代码 Review 一般都会从代码风格、变量命名、语法统一处入手，当然这些应该更多的借助于 CI 等自动化手段来保证，但是在相关流程还不是很完善的前提下还是有必要进行关注。
此外代码可读性、代码健壮性、代码可扩展性都是 Review 时关注的点。每一个关注的点都依赖于 Reviewer 的实际经验，这里只简单举几个例子：
{ 代码可读性 }
代码是写给人看的，因为没有不需要维护的代码，无论是 Author 自己后续维护代码，还是他人接手代码，能快速地理解代码逻辑是非常必要的。
代码 Review 的时候可以关注以下几点：

- 变量、方法的命名是否合适，是否真实反映了他们的目的，这方面网上可以找到不少的资料说明。
- 实现的逻辑是否已有现成的库可以替代。如果有成熟的库可以使用，尽量不要自己去实现，因为可能会引入不必要的 bug。从我个人的角度，简洁（大白话就是代码少）是可读性一个很重要的指标。
- 关于注释。代码注释不求多，而在于有效，以前也经历过代码注释要求至少达到 30% 以上的年代，现在看来过多依赖注释其实是对代码质量的不自信，好的代码应该尽量做到自解释。在实际过程中偶尔能看到代码逻辑实际已经清晰明了，但是用语句不怎么通的英文注释了一通，最后反而是看懂了代码才能理解注释到底说了啥。因此 Review 的时候，不必要的注释可以建议去掉。
- 不好理解的地方要有恰当的注释。代码中如果有特殊处理、特殊的常量、或者不符合一般用法的逻辑需要特别注释说明一下。
- 代码的组织。良好的代码应该有较好的封装以及层级，使得代码看起来清晰明了，比如 DAO 层、Service 层。

{ 代码健壮性 }

- 代码的改动是否会影响其他功能。
- 用户参数是否做和细致的校验。
- 有没有 Panic 的可能（针对 Go 的说法）。
- 是否会破坏 API 的兼容性。

{ 可扩展性 }
当前的实现方式是否能兼容以后类似的扩展需求。比如在新增接口、API 或者调整参数以解决某一问题上，可以考虑是否后续会有其他类似情况发生。举几个例子：

- 假设我们需要定义一个 API 接口去获取一个用户的某些信息，例如联系方式等，我们定义的 API 就不能只返回这些信息，而是应该把用户相关的信息都返回。
- 假设我要定义一个参数，虽然当前定义成单个元素即可满足，例如 string, int，但是以后是否会涉及到多个元素的场景，是否定义成 []stirng, []int 是更优的。

这里只是举了有限的一些例子，在实际 Review 过程中，Reviewer 可以根据自身的经验从各个角度提出优化的意见。一般需要重点看看：

- 你看不懂或疑惑的地方。
- 打破你常规的地方。
- 复杂的地方。