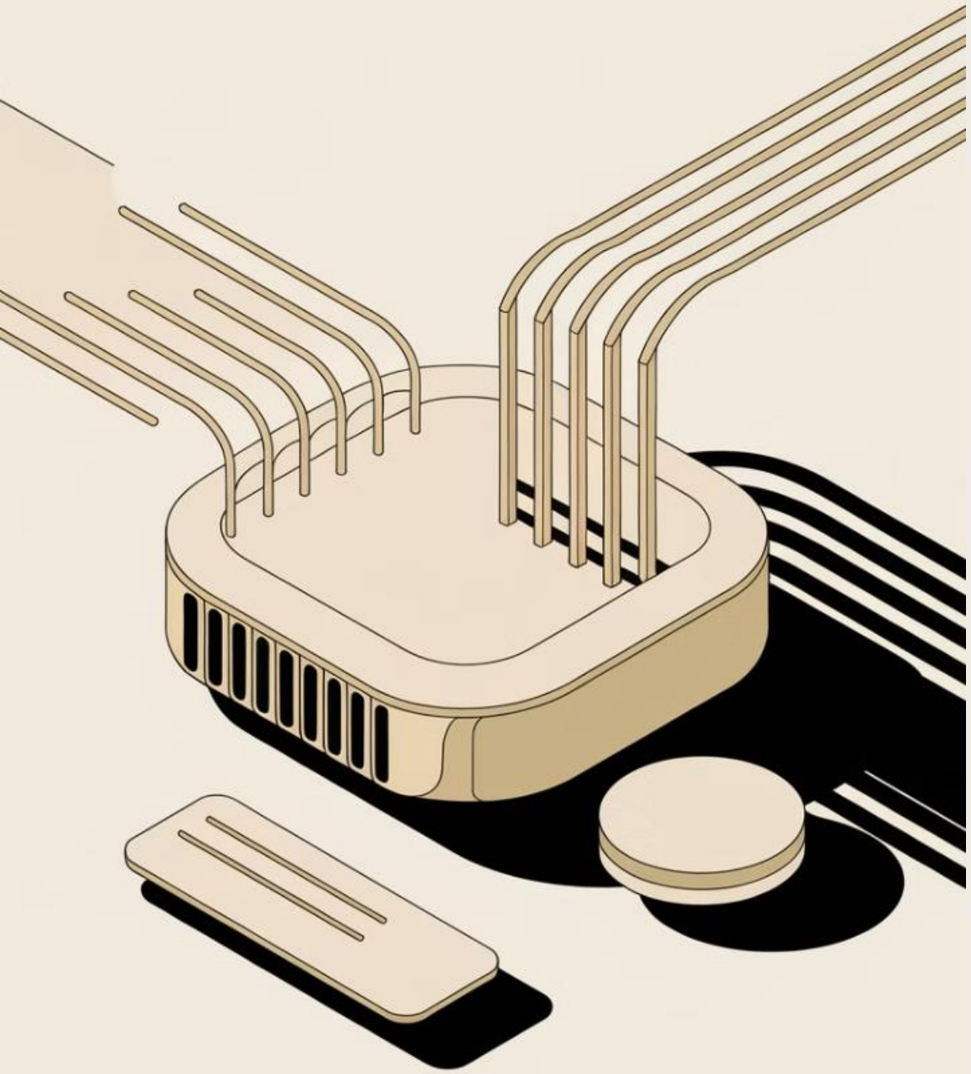
项目总结



C++ WebServer 特性

1 高性能

利用 C++ 新特性提高代码可读性和效率。

2 事件驱动

采用事件驱动模型，提高服务器响应速度。

3 多线程

使用多线程技术，提升服务器并发处理能力。

4 分布式

支持分布式部署，扩展服务器容量。

复习逻辑

1

理解基本服务器原理

首先要理解基本服务器的运行原理，例如网络通信、数据处理、资源管理等。

2

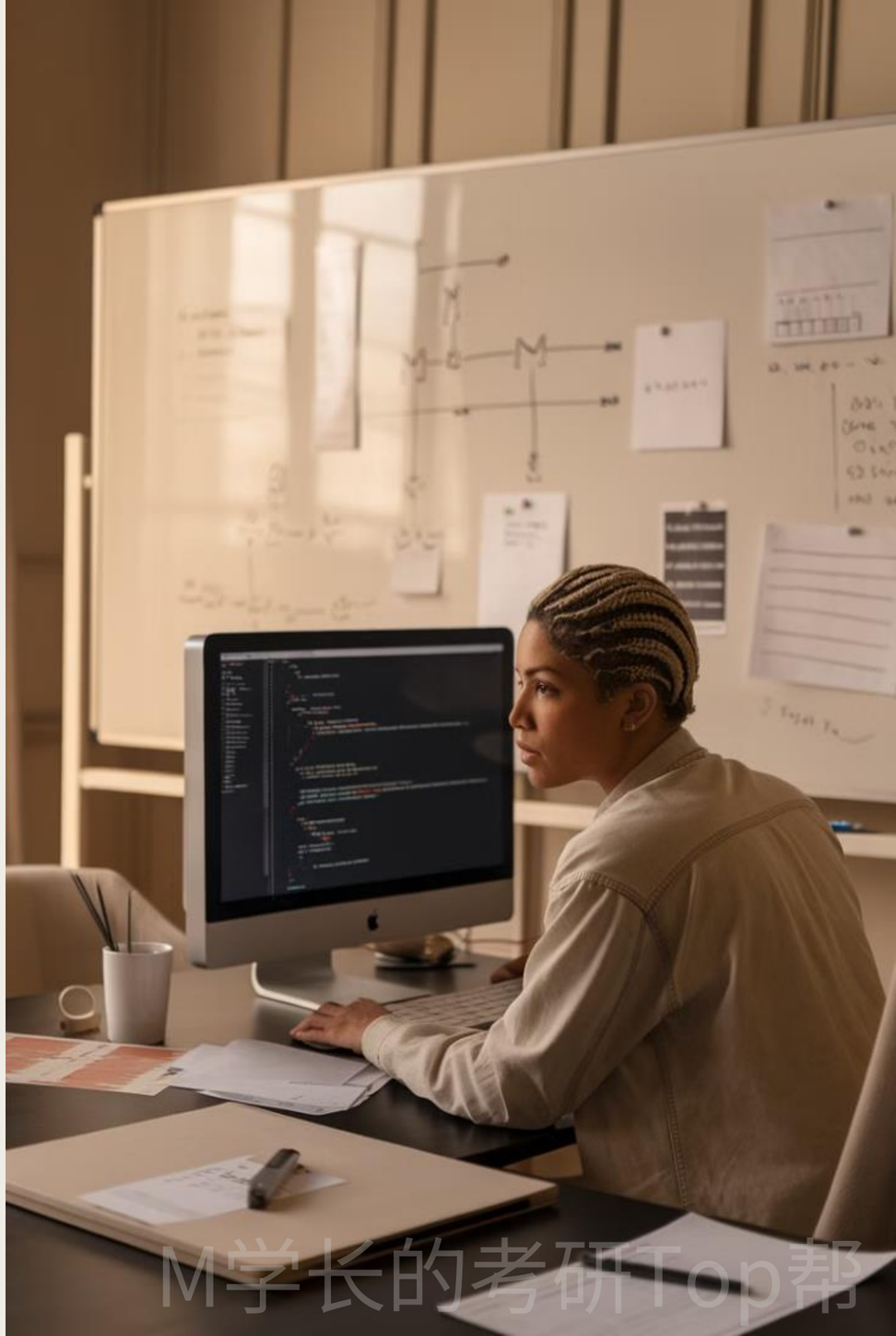
复习重点掌握部分

然后，要复习重点掌握的部分，例如事件驱动模型、线程池、数据库操作等。

3

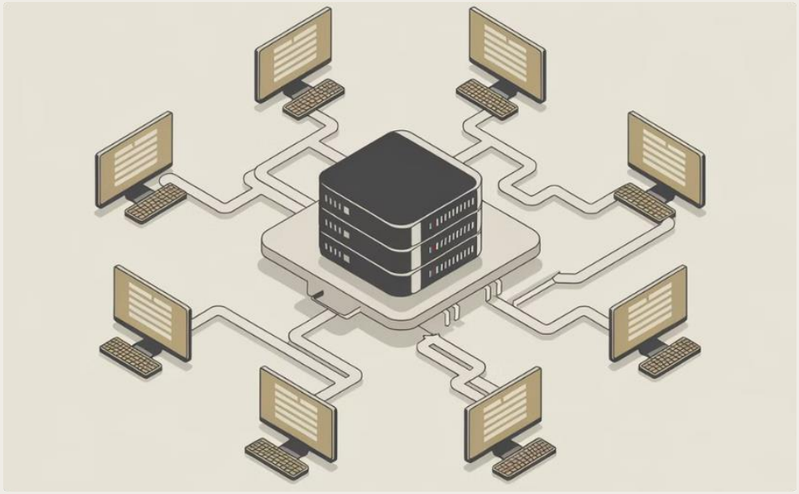
选择擅长的写入简历

最后，选择自己擅长的部分写入简历，并能清晰地解释其原理、优势、代码中的使用以及注释。



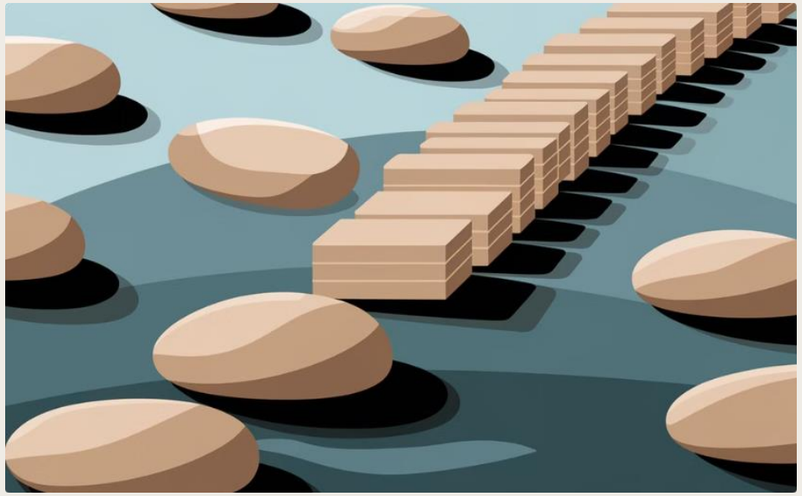
项目基本架构

事件驱动模型与线程池



事件驱动模型

采用 epoll 作为 I/O 多路复用技术，实现非阻塞 I/O 操作，提高服务器处理并发连接的能力。



线程池

通过预创建一定数量的线程并将它们放入线程池中，可以有效地管理和复用线程资源，减少线程创建和销毁的开销，提高服务器响应速度。

