

正交设计

CAC@OPPO by 黄俊彬 & 覃宇

软件设计是为了什么？

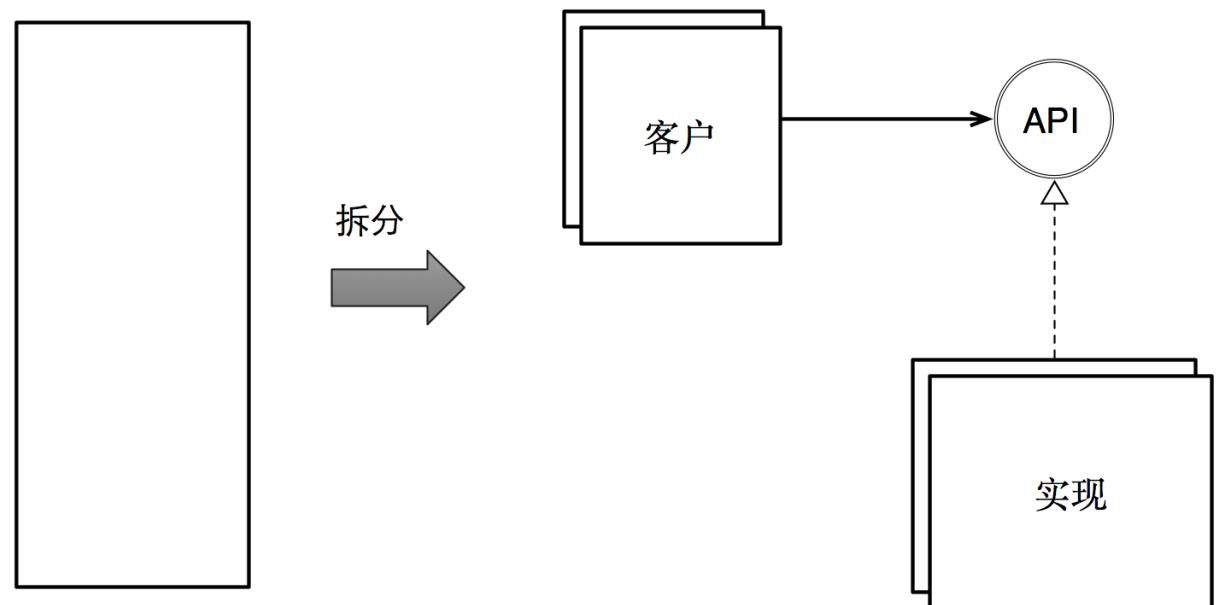
软件设计最重要的目的是实现功能。随着时间推移，不可提前预知的不确定性导致软件的复杂度不断增加，超出了开发者的认知极限，功能实现越来越难。软件设计是解决这个问题的重要手段。

