**1. 发布-订阅模式**

**业务场景：**

1. 会员和教练需要在活动或课程开始前收到提醒，确保他们不会错过重要的活动。

2. 会员可以根据兴趣、类别、地点等条件订阅活动。一旦有相关的事件发生，系统会自动向订阅者推送消息。

3. 公告发布、紧急通知

**详细示例：**

* **活动提醒：**
  + 系统设置定时任务，在活动开始前的特定时间（例如1天或1小时）自动发送提醒通知给所有订阅了该活动的会员和教练。

**技术实现：** 可以使用消息队列中间件（如RabbitMQ、Kafka）实现发布-订阅模式，确保消息的可靠性和系统的可扩展性。

**2. 主从数据库架构**

**性能提升：**

* + **读写分离：** 主数据库（Master）负责处理写操作（如会员注册、信息更新），从数据库（Slave）负责处理读操作（如查询会员信息、活动列表）。
  + **负载均衡：** 通过增加从数据库的数量，分散读取压力，提升系统的整体响应速度。

**数据备份与高可用性：**

* + **实时同步：** 从数据库实时复制主数据库的数据，确保数据的一致性。
  + **故障转移：** 当主数据库发生故障时，可以迅速切换到从数据库，避免服务中断。

**详细示例：**

* **查询优化：**
  + 会员A登录后查看自己的活动历史记录，从数据库直接提供查询结果，响应更快速。
* **故障应对：**
  + 主数据库意外宕机，从数据库立即提升为主数据库，系统继续正常运行，会员的数据和服务不受影响。

**3. 微服务架构**

**模块化服务：**

* + **会员服务：** 处理会员注册、登录、资料更新等功能。
  + **活动服务：** 管理活动的创建、更新、报名等功能。
  + **支付服务：** 处理会员的支付和退款（尽管支付功能我们使用外部接口，但未来可能会开发独立的支付系统）。
  + **通知服务：** 负责发送邮件、短信等通知。
  + **评论与评分服务：** 会员在活动后提供反馈和评分。
* **优势：**
  + **独立部署与升级：** 每个服务可以独立部署，更新某个服务不会影响其他服务的运行。
  + **技术栈多样性：** 不同的服务可以使用最适合的技术，提高开发效率。
  + **水平扩展：** 根据需要增加服务实例，处理高并发请求。

**详细示例：**

* **功能扩展：**
  + 想要增加event相关实体商品销售功能，可以新建一个电商服务，不需要修改现有的会员或活动服务。
  + 为了减少第三方支付接口调用开销，后期可能需要开发独立的支付业务，可以新建一个支付服务。
* **容错性：**
  + 如果通知服务出现问题，其他服务仍然正常运行，待问题解决后，通知服务可以重新上线，未发送的通知可以补发。

**技术：**

* **服务通信：** 使用API网关和轻量级通信协议（如RESTful API、gRPC）管理服务间的调用。
* **统一认证与授权：** 确保各个服务之间的安全性，实施统一的身份验证机制。
* **监控与日志：** 建立完善的监控系统，实时了解各个服务的运行状况，快速定位问题。

**4. MVC（模型-视图-控制器）架构**

**优化描述：**

* **架构分层：**
  + **模型（Model）：** 定义系统的数据结构和业务逻辑，如会员类、活动类、评论类等。
  + **视图（View）：** 负责呈现数据给用户，提供交互界面，如网页页面、移动端界面。
  + **控制器（Controller）：** 接收用户请求，调用模型处理数据，选择视图进行展示。

**详细示例：**

* **会员注册流程：**
  + **控制器**接收用户提交的注册信息，验证数据的完整性和合法性。
  + **模型**处理业务逻辑，将新会员的数据保存到数据库。
  + **视图**返回注册成功的页面，提示用户下一步操作。
* **活动报名流程：**
  + **控制器**接收会员的报名请求，检查活动的可用性。
  + **模型**更新报名信息，减少活动的剩余名额。
  + **视图**展示报名成功的确认页面，并提供活动详情。

在MVC架构的具体实现上，我使用

**前后端分离：**

* + **前端：** 使用现代前端框架（如React、Vue.js）构建视图层，提高用户体验。
  + **后端：** springboot构建后端，提供RESTful API，作为控制器和模型的实现，供前端调用。
* **代码可维护性：** 清晰的架构层次，方便团队协作和代码维护。