能够显著提升系统的吞吐量和并发处理能力。

通过合理的系统分层和模块划分，降低系统复杂度，提高整体性能。

10.2可扩展性：

软件架构采用模块化设计，将功能拆分为独立的模块，通过定义明确的接口进行通信，便于新功能的添加和现有功能的扩展。

遵循开放封闭原则，对扩展开放，对修改封闭，降低系统修改成本，提高可扩展性。

10.3可靠性：

软件架构通过冗余和备份策略，确保在出现故障或错误时，系统仍然可用。例如，使用冗余服务器和备份数据库，实现容错和恢复能力。

通过全面的测试和验证，发现和纠正潜在的问题和缺陷，提高软件的稳定性和可靠性。

引入错误处理和异常管理机制，捕获和处理可能出现的异常情况，避免程序崩溃或不可预测的行为。

10.4易用性：

在软件架构设计中，关注用户体验和交互设计，确保软件界面友好、操作简便。

通过明确软件的目标用户，进行需求场景分析和易用性设计约束或要求的梳理，确保软件符合用户的使用习惯和需求。

采用UI架构设计，优化信息呈现和操作组织，提高软件的易用性。

10.5可移植性：

软件架构通过跨平台设计，确保软件可以在不同的操作系统和硬件平台上运行。

考虑编译器、汇编器、链接器的兼容性以及API和运行库的适配性，实现软件的跨平台移植。

采用模块化设计，降低软件对底层实现细节的依赖，提高软件的可移植性。