软件设计是为了让软件在长期范围内容易应对变化。——*Kent Beck*

软件设计如何解决复杂性？如何长期应对变化？

模块化，分而治之

- 降低认知难度
- 将影响局部化
- 带来模块复用

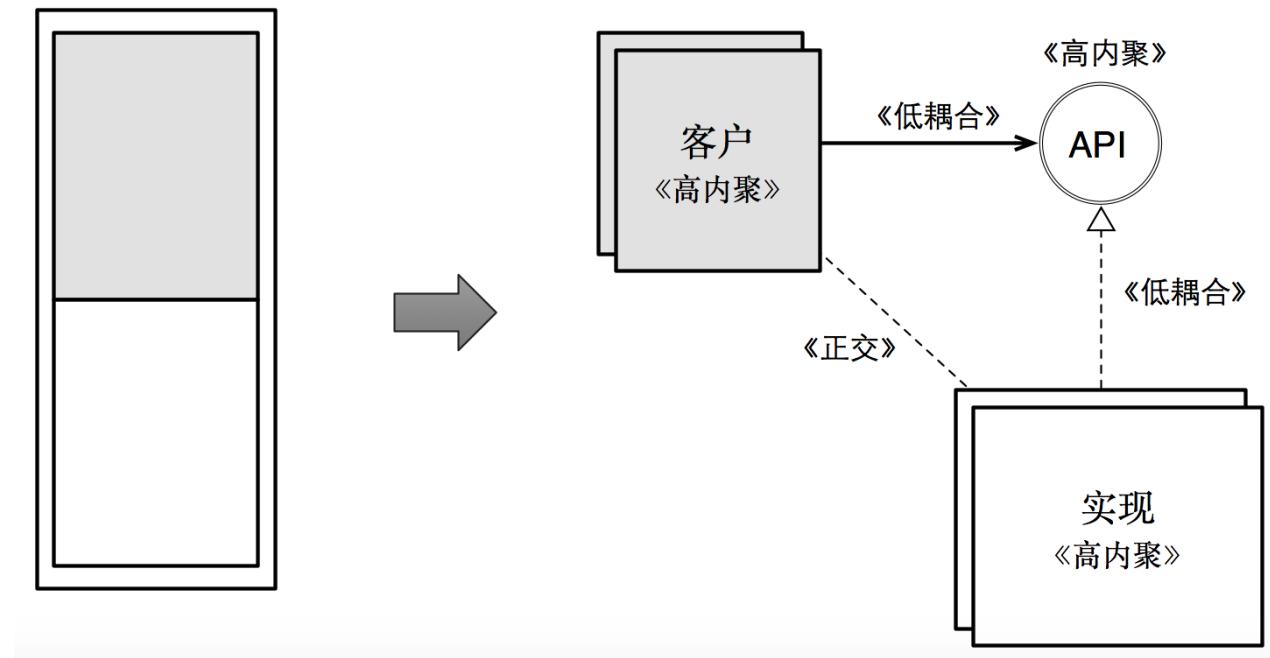


软件模块怎么分，又怎么合？

高内聚，低耦合

软件设计的最基本原则

- 只有**关联(?)**紧密的事物才能并应该被放在一起
- 软件模块之间尽可能不要相互影响(?)
- 一方的变化不会影响另外一方的变化（正交）



你还能想到哪些软件设计原则？

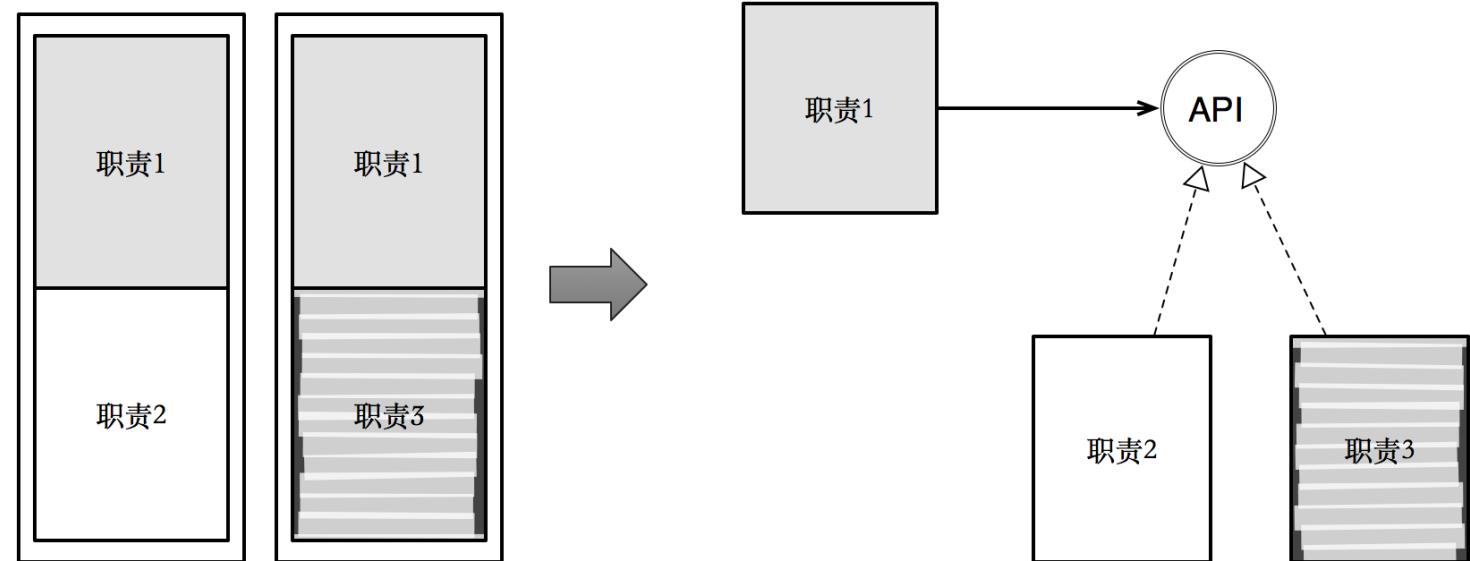
S. O. L. I. D 原则

- S 单一职责原则
- O 开闭原则
- L 里氏替换原则
- I 接口隔离原则
- D 依赖倒置原则
- | 一个类只应该有一个变化原因（职责）。——Robert. C. Martin

