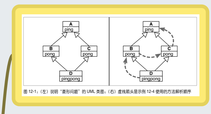
- 继承的优缺点
  - 子类化内置类型很麻烦
    - 内置类型（使用 C 语言编写）不会调用用户定义的类覆盖的特殊方法。
      - 原生类型的这种行为违背了面向对象编程的一个基本原则：始终应该从实例（self）所属的类开始搜索方法，即便在超类实现的类中调用也是如此。
      - __missing__ 方法（参见 3.4.2 节）却能按预期方式工作，不过这只是特例。
    - 不只实例内部的调用有这个问题（self.get() 不调用 self.__getitem__()），内置类型的方法调用的其他类的方法，如果被覆盖了，也不会被调用。
      - dict.update 方法会忽略 AnswerDict.__getitem__ 方法
      - 不要子类化内置类型，用户自己定义的类应该继承 collections 模块（http://docs.python.org/3/library/collections.html）中的类，例如 UserDict、UserList 和 UserString，这些类做了特殊设计，因此易于扩展。
  - 多重继承和方法解析顺序
    - 
    - 方法解析顺序（Method Resolution Order，MRO）
      - 类都有一个名为 __mro__ 的属性，它的值是一个元组，按照方法解析顺序列出各个超类，从当前类一直向上，直到object 类。
      - 若想把方法调用委托给超类，推荐的方式是使用内置的 super() 函数。
        - 使用 super() 最安全，也不易过时。
        - 调用框架或不受自己控制的类层次结构中的方法时，尤其适合使用 super()。
        - 使用 super() 调用方法时，会遵守方法解析顺序
      - 有时可能需要绕过方法解析顺序，直接调用某个超类的方法——这样做有时更方便。
        - 注意，直接在类上调用实例方法时，必须显式传入 self 参数，因为这样访问的是未绑定方法（unbound method）。
      - 方法解析顺序不仅考虑继承图，还考虑子类声明中列出超类的顺序。
        - 把 D 类声明为 class D(C, B):，那么 D 类的 __mro__ 属性就会不一样：先搜索 C 类，再搜索 B 类。
      - 方法解析顺序使用 C3 算法计算。
        - Michele Simionato 的论文“The Python 2.3 Method Resolution Order”（https://www.python.org/download/releases/2.3/mro/）对 Python 方法解析顺序使用的 C3 算法做了权威论述。
        - 除非大量使用多重继承，或者继承关系不同寻常，否则不用了解 C3 算法，因此也不用阅读这篇论文。
  - 处理多重继承
    - 继承有很多用途，而多重继承增加了可选方案和复杂度。
    - 是避免把类图搅乱的一些建议
      - 1. 把接口继承和实现继承区分开
        - 使用多重继承时，一定要明确一开始为什么创建子类。
        - 主要原因
          - 继承接口，创建子类型，实现“是什么”关系
          - 继承实现，通过重用避免代码重复
        - 其实这两条经常同时出现，不过只要可能，一定要明确意图。通过继承重用代码是实现细节，通常可以换用组合和委托模式。而接口继承则是框架的支柱。
      - 2. 使用抽象基类显式表示接口
        - 现代的 Python 中，如果类的作用是定义接口，应该明确把它定义为抽象基类。
      - 3. 通过混入重用代码
        - 如果一个类的作用是为多个不相关的子类提供方法实现，从而实现重用，但不体现“是什么”关系，应该把那个类明确地定义为混入类（mixin class）。
        - 从概念上讲，混入不定义新类型，只是打包方法，便于重用。混入类绝对不能实例化，而且具体类不能只继承混入类。
        - 混入类应该提供某方面的特定行为，只实现少量关系非常紧密的方法。
      - 4. 在名称中明确指明混入
        - 因为在 Python 中没有把类声明为混入的正规方式，所以强烈推荐在名称中加入 ...Mixin 后缀。
      - 5. 抽象基类可以作为混入，反过来则不成立
        - 抽象基类可以实现具体方法，因此也可以作为混入使用。
        - 不过，抽象基类会定义类型，而混入做不到。
        - 此外，抽象基类可以作为其他类的唯一基类，而混入决不能作为唯一的超类，除非继承另一个更具体的混入——真实的代码很少这样做。
        - 抽象基类有个局限是混入没有的：抽象基类中实现的具体方法只能与抽象基类及其超类中的方法协作。
          - 这表明，抽象基类中的具体方法只是一种便利措施，因为这些方法所做的一切，用户调用抽象基类中的其他方法也能做到。
      - 6. 不要子类化多个具体类
        - 具体类可以没有，或最多只有一个具体超类。
        - 也就是说，具体类的超类中除了这一个具体超类之外，其余的都是抽象基类或混入。
      - 7. 为用户提供聚合类
        - 如果抽象基类或混入的组合对客户代码非常有用，那就提供一个类，使用易于理解的方式把它们结合起来。
      - 8. “优先使用对象组合，而不是类继承”
        - 优先使用组合能让设计更灵活。
        - 组合和委托可以代替混入，把行为提供给不同的类，但是不能取代接口继承去定义类型层次结构。