HyperCraft

软件架构文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 16/5/2023 | 1.0 | 初稿 | 袁翊天、夏嘉阳、任柏、黄予晗 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 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 5

4. 进程视图 6

5. 部署视图 6

6. 实现视图 7

7. 技术视图 7

8. 核心算法设计 8

8.1 体素贪心网格化算法 8

8.2 方块更新算法 8

9. 质量属性的设计 8

软件架构文档

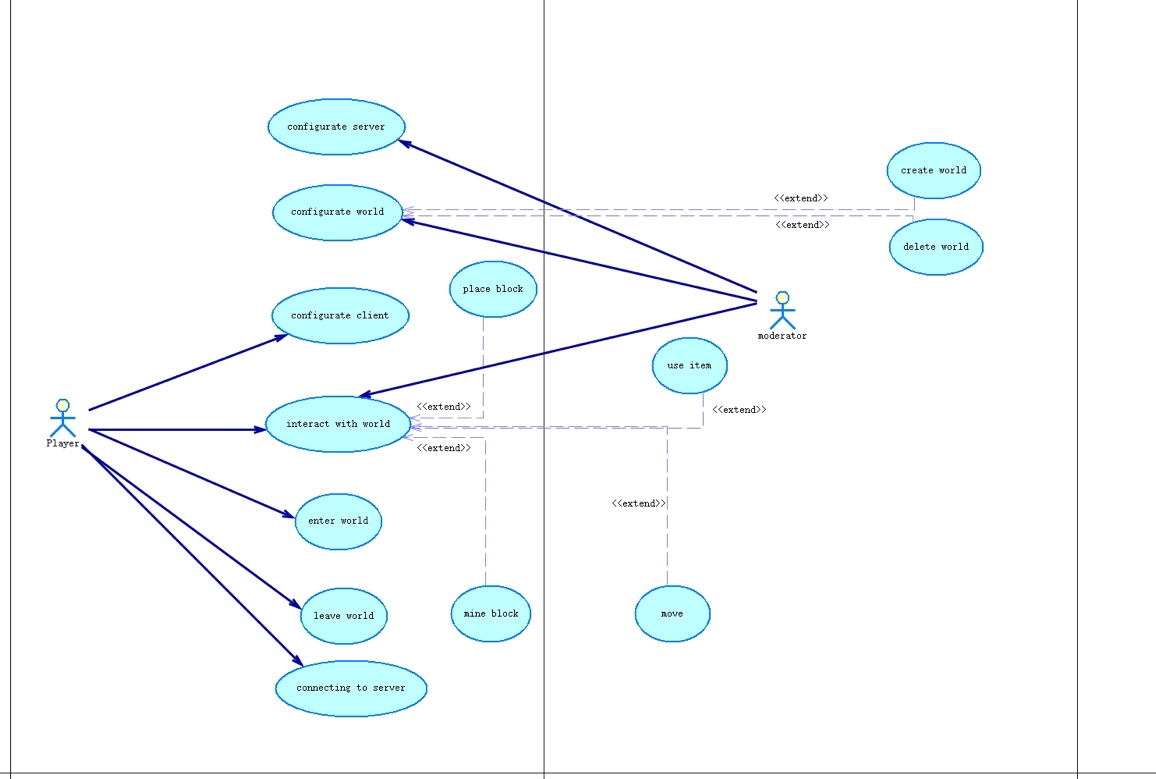
# 简介

## 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。

本文档将从用例、逻辑、进程、部署、实现、技术等视图为切入方面，简要介绍 HyperCraft 游戏的软件架构与设计思路。本文档旨在从系统架构层面为游戏的开发提供方法论，也可以作为想要制作同类游戏的开发者的借鉴。本文档面向的读者是具有一定游戏开发经验、熟悉 Vulkan 等游戏开发框架的开发者。

