在线oj项目-微服务划分

为什么使用微服务架构

- **单体架构**
  - **可扩展性差**：随着业务的发展，单体应用的规模会不断增大，各个业务耦合度大，导致开发和维护变得困难。
  - **技术栈受限**：单体架构通常使用单一的技术栈，无法充分利用团队的技术专长和现有技术资源，也无法有针对性的选择技术栈。
  - **可靠性问题**：单体应用中一个模块的故障可能导致整个应用崩溃，可靠性较低。
- **集群架构**
  - **资源利用率低**：在集群架构中，资源分配和管理通常较为集中。这种集中式的资源管理方式可能导致资源分配不够灵活，无法满足各个服务或应用的实际需求。
  - **容错与隔离性不足**：各个业务耦合度大，组件之间的紧密耦合使得故障传播的可能性增加。
  - **部署与升级复杂**：在集群架构中，部署和升级通常需要协调多个组件或节点，这可能导致复杂性和风险的增加。
- **分布式架构**
  - **高耦合度**：分布式架构对于服务的拆分，往往粗粒度，这使得各个业务间的耦合问题并没有解决，限制了系统的灵活性和可扩展性。
  - **容错与隔离性不足：**由于耦合度大，由于组件之间的紧密依赖关系，故障的传播和放大效应也可能更加明显。
- **微服务架构的优势**

  而微服务和他们相比呢，进行了更细粒度的垂直拆分，所以便具有以下优势：
  - **易开发和维护：**每个微服务负责的业务比较清晰, 体量小, 开发和维护成本降低。
  - **容错性高：**一个服务发生故障, 可以使故障隔离在单个服务中, 不影响整体服务故障。
  - **扩展性好：**每个服务都是独立运行的, 我们可以结合项目实际情况进行扩展, 按需伸缩。
  - **技术选型灵活：**每个微服务都是单独的团队来运维, 可以根据业务特点和团队特点, 选择适合的技术栈。

微服务带来的挑战

- **服务依赖：**随着服务的数量增多, 服务之间的关系也会变得更加复杂。一个服务的更改, 需要考虑对其他服务的影响。

- **运维成本：** 一个业务流程会涉及多个微服务共同完成, 有更多的服务需要编译, 部署, 运行, 甚至可能是不同的编程语言, 不同的运行环境, 当然也需要集群来处理故障转移等。这对于运维人员而言, 挑战是巨大的。
- **开发和测试：** 一个业务流程可能涉及多个微服务共同完成, 服务调用引入网络延迟, 不可靠的网络, 如何进行容错处理等问题。这对开发和测试而言, 难度也会提升。
- **服务监控：** 在一个单体结构中, 很容易实现服务的监控，因为所有功能都在一个服务中。微服务架构下, 不仅需要对整个链路进行监控, 还需要对每一个服务实现监控。
- **负载均衡：** 微服务架构中的服务实例数量可能非常庞大，因此需要有效的服务发现和负载均衡机制来管理请求流量和保证高可用性。
- ...

如何划分微服务

微服务划分目标：

- **业务边界清晰**：
  - 每个微服务应该对应一块清晰的业务功能，有明确的责任和边界。
  - 服务间的依赖关系应该是单向的，避免循环依赖和复杂的调用关系。
- **最小化地变更**
  - 新增或者变更业务时有很明确的服务对应。要避免既可以在这个服务上实现，也可以在那个服务上实现的情况。
- **最大化的复用**
  - 在服务划分时，应考虑复用的场景，将具有共同功能或业务逻辑的服务抽象出来，形成可复用的组件
- **性能稳定简洁**
  - 服务的划分关注对性能的影响、是否稳定及架构是否简洁。
  - 引入某个中间件时也需要谨慎，引入额外的中间件势必会造成额外的维护成本。
- **积木式可拆解**
  - 微服务再划分好之后，随着版本的迭代，业务的累加。我们需要将某部分业务从某个微服务中拆出来，也要做到快速无感。

微服务划分原则：

- **单一职责原则：**
  - 每个微服务应该只负责一个特定的业务功能，有助于保持服务的聚焦和简单，便于独立开发和维护。

- **高内聚低耦合原则：**
  - 将联系紧密的功能聚合到一个微服务中，而将联系松散的功能拆分成不同的微服务，降低微服务之间的耦合度，提高系统的可维护性和可扩展性。
- **服务自治原则**
  - 每个微服务应该是独立的、可自治的，能够独立地开发、测试和部署，这有助于提高团队的协同开发效率和产品的交付速度。
- **单向依赖原则**
  - 微服务之间需要做到单向依赖, 严禁循环依赖, 双向依赖
  - 循环依赖: A -> B -> C ->A
  - 双向依赖: A -> B, B -> A
- **符合实际情况：**
  - 微服务拆分时，我们也需要对当前的实际情况进行合理的评估。
  - 如果某个业务功能的并发量较低，或与其他业务功能紧密耦合，那么将其拆分成独立的微服务可能并不必要。过度拆分可能导致每个微服务的资源利用率低下，增加不必要的运维和管理成本。
- **微服务划分步骤：**
  - 按照业务划分
  - 按照技术划分
  - 按实际情况划分

划分我们系统的服务

- **按照业务划分**

服务名称	具体功能
后台数据管理	题库管理、竞赛管理、C端用户管理、定时任务管理
用户服务	登录、注册、退出登录、个人中心、我的竞赛、我的消息
题库竞赛服务	题库列表、竞赛列表、竞赛报名、查看排名、竞赛或者刷题时题目切换、提交代码、运行代码、获取代码执行结果

