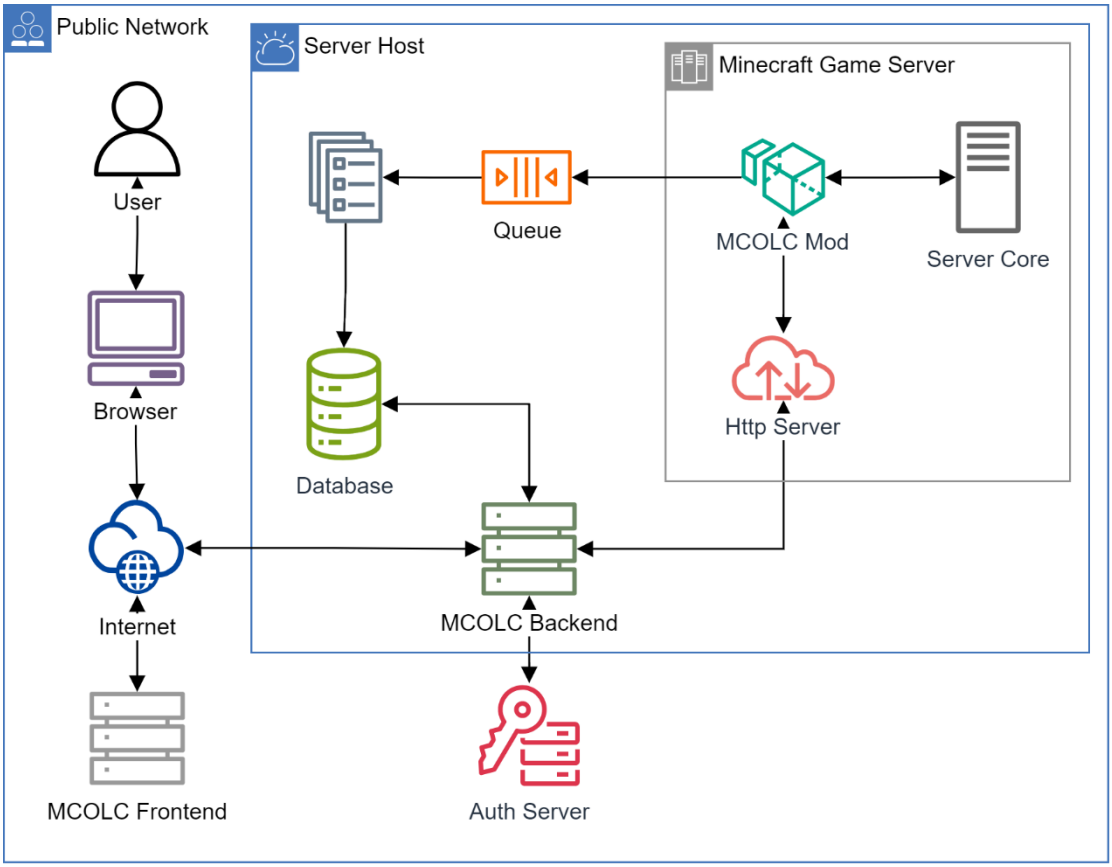
我对软件架构的理解

这学期在毕业前选了曹老师的《软件体系结构》课。本来想着大四可以轻松一点，一节课都不选，但是感觉在此之前接触的都是算法 AI 方面的知识，所以选择上曹老师这门偏软工设计的课程。

在上这门课之前，对软件架构是一概不知，认为就是设计模式。上了课之后，如果设计的软件是一个房屋，那么软件架构设计就是设计房屋间各个组件之间的关系来适应不同的应用场景。在机器学习课程上，有一个算法是“没有免费的午餐”定理（NFL），说是所有学习算法的期望性能都和随机胡猜是一样的。这是不是说明发明各种算法没有什么用，不是这样的，这个定理的假设是所有的问题都同等重要，但是现实生活中却不是。我觉得在软件设计领域也是一样，没有具体的问题就没有具体的软件架构。

下面就举几个具体的例子来说一下自己的理解，首先在学习《事件驱动架构》



这一课的时候，我学习到了有些用户产生的事件可以进行解耦，比如微信群里面的群发消息，用户发出消息后无需关心有谁看到，是谁响应。在这学期上的另一门课《网络应用开发》的时候，我们的项目就遇到了这样一个急需解决的问题：上面的项目是一个 MineCraft 在线社区平台，支持玩家在游戏外查看自己在各个服务器中的数据，包括装备，物品等。右上角的 MCOLC Mod 是 MineCraft 里面的一个模组，开启一个 Http Server 来等待其他服务器请求用户数据。MCOLC Backend 就需要不断发送请求给 Http Server 来请求用户的具体数据，如果用户下线，那么我们需要将用户的数据存储到中间相应的数据库，从而支持离线查看用户数据。

如果 MCOLC Mod 直接与 Database 对接，那么就处理不了下面的情况：如果有很多的人在这个时候访问用户数据，而此时 MCOLC Mod 还在向数据库里面存入东西，就更本应付不了这种高并发的场景。

针对这个具体的问题，我们才会设计出相应的软件架构。我们可以将存数据看作一个事件，我们设置一个消息中间件 RabbitMq，MCOLC Mod 数据存入之后就无需管这个数据怎么后续存入数据库，从而腾出手来继续处理其他用户的请求。可持久化组件可以异步地从队列中拿到消息，然后存入数据库就可以。

这证明了软件架构就是软件方面的算法，服务于目前的具体问题。

当然，在这门课上还学习到了许多其他有用的软件算法。如果我们想要追求可以轻松地水平向扩展，我们可以使用微服务架构。每一个具体地服务（商品服务，订单服务和运单服务）可以使用各自单独的一套独立架构来实现。我还记得老师上课说这样做的好处是每一个功能都可以有一个专门的团队来进行维护，节省人力。每一个服务可以用不同的数据库，不同的实现语言；但是缺点是复杂性

增加，网络通信开销和跨服务协调等问题。如果我们想要提高系统的响应效率，或者后端有一个处理时间十分长的应用，我们不想让用户等待很长的时间，我们就可以使用reactive 编程框架。记得曹老师上课举得一个例子：母亲叫你去洗澡，你可以先答应（首先响应用户的请求），但是不去，等到 10 分钟后不得不去的时候，再去洗也可以（后台异步处理，响应线程快速地反应。还学了许多技术，比如 springbatch 处理大批量地数据，restful 统一接口框架；MV 框架通过分离模型、视图和控制器，提高了代码的可维护性、可测试性和开发效率。

我们从中可以看到，没有任意一个软件架构是万能的，不同的问题需要不同的架构。

同时，软件架构也是一个问题抽象的过程。我记得比较清楚的是：曹老师在讲解响应式编程的时候，就结合时间驱动对现实问题进行了抽象。软件架构中有两种维度上面的分离，一种时间上的解耦，一种是物理上的解耦。我记得最清楚的是在时间上不解耦，在空间上解耦的事件：机场的广播，这个是有时间时限的，但是空间上是解耦的，没有指定某个人。

综上所述，软件架构就是对具体的现实问题进行抽象建模，然后解决这个问题的一种软件组织算法。

最后感谢曹老师一个学期教会了许多软件知识。