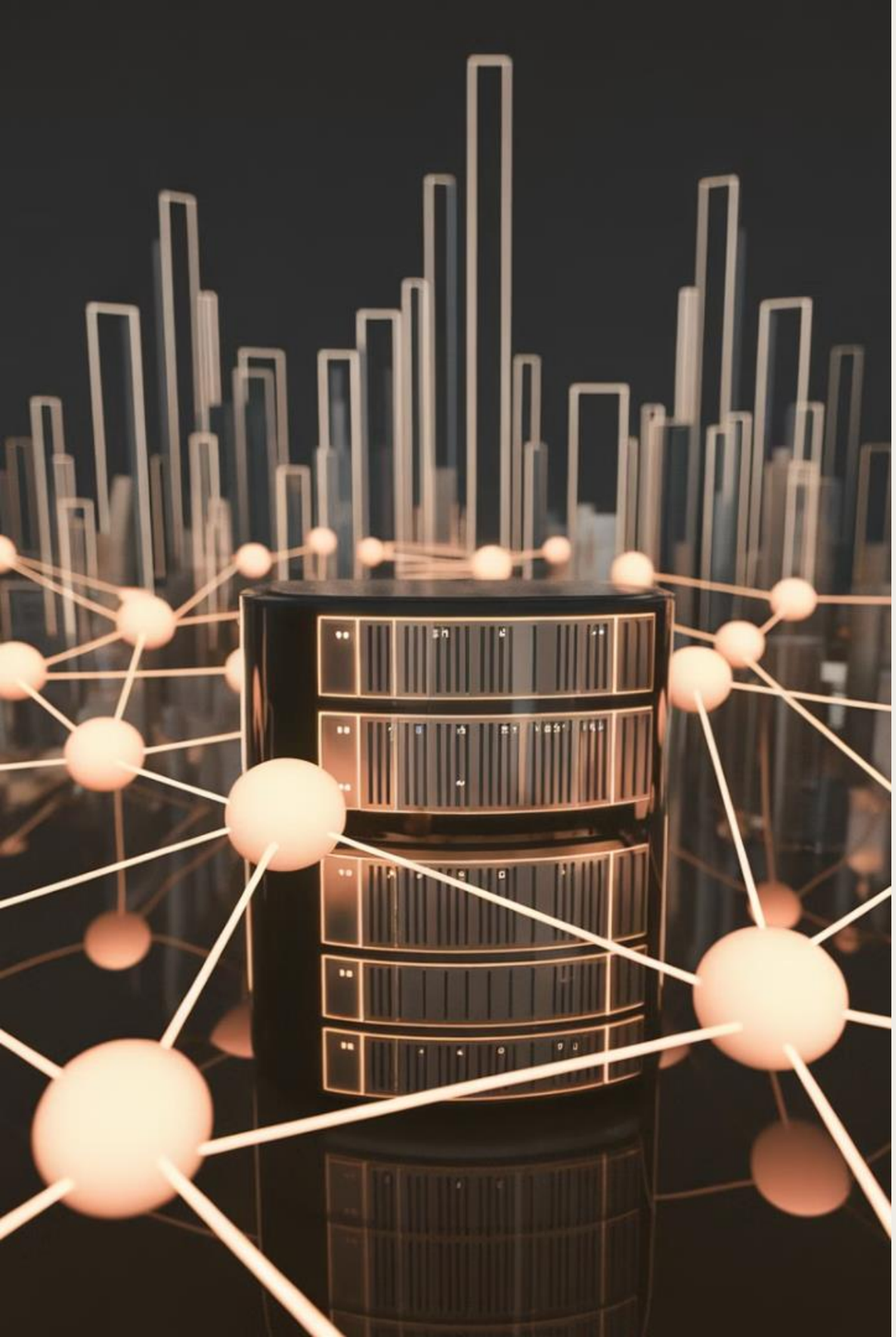
日志系统与前端交互

日志系统

使用宏函数方便地实现Logger，记录服务器运行信息。

前端交互界面

提供静态页面服务和RESTful API，支持与前端页面的交互，实现动态Web应用。



分布式与数据库安全

分布式反向代理

通过与 Nginx 等反向代理服务器配合，实现负载均衡和分布式部署，提高服务器的可扩展性和可靠性。

SQL 防注入

采用预处理语句和参数化查询，有效防止 SQL 注入攻击，保证数据库操作的安全。

MongoDB 数据库

引入 MongoDB 作为大数据数据库，支持高性能、高可用性和易扩展性，适合处理大规模数据存储。

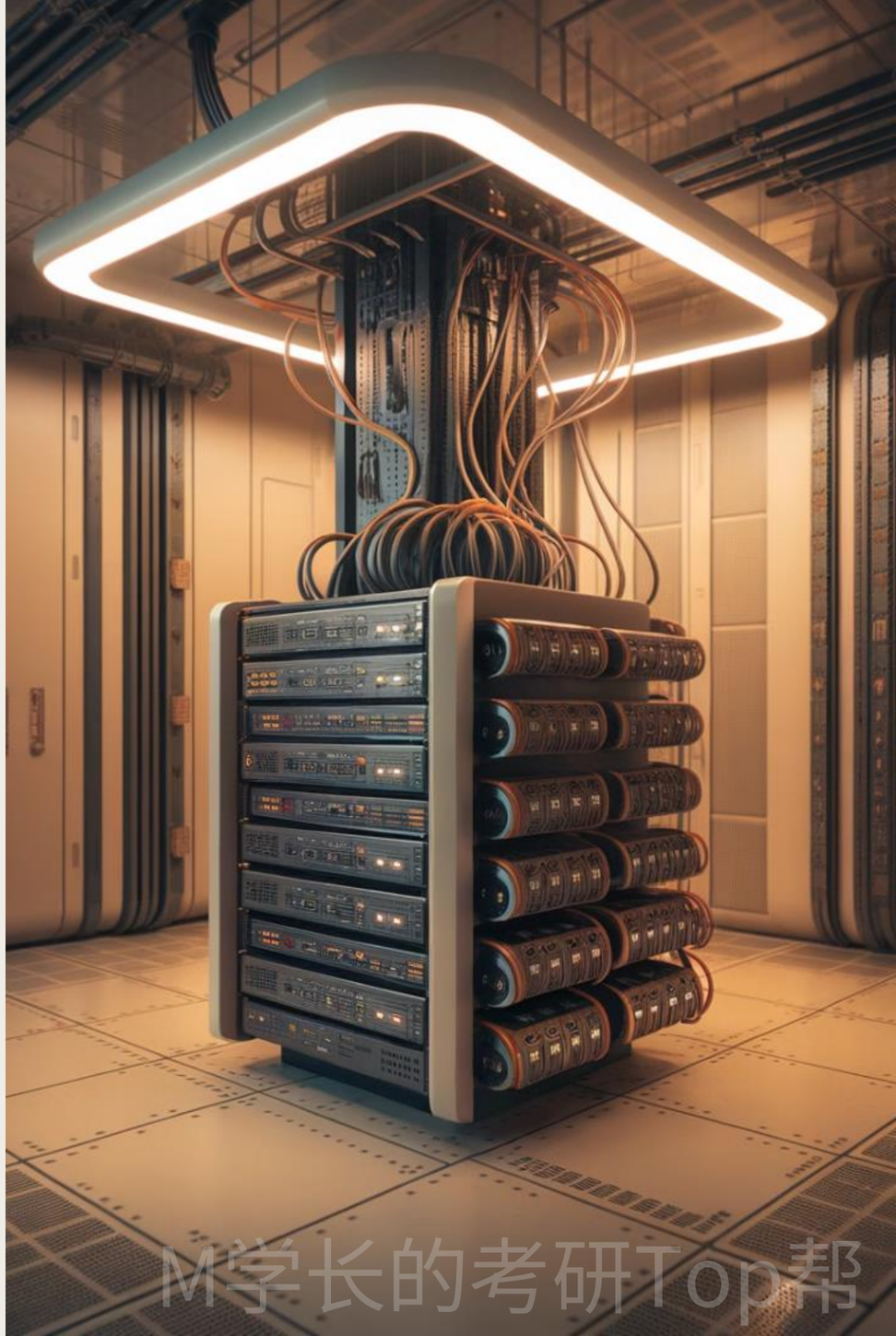
SSL安全性提升

掌握数字证书，数字签名，RSA加密算法的基本原理，使用SSL为服务器提高安全性

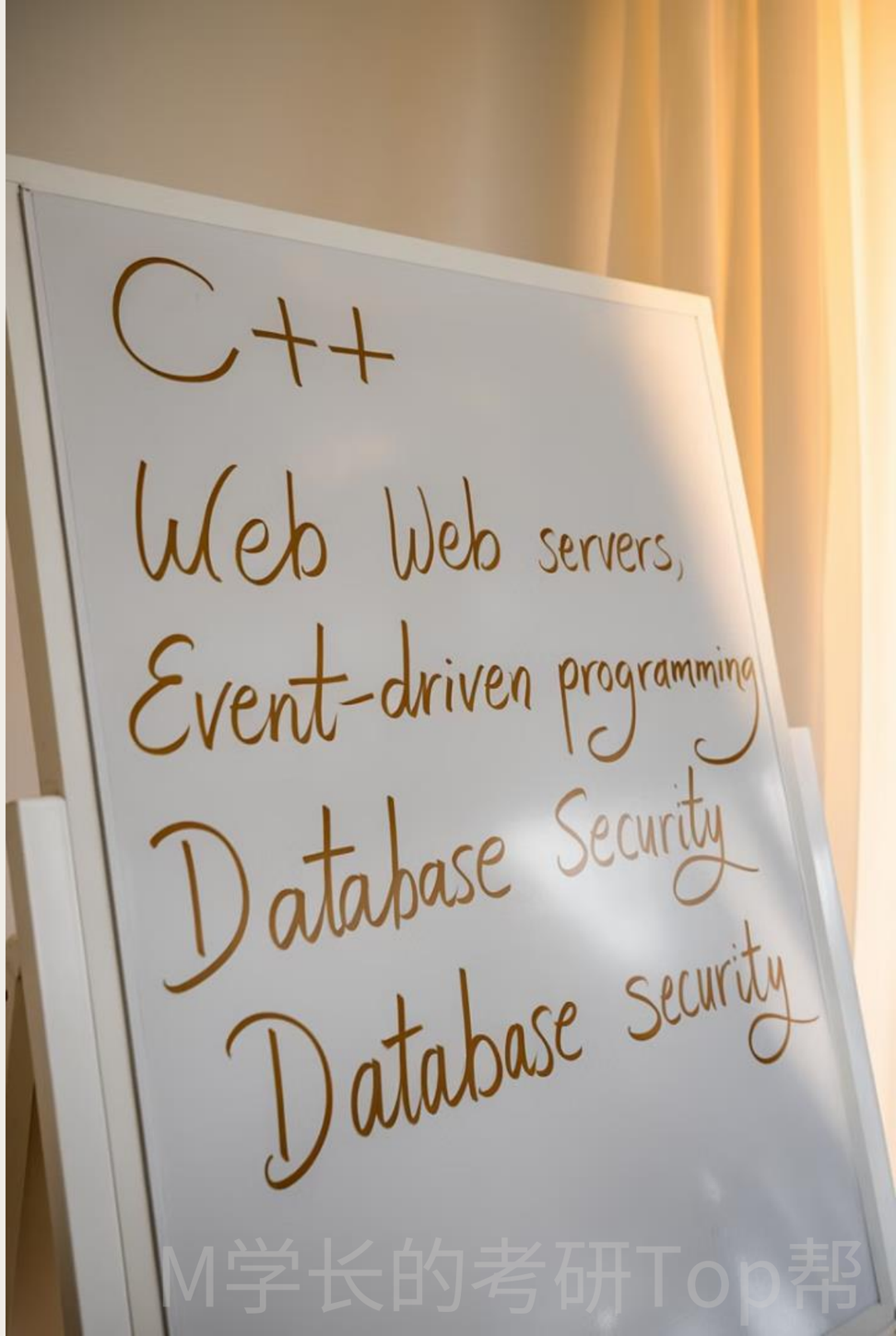
项目总结与技术应用

C++ WebServer 结合了现代 Web 服务器的多项先进技术和实践，包括事件驱动模型、线程池技术、反向代理与负载均衡，以及对新兴数据库的支持。

通过这些技术的应用，服务器能够提供高性能、高可靠性和高扩展性的 Web 服务。同时，服务器还注重安全性，采取了多种措施来防止常见的 SQL 注入攻击。



重要知识点



事件驱动模型

事件驱动模型