# 用例视图



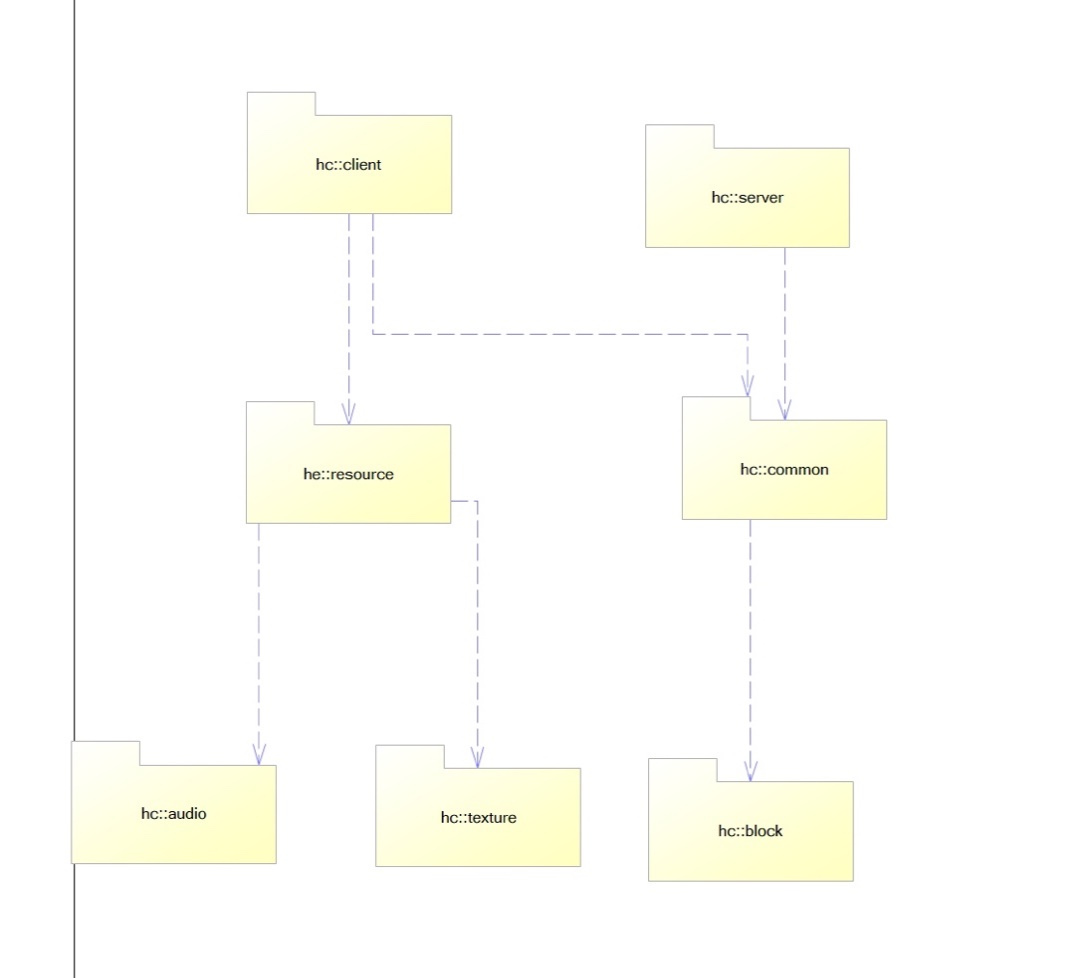
用例视图展示了 HyperCraft 游戏与外界交互的系统模型。上图是HyperCraft 游戏系统启动时一个典型的用例场景。

HyperCraft 面向的用户有两类：普通玩家（player）和管理员（moderator）。玩家作为游戏的服务主体，可以直接与游戏世界进行交互、进入和退出世界。为支持联机功能，HyperCraft 还为玩家提供了配置和启用个人客户端、连接到服务端等服务。管理员是拥有更高权限的系统管理者，HyperCraft 提供配置服务器端和配置世界属性（包括创建和修改世界）的接口，方便管理员发布更新内容、改善游戏体验。管理员还被允许直接与世界交互，便于观察世界和管理玩家行为。