如何在编码开发时贯彻这些原则？

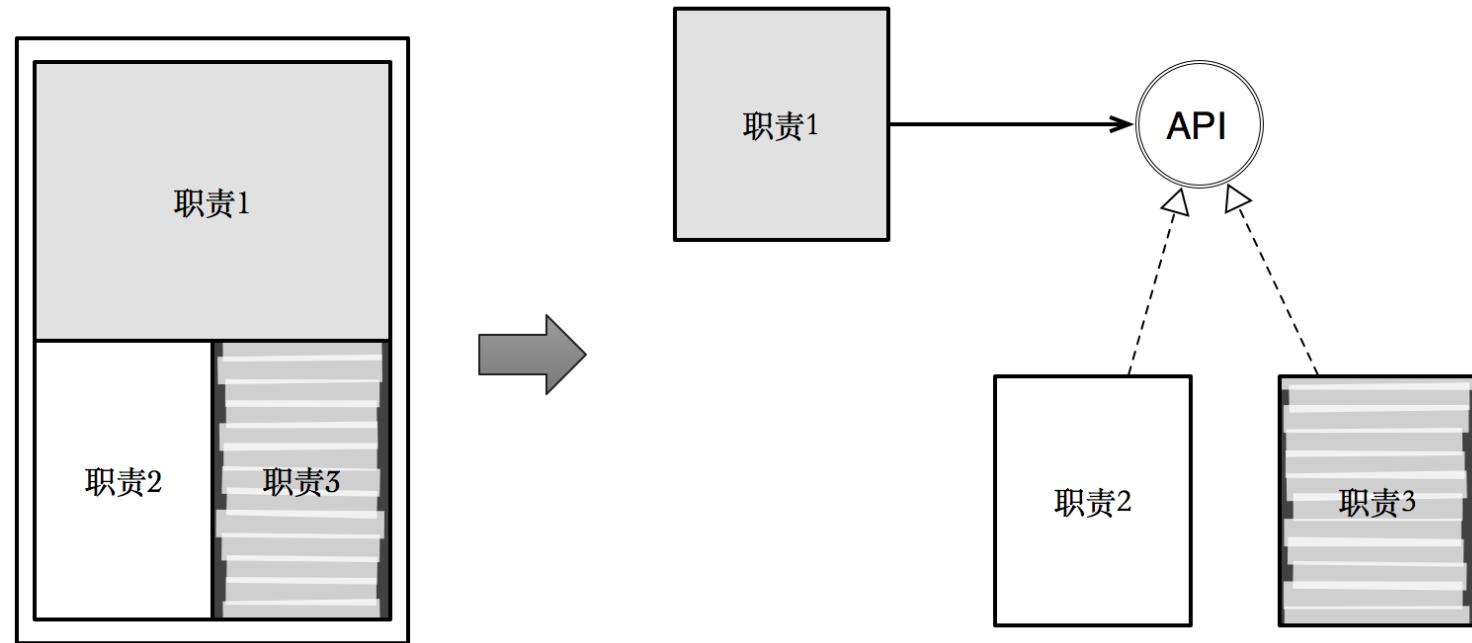
避免一个变化 导致多处修改

遵循单一职责原则，
消除重复