服务器通过监听事件来驱动程序运行，而不是顺序执行。

网络IO事件

例如可读、可写事件，触发相应的处理。

epoll 实现 I/O 多路复用

1

创建 epoll 实例

调用 epoll_create1(0) 创建 epoll 文件描述符。

2

注册事件

使用 epoll_ctl 将监听套接字添加到 epoll 实例中，并注册 EPOLLIN 事件。

3

等待事件

调用 epoll_wait 等待事件发生，返回一组发生了注册事件的文件描述符。

4

处理事件

根据事件类型进行相应操作，如接受新连接、读取数据或发送数据。

epoll 能够更高效地处理大量并发连接，因为它仅通知活跃的连接，减少了不必要的检查，提升了性能。

Reactor 与 Proactor 模式

1 **Reactor 模式**

应用程序同步地响应 I/O 事件，并在事件就绪时执行读写操作。

2 **Proactor 模式**

应用程序异步地启动 I/O 操作，并在操作完成后接收通知。

3 **Reactor 模式适用场景**

I/O 操作不会导致阻塞的场景，例如非阻塞 I/O。

4 **Proactor 模式适用场景**

I/O 操作可能导致阻塞的场景，例如阻塞 I/O。

事件循环设计与挑战

事件循环设计

事件循环通过 epoll 高效管理和调度事件，根据事件类型调用相应处理函数。

并发请求挑战

有效分配和管理线程池中的线程，避免锁竞争和提高内存使用效率。