在一个典型的用例场景中，玩家通过注册和登录个人账号来配置客户端，然后选择进入世界，向服务器端发出请求。服务器端可以选择相应玩家的请求，判断玩家的请求类型（新建世界还是重入已有的世界），然后建立和客户端的连接。连接完成后，玩家将获取到生成游戏世界所需要的必要资源，在客户端主机上进行世界生成，进而和世界进行交互。玩家可以随时选择退出世界，服务器端判断是否需要保存玩家档案以供下一次重入时恢复玩家状态，然后关闭服务器端和客户端的连接。管理员在联机过程中，可以观察用户的坐标、世界的状态等，如有需要，可以强制将用户移除游戏（断开连接）。管理员也可以随时配置服务器端状态和世界内容。

# 逻辑视图

## 概述



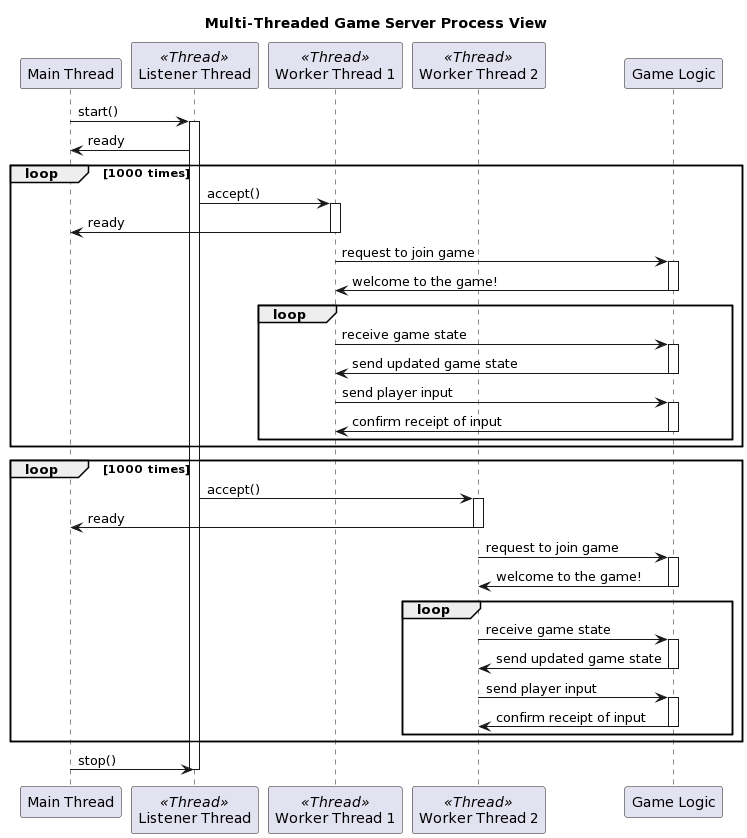
HyperCraft 的逻辑架构整体采用客户端-服务器架构（Client-Server Architecture）风格。对于一个支持联机的高性能开放世界游戏，选择此架构风格是自然的。客户端负责用户界面和用户交互，服务器负责处理请求、存储数据等。客户端通过网络与服务器通信，实现分布式计算和资源共享。此外，为了提高开发效率，增强代码的可复用性，我们依据面向服务架构（Service-Oriented Architecture，SOA）将系统划分为一组松耦合的服务，具体来说也就是 C++ 中的一系列功能类。

hc::client 和 hc::server 是本系统最核心的两个设计包，它们分别为客户端和服务端提供相应的服务。多个子功能由相应的模块实现，每个模块提供特定的功能，并通过标准化的接口进行通信。整个系统的设计强调可重用性和组合性，以实现灵活的业务流程和集成。

## 在构架方面具有重要意义的设计包

hc::client 是为玩家提供客户端服务的核心设计包。hc::client 依赖于多个子模块，例如 hc::resource 用于收发管理客户端主机上的资源，hc::common 用于提供对游戏中的方块类进行操作的服务。在实现上，hc::resource 需要托管各种资源，并下发给硬件用于计算生成，对游戏的性能有重要的影响。

