**个人作业2：软件架构的选择——C/S架构**

**202214146189 周佳霖**

**1 软件架构概述**

客户端-服务器架构（C/S模式）：

分为客户端（请求服务）和服务端（提供服务）两个部分。二者通过网络进行通信。

其中，客户端通常是用户交互的界面，可以是浏览器、移动应用或者桌面应用。客户端的主要职责包括发起请求、显示结果以及进行用户交互。发起请求是指客户端通过网络向服务器发送请求，显示结果是接收并展示服务器返回的数据，用户交互则提供友好的用户界面，允许用户输入并进行操作。

服务器是提供资源和服务的一方，通常具有强大的计算能力和存储能力。服务器的主要职责包括接收请求、处理请求和返回响应。接收请求是指服务器接收并解析来自客户端的请求，处理请求则是执行相应的业务逻辑、查询数据库或调用其他服务，返回响应是将处理结果封装成响应返回给客户端。

工作原理主要是以下几个步骤：首先客户端向服务器发起请求，请求中包含有所需操作和相关数据，服务端接到请求之后进行解析和验证，然后执行服务的逻辑，最后响应返回给客户端，客户在客户端进行结果的查看。  
**2 应用场景**

C/S架构主要的应用场景是各种网络应用和分布式系统。应用场景举例如下：

（1）最典型的应用场景——web应用：客户端浏览器向web服务器发送http请求，服务器处理请求之后返回相应的HTML页面、CSS样式和JavaScript脚本。

（2）移动应用：移动设备上的app作为客户端向服务器请求服务，如社交媒体应用、即时通讯应用和在线购物应用等。

（3）企业信息系统：企业内部的电脑或终端设备作为客户端向服务器发送请求，如客户关系管理系统、企业资源计划系统等。

（4）在线游戏：负责图像渲染和用户输入的客户端向处理游戏逻辑、玩家数据同步、多玩家交互的服务器发送请求。  
**3 优点和缺点**

优点：

1. 容易对系列服务进行建模，供客户端请求。
2. 中央化管理，服务器集中管理数据和业务逻辑，便于系统维护和更新，进行集中式的修改。
3. 安全，由于中央化管理，可以在服务器端设置一些安全措施来保证数据安全，减少数据风险。
4. 可扩展，由于中央化管理，可以通过增加服务器数量和增强服务器配置来扩展整个架构系统，应对更多的客户端请求。
5. 数据一致，数据集中存储在服务器，通过数据库机制可以保证数据的一致性和完整性。

缺点：

1. 高负载，请求通常是在服务器的不同线程中进行响应的，因为不同客户端有不同形式，进程间通信会造成很大的负载。
2. 单点故障，由于中央化管理，一旦服务器出现故障可能会对整个系统造成很大的影响。
3. 网络依赖，通信依赖网络，如果网络状况不佳，可能会影响系统的性能和用户体验。
4. 高成本，构建和维护高性能服务器需要较高的成本。
5. 延迟问题，通信耗费的时间可能对一些实时性要求高的应用的响应速度造成影响。  
   **4 技术栈**

**客户端技术栈：**

1. 桌面应用

* C++/Qt
* Java（JavaFX、Swing）
* C#（.NET Framework、WPF）
* Python（Tkinter、PyQt）

1. 移动应用

原生开发：

* Android：Java/Kotlin
* iOS：Swift/Objective-C

跨平台框架：

* React Native
* Flutter
* Xamarin
* Apache Cordova

1. Web应用

* HTML、CSS、JavaScript
* 框架和库：
  + React.js
  + Vue.js
  + Angular
  + Svelte

**服务器技术栈：**

1. 后端语言

* JavaScript/TypeScript（Node.js）
* Python（Django、Flask、FastAPI）
* Java（Spring Boot、Quarkus）
* PHP（Laravel、Symfony）
* Ruby（Ruby on Rails）
* C#（ASP.NET Core）
* Go（Gin、Beego）
* Rust（Actix、Rocket）

1. 数据库

* MySQL
* PostgreSQL
* Oracle
* SQL Server

1. 应用服务器

* Apache HTTP Server
* Nginx
* Tomcat、Jetty
* Express.js（Node.js）

**通信技术栈：**

协议：

* HTTP/HTTPS
* WebSocket
* gRPC
* MQTT

**安全技术栈：**

* SSL/TLS（HTTPS）
* 防火墙、API安全网关

**5 市场使用情况**

1. Amazon（AWS Management Console）

AWS 管理控制台是 Amazon 提供的 Web 应用界面，用于管理其云服务。其使用客户端-服务器架构，其中客户端使用React.js和其他现代Web前端框架，提供动态、响应式的用户界面，服务端采用微服务架构，由AWS自身的服务（如 API Gateway、Lambda）提供支持，之间通过REST API和WebSocket进行数据交互通信。

该应用主要用于云资源的管理和监控，功能强大，集成AWS生态系统，支持大规模、高并发。

1. Slack

Slack是一款面向团队协作的实时消息和项目管理工具，提供聊天、文件共享、语音通话等功能。其使用客户端-服务器架构，其中使用Electron构建跨平台桌面客户端，React Native构建移动客户端，使用WebSocket提供实时消息同步，基于微服务架构实现高扩展性。

该应用主要用于项目管理和文档共享以及团队企业协作，界面友好、实时性强，集成功能丰富。

1. Steam

Steam是全球最大的游戏平台，提供游戏购买、下载、更新和社交功能。其使用客户端-服务器架构，其中使用 C++ 开发本地应用，与服务器通信进行用户验证、游戏管理等，后端采用分布式服务，提供全球游戏分发。之间使用 HTTP/S 和WebSocket支持游戏库同步和实时聊天。

该应用主要用于游戏的购买和管理，有着稳定的全球游戏分发能力，功能多样。

1. Vue.js

Vue.js是用于实现C/S架构的工具框架，是一个前端框架，负责客户端（Client）的开发，主要用于与用户交互和构建视图层。在典型的C/S架构中Vue.js扮演客户端的角色。其运行在用户的浏览器中，完全属于客户端环境，使用AJAX或Fetch API通过HTTP/HTTPS协议向服务器发起请求，与后端进行通信并获取数据。

1. Springboot

Springboot是用于实现C/S架构的工具框架，是一个后端开发框架，负责服务器端（Server）的逻辑处理和数据管理。在C/S架构中，Spring Boot扮演服务器的角色。其运行在服务器（如云服务器、物理服务器）上，处理客户端的请求。  
**6 个人感受**

客户端-服务器架构作为网络应用和分布式系统的基石，已经在各个领域得到广泛应用。随着技术的进步，C/S架构也在不断地演进和进步：云计算的普及使得服务器资源可以按需分配和扩展，降低了企业的IT成本；边缘计算则通过将计算和数据存储分布到靠近用户的边缘节点，减少了延迟，提高了系统的响应速度；微服务架构通过将单一的应用拆分成多个小的服务，每个服务负责特定的业务功能，这样可以提高系统的灵活性和可维护性，并且便于服务的独立部署和扩展；服务器无服务器架构允许开发者编写代码并将其部署在云平台上，而无需关心底层服务器的管理和维护。这样可以进一步简化开发流程，提高开发效率。

想要在目前的开源架构上进行进步实现，我们需要有一些基础：需要学习客户端架构如vue等的相关知识基础，学习服务器端如springboot等的相关知识基础。除此之外，还要了解通信和网络基础，数据库和存储基础。在掌握这些基础知识之外，再提高自己的架构设计能力，掌握前后端协作流程，设计良好的 API 契约，学习分布式系统的基本原理，根据业务需求选择合适的架构模式。