优化措施

使用固定大小的线程池，减少锁的使用范围和持有时间，维护连接状态。

线程池设计

并发处理能力

多线程允许服务器应用程序同时执行多个任务，提高系统吞吐量和服务响应速度。

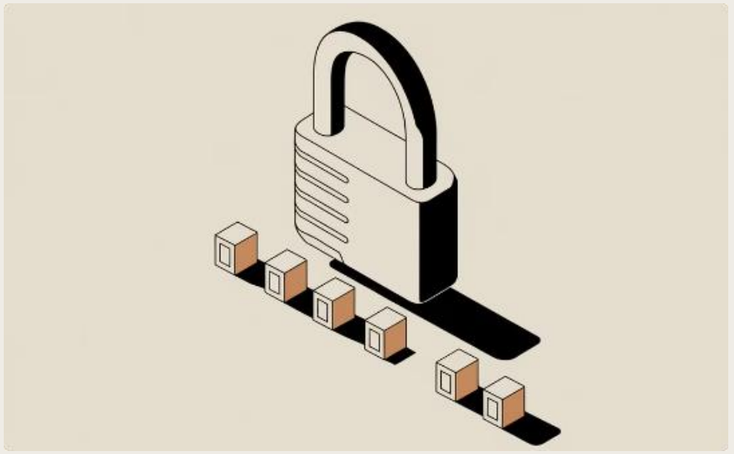
资源复用

线程池创建一组可重用的线程来服务请求，减少线程创建和销毁带来的开销。

任务调度与分配机制

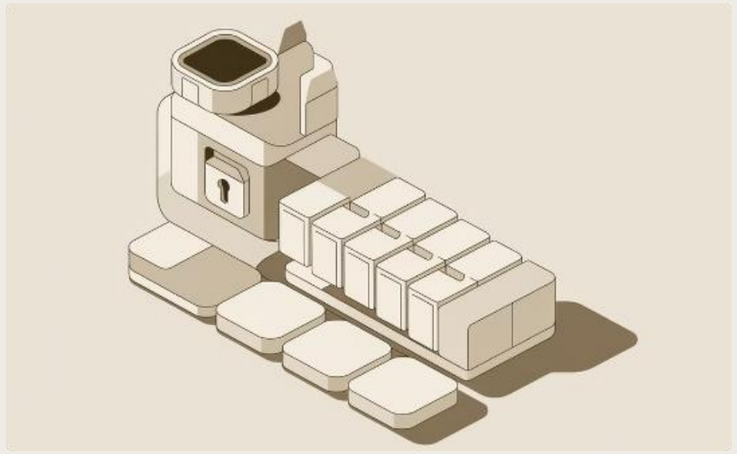
共享任务队列

所有工作线程从一个共享的任务队列中取任务执行。



互斥锁

使用互斥锁queue_mutex保证任务队列的线程安全。



条件变量

使用条件变量condition来等待任务的到来或线程池的停止指令。

性能监控与问题解决

性能监控工具

使用 ApacheBench (ab) 工具对服务器进行基准测试，评估服务器性能。

问题定位

通过增量判断，定位性能瓶颈。例如，引入事件驱动后性能下降，则说明事件驱动逻辑存在问题。

C++ 新特性应用

线程池

std::thread 用于创建和管理线程，简化多线程编程。

Lambda 表达式用于定义线程执行的任务，使代码简洁。

同步与条件同步

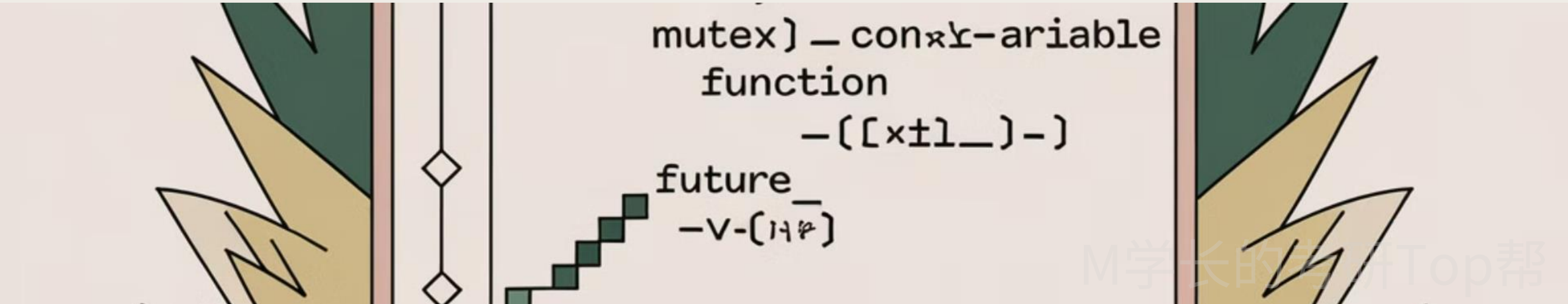
std::mutex 和 std::unique_lock 用于同步访问共享数据。