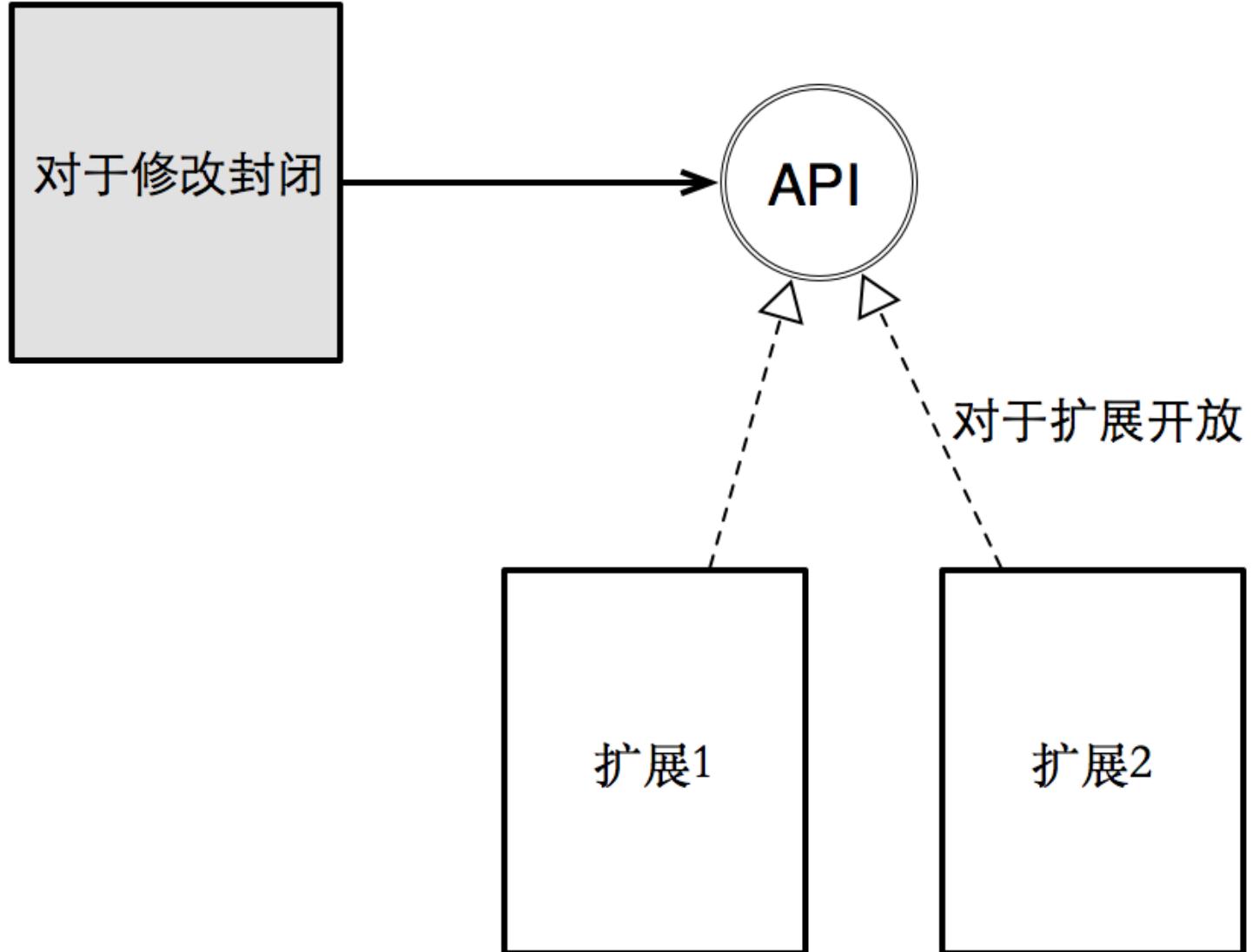
避免多个变化 导致一处修改

遵循单一职责原则，
拆分模块，分离不同
变化方向



