Entity、DTO、VO

简介

- **Entity**：与数据库表中的字段一一对应的实体类。

  它与数据库表一一对应，用于持久化数据。

- **DTO**：接收前端的传递的数据。

  DTO（Data Transfer Object，数据传输对象），通常是轻量级的，只包含需要传输的数据。

- **VO**：返回给前端的数据。

  VO（View Object，视图对象），用于在展示层显示数据，通常是将表示数据的实体对象中的一部分属性进行选择性的组合形成的一个新对象，目的是为了满足展示层数据要求的特定数据结构。

为什么要这么区分：

- **提高代码的可读性和可维护性**：每种对象都有其特定的用途和职责，使得代码结构更加清晰，开发人员能够更容易地理解数据的流动和使用方式。每个对象采取统一的命名，项目会变得更加一致和可预测，从而减少了出错的可能性。
- **解耦**：Entity、DTO、VO的划分降低了各个部分的耦合度。修改某一层的逻辑或数据时减少对于其它层的影响。
- **优化性能**：不同的对象，比如DTO和VO可以根据自身的功能和当前的需求，进行裁剪和拼装。合理的利用网络传输资源，提高性能。

除了这些Entity、DTO、VO还有：DO、BO、AO等等。这些对象的创建都是为了更好地组织和管理代码，提高系统的可维护性和可扩展性。然而，过度细分这些对象可能导致代码复杂性增加、维护成本上升、性能下降以及团队协作的挑战。因此，在使用这些对象时，我们需要综合考虑项目的实际需求和团队的能力，避免过度设计，确保在保持代码清晰和可维护的同时，不引入不必要的复杂性和开销。

划分过细的缺点：

- **增加复杂性**：过多的概念划分可能使开发者需要花费更多的时间去理解每个对象的作用和用途，增加了学习和理解的难度。这可能导致开发效率降低，甚至引入不必要的错误。
- **过度设计/影响性能**：过度设计意味着在实现简单功能时引入了过多的抽象和复杂性，这不仅增加了开发成本，过多的对象转换和数据处理可能会引入额外的性能开销。
- **维护成本上升**：随着对象划分的增多，系统中的对象数量也会增加，这可能导致代码库的膨胀。过多的对象意味着更多的类、接口和方法需要维护，从而增加了维护成本。

- **耦合度增加**：尽管对象划分的初衷是为了降低耦合度，但过于细致的划分有时反而可能导致耦合度上升。因为开发者可能会创建过多的中间层或转换层来处理不同对象之间的交互，这增加了系统各部分之间的依赖关系。