std::condition_variable 用于线程间的条件同步，有效管理线程间工作流程。

函数操作与异步

std::function 和 std::bind 提供强大的函数操作能力，使任务调度更加灵活。

std::future 和 std::packaged_task 用于异步操作和结果传递。



SQL 注入与安全性

SQL 注入

攻击者通过在查询语句中插入恶意代码片段，篡改原意、获取未经授权的信息或执行非预期的操作。

预防措施

应用程序需要对用户输入的数据进行有效验证和过滤，防止恶意代码注入。

预编译 SQL 语句

1

转换为预编译语句对象

使用 sqlite3_prepare_v2 函数将 SQL 语句转换为预编译语句对象，该对象在数据库内部表示 SQL 语句的结构和操作。

2

固定 SQL 语句结构

预编译过程中，SQL 语句的结构被固定下来，防止外部输入改变 SQL 语句的基本结构，有效防止 SQL 注入攻击。

3

预编译与直接拼接对比

直接拼接 SQL 字符串可能会受到用户输入的影响，而预编译语句不受用户输入的特殊字符影响。

线程安全处理

互斥锁

使用互斥锁（如std::mutex）保护共享资源，确保服务器在多线程环境下的稳定运行。

锁粒度

理论上应该一种数据一个锁，比如用户信息一个锁，文件管理另一个锁，否则效率太低，锁的粒度越细效率越高，但管理也会越复杂。

分布式反向代理与安全防护

1 可扩展性

分布式服务器可以横向扩展，通过增加服务器来提升处理能力和存储容量，应对大规模并发请求和数据增长。

2 高可用性

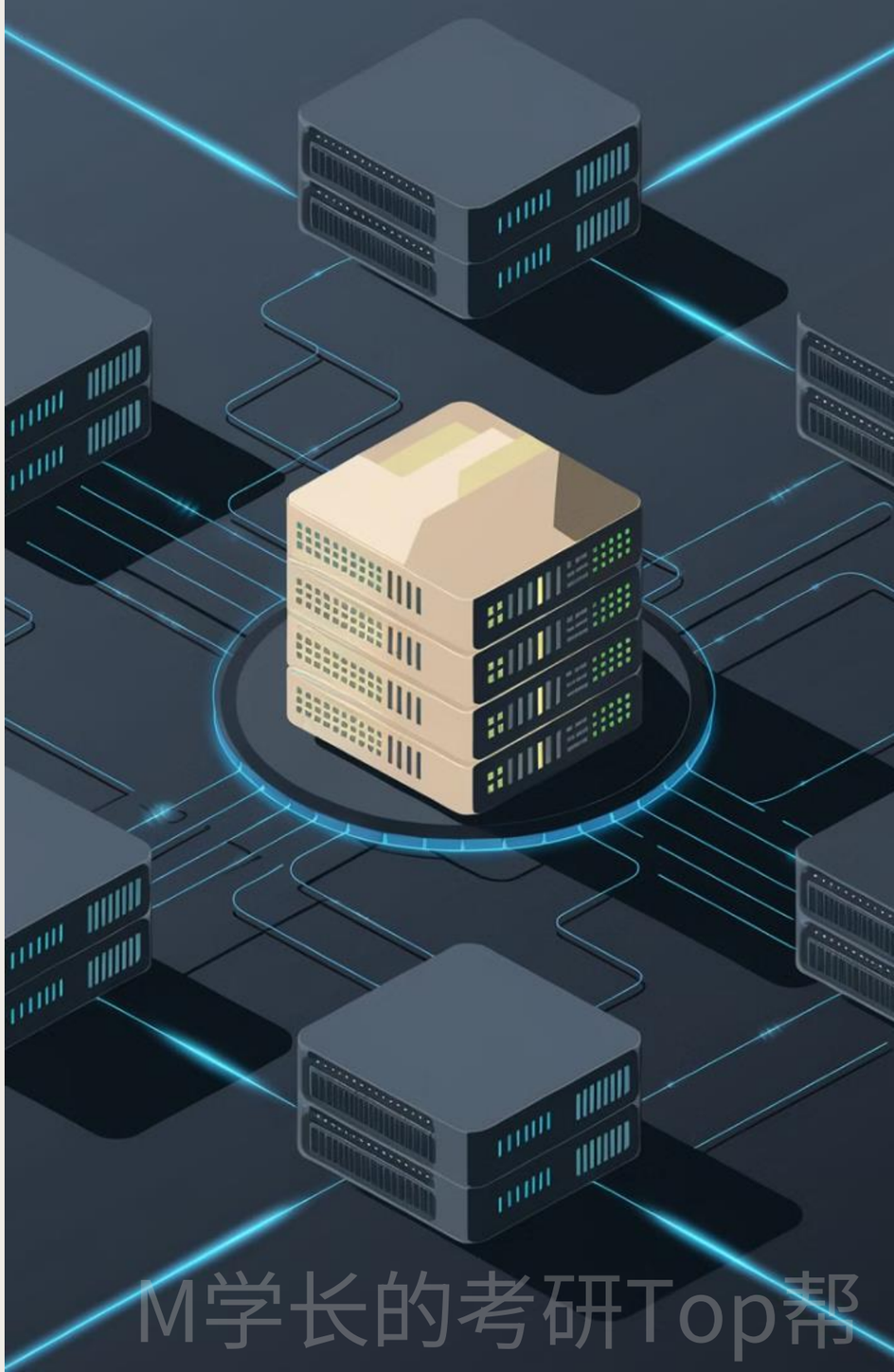
通过数据和任务复制，实现服务的高可用性，即使部分系统组件失败，整个系统仍能正常运行。