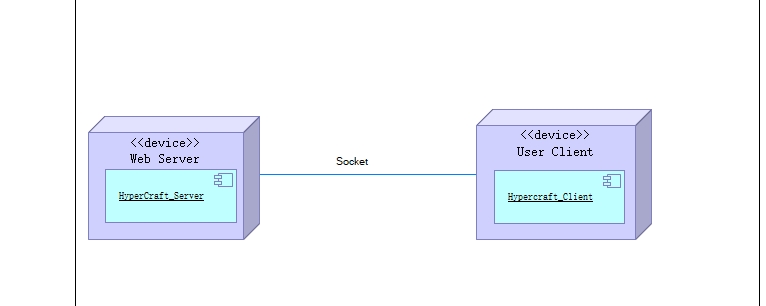
# 进程视图



HyperCraft 系统服务端和客户端节点都是单进程的，但是为了提高性能，我们广泛采用了多进程技术。例如在客户端，利用各个进程协同计算，充分利用多核 CPU 和 GPU 的性能。

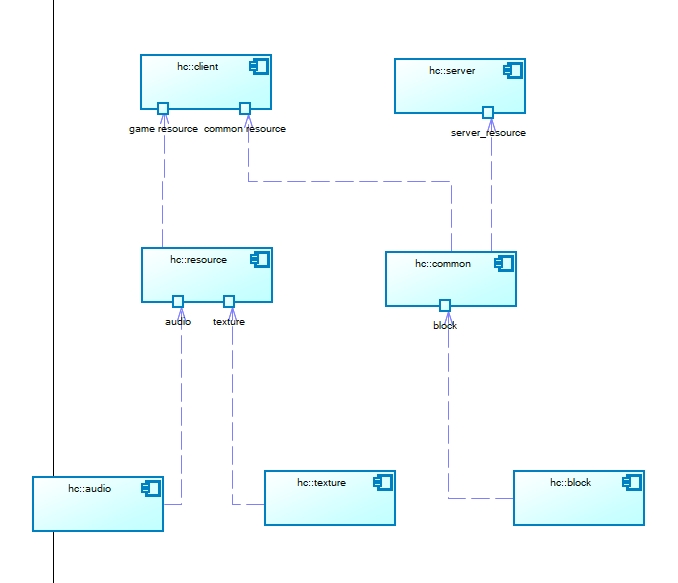
为了快速响应客户端请求，服务端也利用多进程技术进行了优化。服务端主线程对所有线程进行统一调度和管理。其中，监听线程（listen thread）用于监听对应端口，响应客户端请求。当接收到客户端请求时，主线程将开辟多个工作进程，对应不用的游戏逻辑，分别广播不同的资源。当响应某个客户端请求时，服务端进程还可以接收其他客户端的请求，实现了一定程度的并发。