- **按照技术划分**

服务名称	具体功能	拆分原因（不包含原有服务）

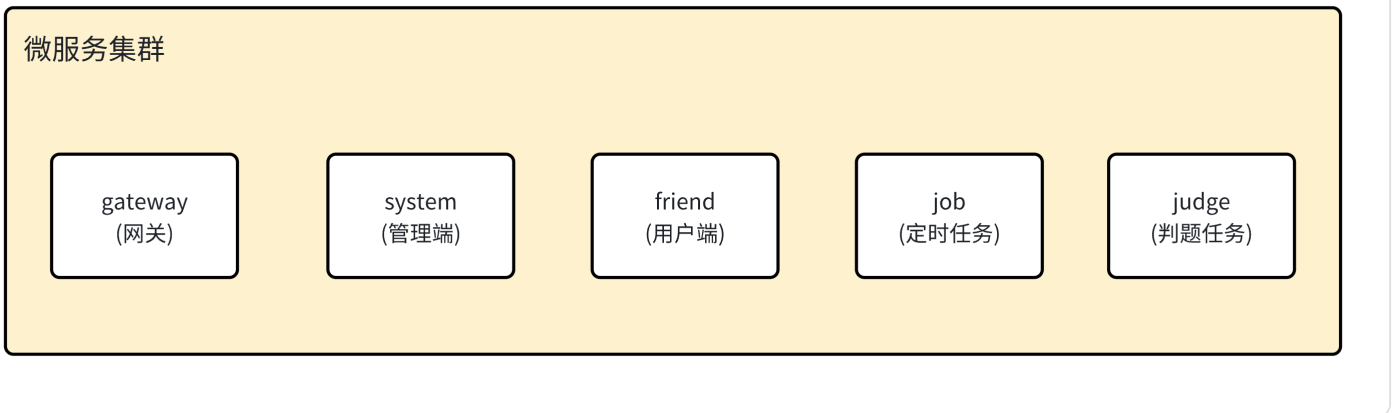
网关服务	统一的权限控制，统一的请求入口动态路由转发等	功能即原因
后台数据管理服务	题库管理、竞赛管理、C端用户管理、~~定时任务管理~~	
定时任务服务	定时任务增删改查、定时任务功能实现	从技术的角度分析定时任务的执行可能需要和多个服务配合完成。并且定时任务的执行可能会周期性或长时间占用资源。那么我们可以将定时任务管理拆分出来。
登录注册服务	登录、注册	可能需要与第三方的认证服务进行交互，存在性能瓶颈
用户服务	~~登录、注册、~~退出登录、个人中心、我的竞赛、我的消息	
消息服务	发送消息，接收消息	消息种类较多，短信、站内信、邮件等等。和多个第三方组件交互。
题库竞赛服务	题库列表、竞赛列表、竞赛报名、查看排名、~~竞赛或者刷题时题目切换、提交代码~~、~~运行代码~~、~~判题~~、~~获取代码执行结果~~、~~竞赛结果统计~~	
答题服务	竞赛或者刷题时题目切换、提交代码、获取代码执行结果	竞赛、刷题时高频使用功能。
判题服务	判题	判题逻辑可能比较复杂，不同类型的题目判题逻辑不一致，存在大量复杂计算。
代码沙箱服务	运行代码	与第三方组件进行交互，可能是性能瓶颈

- **按照实际情况划分**
  - 从课程角度出发：
    - 满足课程目标，精力集中在课程重点。
  - 从企业级项目研发角度出发：
    - 项目有一个逐渐完善的过程，大部分团队前期都是满足基本需求即可。
    - 后期版本迭代需求增加，或是用户量增加根据实际情况分析压力大的业务，灵活拆分新的微服务。

服务名称	**具体功能**	**合并原因**
网关服务		

	统一的权限控制，统一的请求入口动态路由转发等	
后台数据管理服务	题库管理、竞赛管理、C端用户管理	
定时任务服务	定时任务增删改查、定时任务功能实现（竞赛结果统计、发送消息）	消息发送我们仅实现系统消息，暂不是先实时通讯。掌握消息系统表结构设计、缓存设计等。 （消息服务构建可深可浅，如果要做的完整，可能需要专门做一个新的课程详细阐述。我也期待各位通过我们的项目学会做项目的方式方法，自主开展技术调研与设计）
用户端端服务	登录、注册、退出登录、个人中心、我的竞赛、我的消息 题库列表、竞赛列表、竞赛报名、查看排名、 竞赛或者刷题时题目切换、提交代码、获取代码执行结果	我们登录注册不需要与第三方的认证服务进行交互。身份认证通过token性能可以保障。 提供给用户的功能多为查询功能 （目前我们设计的提供给用户的功能，能满足在线OJ系统基本需求，也能满足大家学习的广度。）
判题+代码沙箱服务	判题、运行代码	我们目前题目只有编程题，并且仅支持Java编码。判题逻辑并不繁琐。性能瓶颈只是代码沙箱服务，我们需要借助docker。 我们的重心将放在代码沙箱实现，因为这是我们本次项目的核心。

- **最终我们敲定的服务架构如下：**



- **总结：**

在微服务划分的过程中，我们需从多角度深思熟虑。起初，服务应被细致划分，随后根据实际情况合理合并。合并后，编码时需预见未来版本迭代和需求的累积。例如，若判题模块承载过重业务，耗费大量资源，则应考虑将其拆分，力求以最少的代码变动实现平滑拆分。

在实际生产环境中，微服务的拆分通常经过团队充分讨论，最终确定一个临时合适的架构。由于技术和需求不断演变，架构亦需随时间调整。因此，技术的关键在于灵活适应变化，以最小改动满足当前需求，这标志着微服务划分的成功。