3 资源利用率

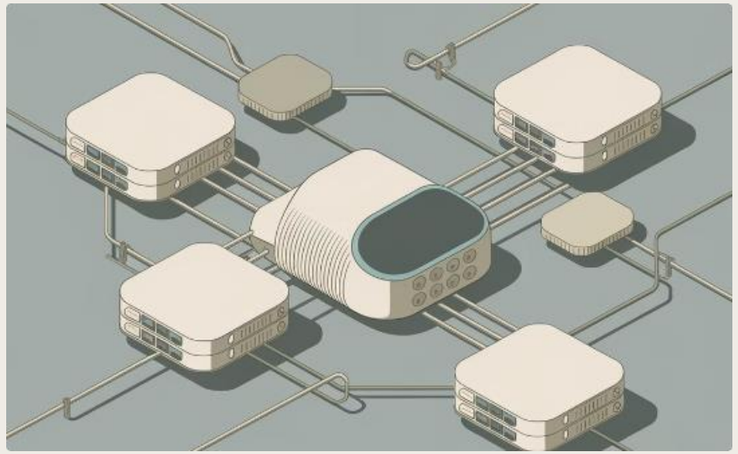
根据实际负载情况动态调整资源分配，优化资源使用效率，提高整体资源利用率。

4 系统维护

每个组件可以独立部署和升级，提高系统维护和升级的灵活性。



分布式部署与高可用性



负载均衡

使用upstream模块定义后端服务器池，根据权重分配请求，实现负载均衡。

故障转移

当后端服务器不可用时，Nginx自动停止发送请求，将请求转发至其他健康节点，保证服务可用性。

代理缓存

配置代理缓存，对特定路径的GET响应进行缓存，减轻后端服务器压力，提高性能和响应速度。

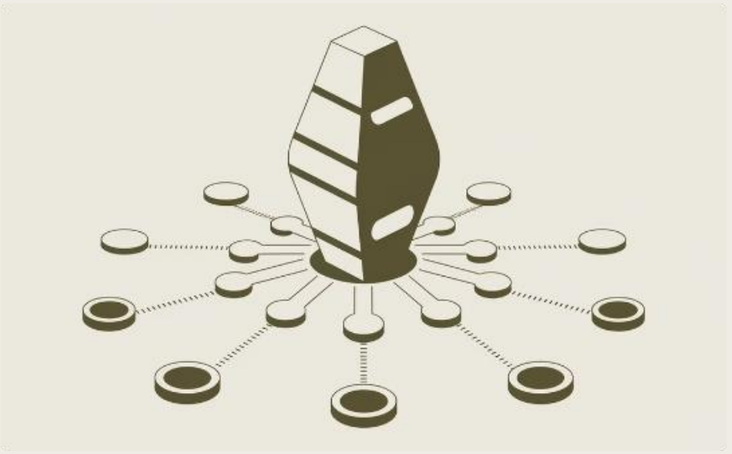
MongoDB 数据库集成

高性能

MongoDB 的文档数据模型允许存储复杂的嵌套数据结构，提高查询速度。

高可用性

通过副本集实现高可用性，保证数据库服务不中断。



易扩展性

支持水平扩展，通过分片技术可以将数据分布在多个服务器上。

前端交互支持

静态页面服务

服务器通过配置路由和处理函数来响应客户端请求，返回静态HTML页面。

RESTful API 设计

遵循清晰的资源定位、标准HTTP方法、无状态和统一响应结构原则。

什么是RESTful API

资源

RESTful API 中，一切都被视为资源，每个资源都有唯一的 URI。

无状态

每次请求之间相互独立，服务器不存储之前请求的状态信息。

统一接口

使用标准的 HTTP 方法对资源进行操作，简化交互。

可缓存

响应结果可以被标记为可缓存或不可缓存，提高性能和效率。

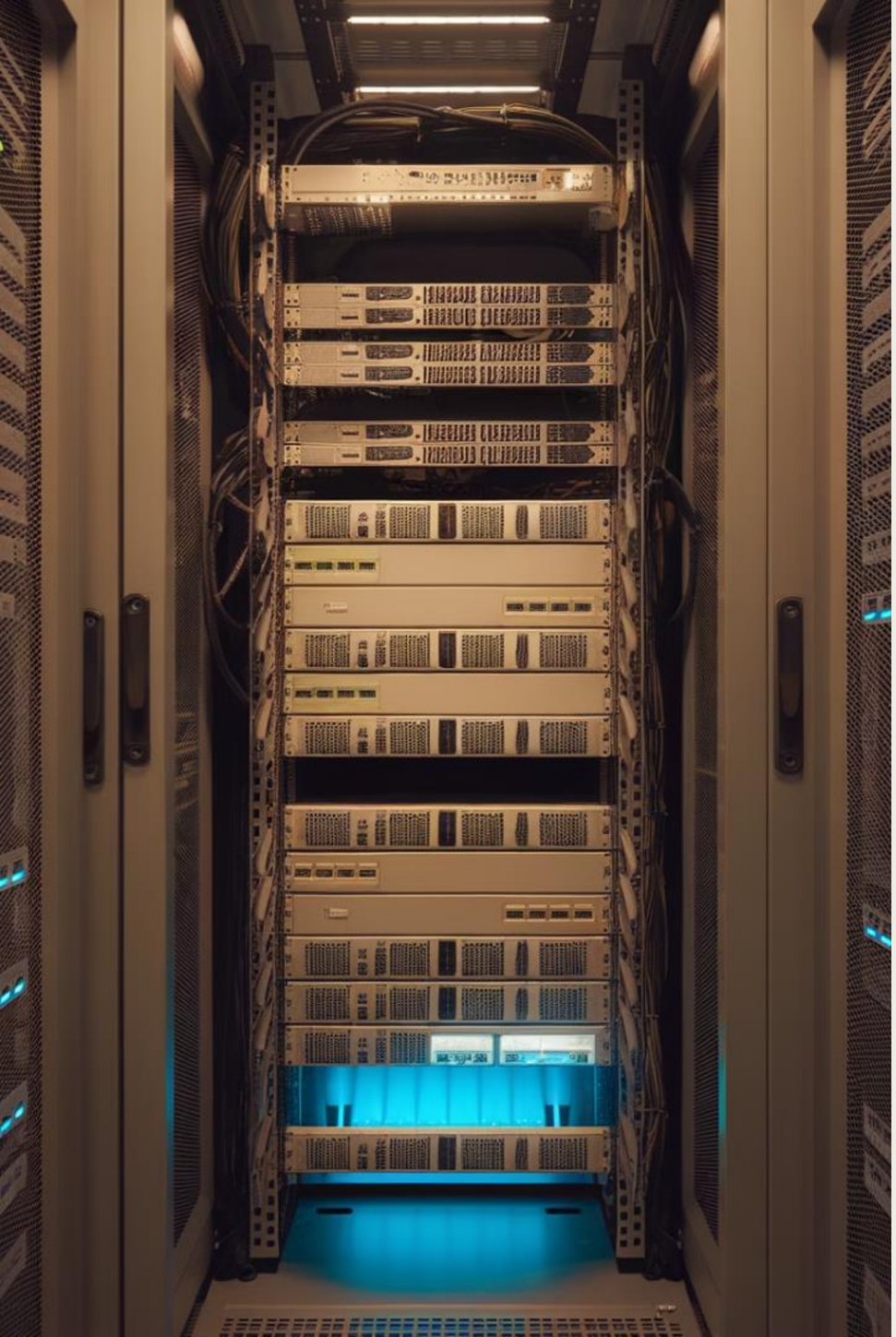
项目介绍

这是一个独立开发的 C++ 分布式高性能 web 服务器，运行环境为 Ubuntu。

服务器采用事件驱动模型和线程池来提高性能，并使用 C++ 新特性提升代码安全性和简洁性。

通过 Nginx 实现分布式，数据库方面前期使用 SQLite，后期升级为 MongoDB。

支持文件上传和下载，并通过预防 SQL 注入来保障数据库安全。



服务器性能

请求数量	100 万
完成时间	1 分钟
50% 请求	5 毫秒
90% 请求	8 毫秒
99% 请求	21 毫秒
最长请求	79 毫秒