# 部署视图



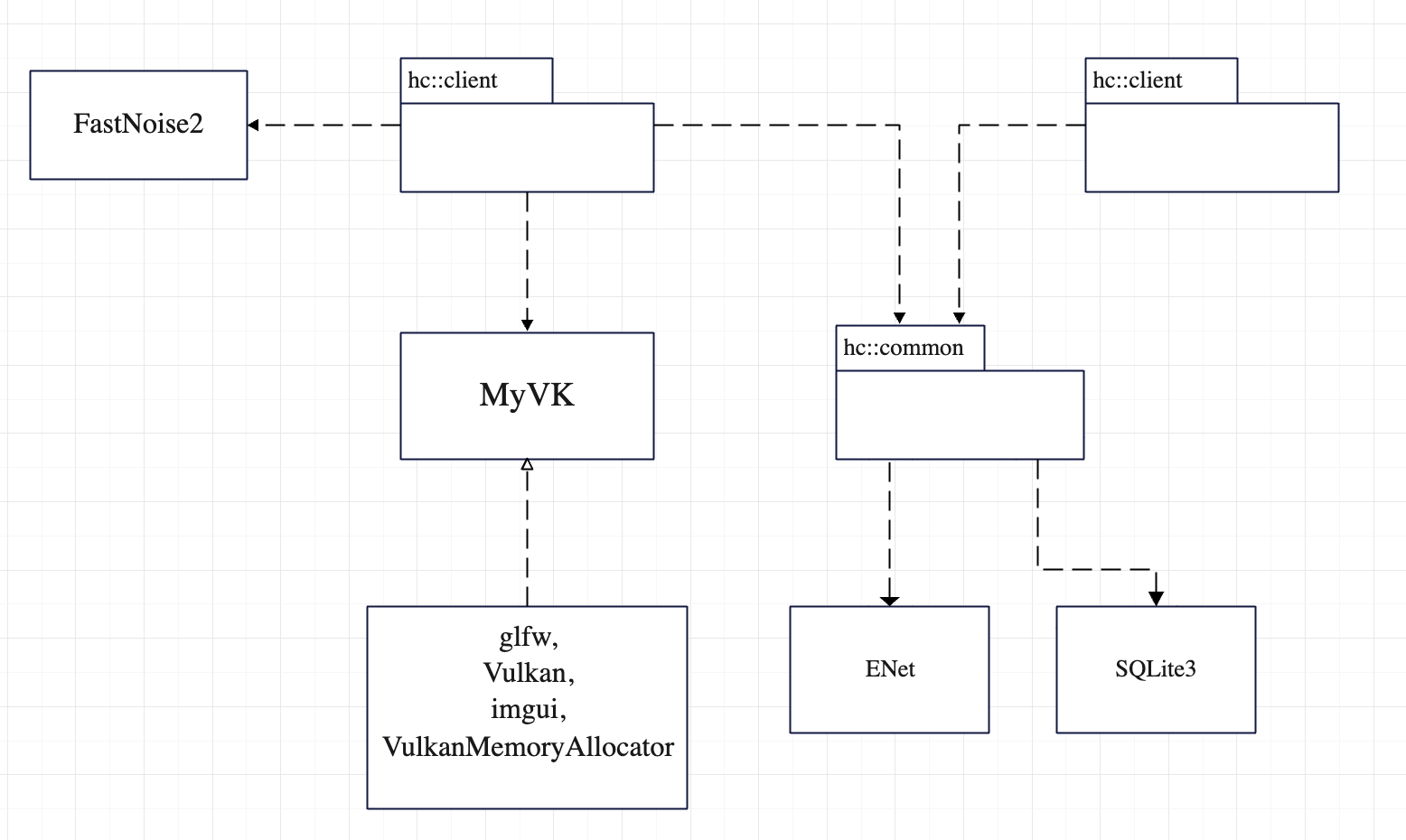
HyperCraft 系统的部署视图比较简洁清晰。系统运行时，客户端和服务端之间通过 TCP/IP 协议建立 socket 连接，两者进行点对点通信。客户端节点对应的是客户端的游戏进程，服务端节点对应的是服务端的资源广播进程。在常见配置下，客户端主机拥有支持高性能运算的图像处理单元（GPU），可以协同 CPU 进行快速的计算和渲染，显著提高游戏的性能。服务端目前只针对各个客户端进行资源的广播，所以对性能的要求不是很高。

# 实现视图



实现视图整体上与逻辑视图类似，核心设计包为 hc::client 和 hc::server。其中 hc::common 包含了系统中服务的重要实现，与 hc::client 和 hc::server 均有耦合。

# 技术视图



HyperCraft 采用的编程语言为 C++。得益于强大的面向对象功能和内存管理机制，C++ 被广泛应用于游戏、高性能网络开发。开发工具选用 CLion。实现上，hc::client 依赖于 FastNoise2 和 自主设计的 MyVK 库，MyVK 由 glfw, Vulkan 等依赖库实现。hc::common 依赖于 ENet 和 SQLite3，后者用于存储游戏世界、玩家状态等数据。

# 核心算法设计

## 体素贪心网格化算法

HyperCraft的体素网格化算法基于贪心的思想设计，能够合并同类的体素面，减少网格中三角形数量，从而减少网格的数据量，提升渲染效率。

## 方块更新算法

HyperCraft的多线程方块更新算法受到Posix Signal机制启发，能够减少任务队列中重复任务的数量，提升方块更新（包括光照更新和渲染网格更新）的效率。

# 质量属性的设计

HyperCraft 的软件架构强调高性能、可靠性和可扩展性。高性能利用进程视图中的多线程技术得以解决。在设计逻辑视图时，我们采用高重用性和组合性的 SOA 风格，减小了不同类之间的耦合程度，也使得代码易于阅读和理解。此外，为了促成可移植性，我们采用统一的编程规范和编译标准，运用 C++ 标准库实现算法。