服务器部署与 Docker

1 部署位置

服务器部署在本机 Docker 环境中。

2 选择 Docker 的原因

方便环境迁移，学习服务器开发，节省配置时间。

3 Docker 的优势

轻量级、高效、便携、隔离性。

项目动机

提升技术能力

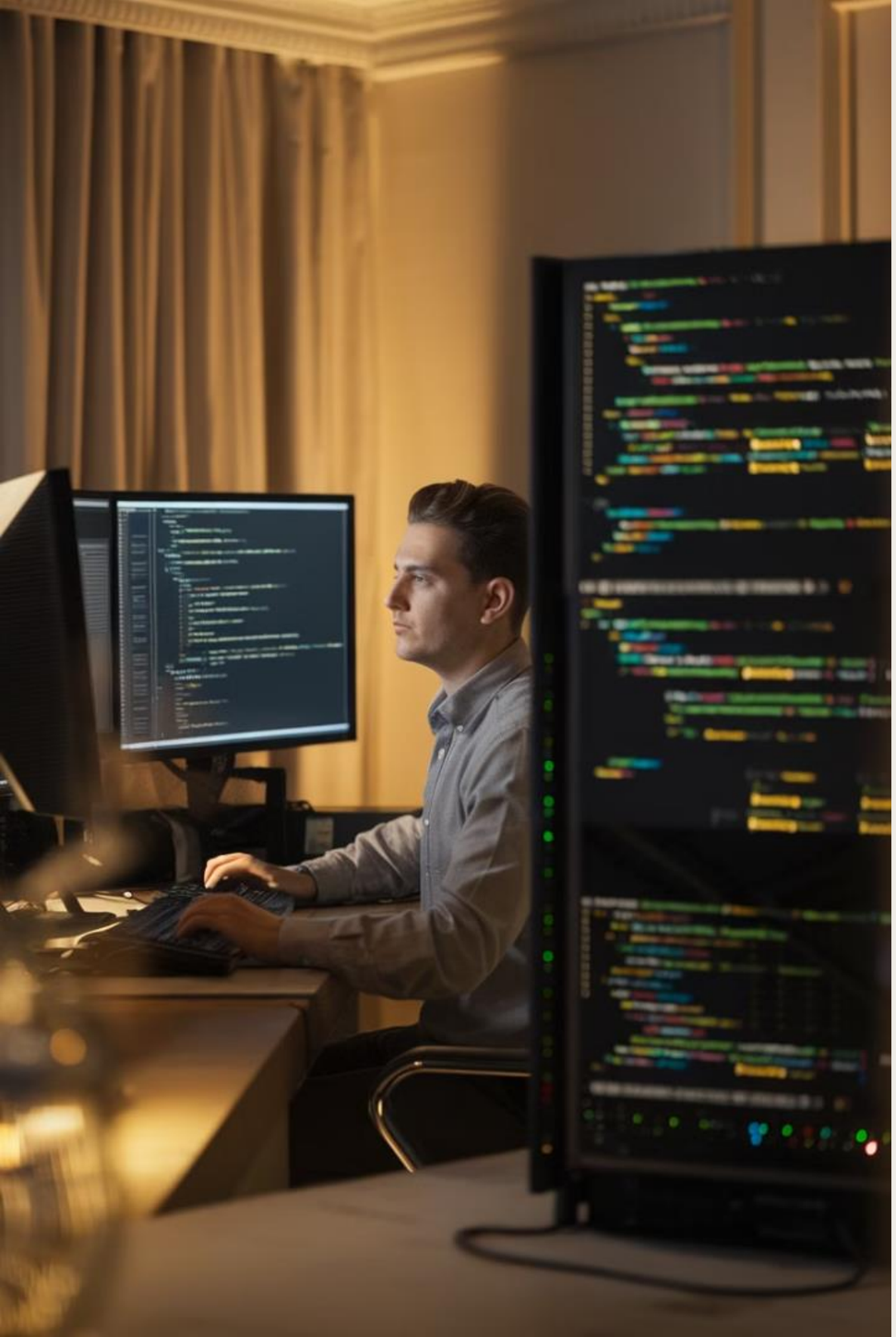
希望在读研前提高自己的技术开发能力，将考研学习的理论知识付诸实践。

实践学习

通过独立开发一个服务器项目，深入理解网络通信流程、前后端逻辑和多线程等概念。

未来发展

为未来从事互联网工作打下基础，积累项目经验。



学习过程

1 语法学习

通过网课、教程书籍学习 C++ 语法。

2 服务器开发

阅读 Unix/Linux/C++ 服务器网络编程书籍和文章，学习服务器运行原理。

3 实践项目

参考博客、书籍，独立开发简单的服务器，逐步添加功能。

4 知识扩展

学习 C++ 新特性、多线程开发、事件驱动模型和分布式知识。

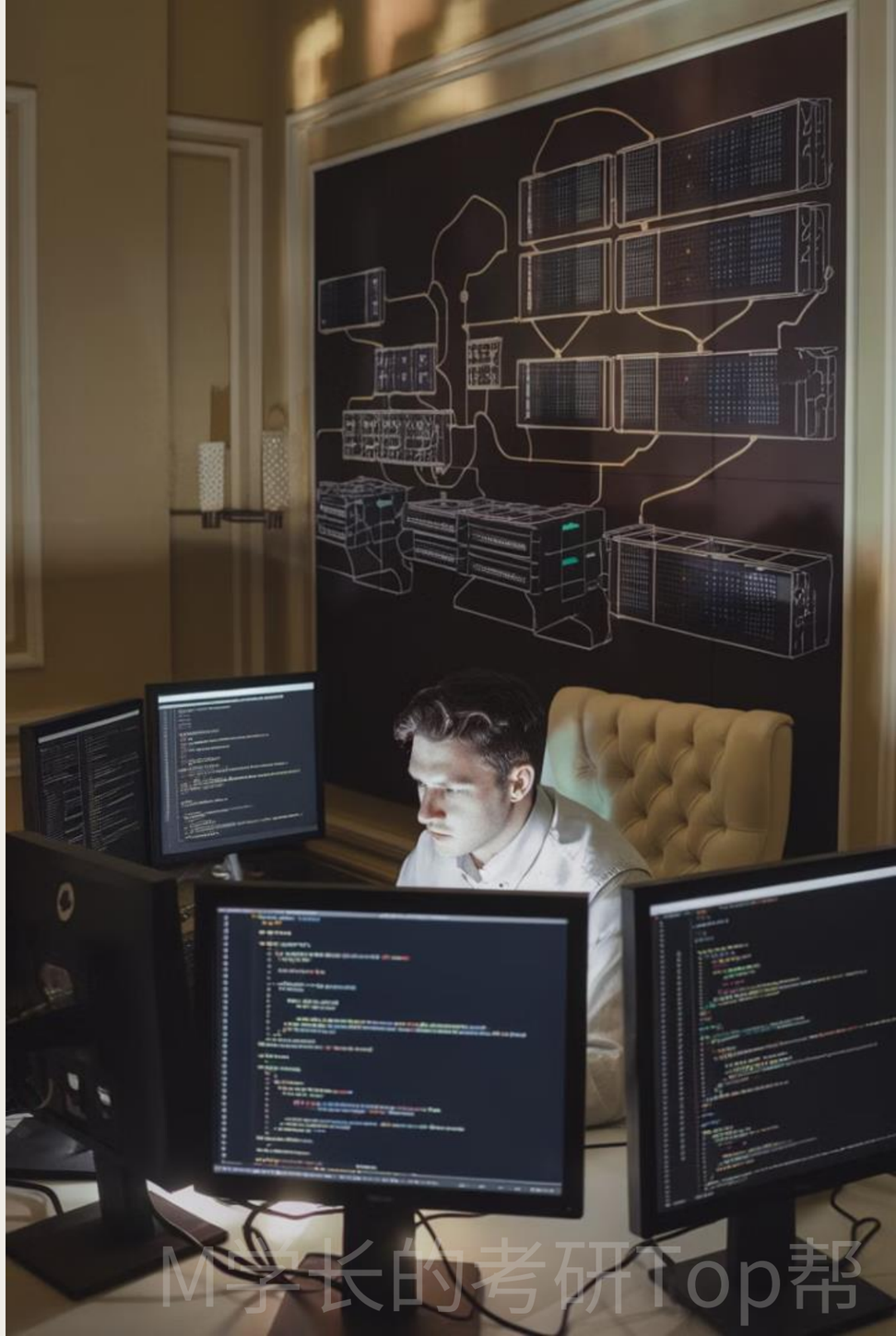
开发困难与解决

服务器崩溃

单个服务器崩溃导致整个系统瘫痪，通过分布式架构解决。

代码能力

代码能力不足，通过查阅资料和博客学习C++新特性。



Bug 处理

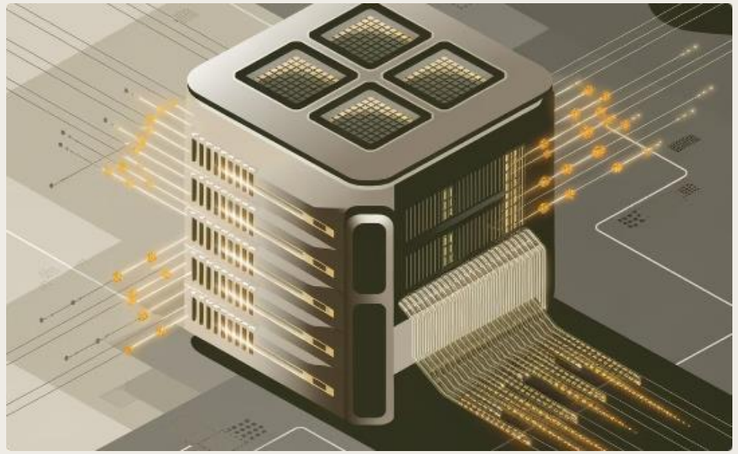
调试方法

主要依靠日志系统定位问题，偶尔使用GDB调试。

常见问题

新特性语法不熟悉，或者项目运行逻辑有问题。

业务需求与架构设计



高性能

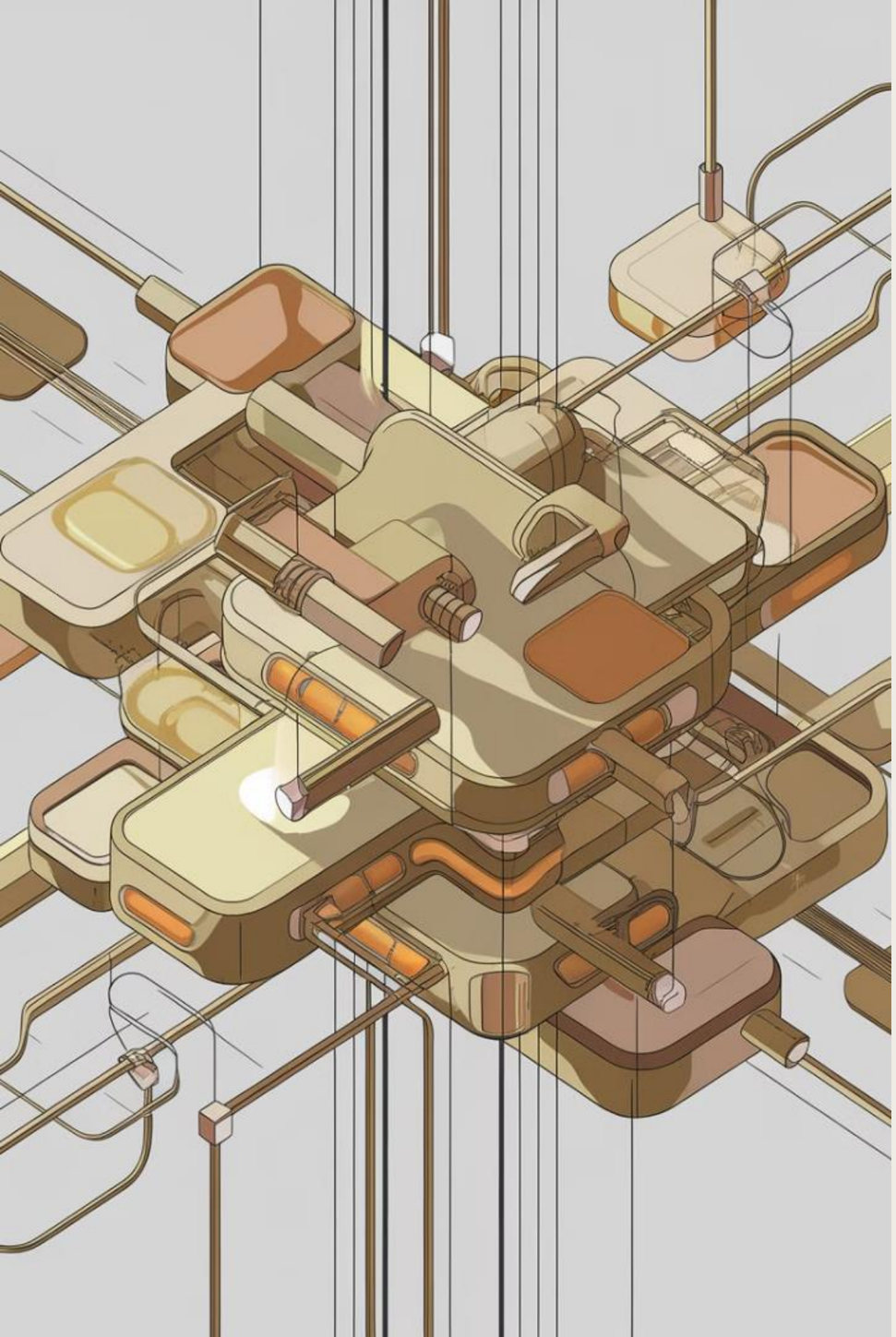
服务器需要快速响应请求，满足高并发需求。

安全性

服务器需要防止攻击，确保数据安全。

可扩展性

服务器需要能够应对未来业务增长，灵活扩展。



如何降低耦合度

模块化设计

将不同功能模块分离到不同文件，降低代码耦合度。

灵活扩展

每个文件负责一个模块，便于长远开放。

代码组织

日志、响应、请求、线程池等库函数分离。

项目代码量

项目总的代码量前后端加起来有一二三四千行吧，实际上我们项目大概一千多行，但那是因为我代码风格比较精简，同时前端文件没有算进去，你个人的项目，老师看你跨考或者经验不多的情况下初次写一两千行已经很厉害了，太多老师也不太会信。

未来优化方向

性能优化

线程池引入线程优先级，提升性能。

数据库安全

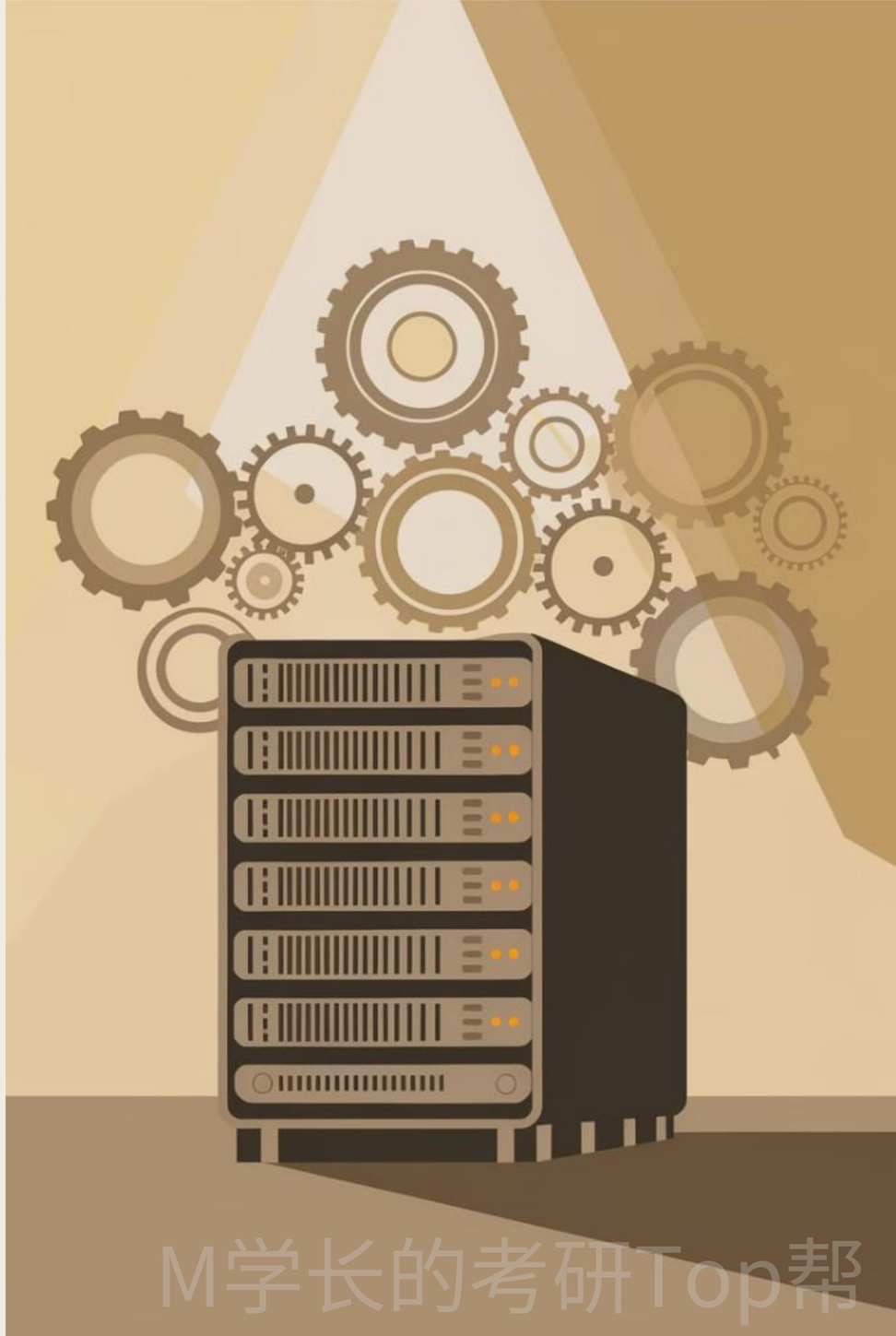
引入更多防止SQL注入机制，增强安全性。

分布式管理

优化分布式管理策略，实现负载均衡。

资源池管理

数据库/http连接采用池管理，提高效率。



解决困难的方法

搜索引擎

遇到语法错误或bug，可以尝试使用搜索引擎或技术论坛查找解决方案。

理论学习

对于概念或编程方面的问题，可以查阅相关理论知识，例如阅读书籍或博客文章。

寻求帮助

可以向学长学姐、同学或朋友寻求帮助，共同解决问题。

日志调试

通过记录日志信息，可以帮助分析问题并找到解决方案。

未完成的工作

代码质量

目前还没有引入代码审查、单元测试或静态代码分析等实践。

备份数据库

目前还没有使用大数据进行备份数据库。

面试技巧

自信

对项目内容了如指掌，清楚地回答问题，不要犹豫。

诚实

不会的问题就承认不会，不要不懂装懂，坦诚地说出学习计划。

展示学习能力

强调项目是自学完成的，突出独立学习的能力。

面试沟通策略

理解问题

没听懂问题就表达自己的理解，询问对方是否询问的是xx，避免答非所问。

引导话题

引导老师到自己想回答的地方，例如，在回答简历相关问题时，可以顺势提及自己的C++服务器项目，展示动手能力。

主动展示

即使老师没有提问，也可以主动展示项目相关内容，例如，在遇到困难时，可以介绍自己的实现方法和相关理论知识。

面试应对策略

时间管理

不会的题目不要浪费太多时间，直接说不会，**并**询问是否可以换题或得到提示。

深入理解

不要给出简短的答案，尽量展现对项目的理解**和相关知**识，并结合具体实践。

沟通技巧

在回答问题时，要展现出对项目的理解，而**不是死**记硬背。

面试应对策略

1 口述练习

面试前，要自己口述一遍回答问题，不要只看两眼就觉得会了。

2 代码逻辑

代码部分，不用力求每行代码都完全理解，重点是能讲清楚重点功能的代码逻辑。

3 口述注释

面试时，可以将代码逻辑看作口述注释，或者介绍函数逻辑。

自学项目推荐

CSAPP

CSAPP 是一个非常经典的项目，可以帮助新手提高写代码能力和对后端理解。

http://csapp.cs.cmu.edu/3e/labs.html

谢谢大家