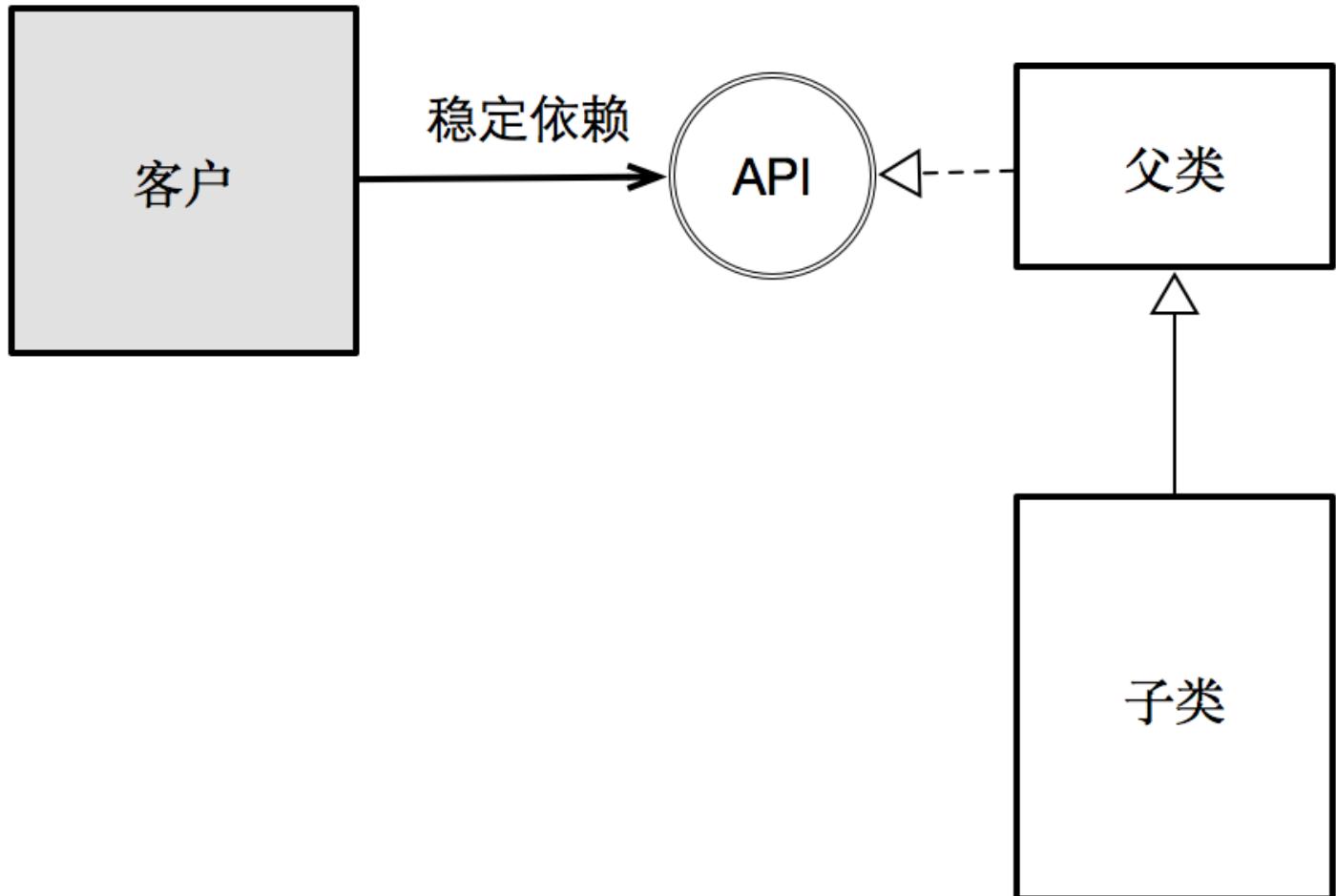
避免一个变化 导致多处修改

遵循开闭原则，提取
可以扩展的接口，分
离不同变化方向



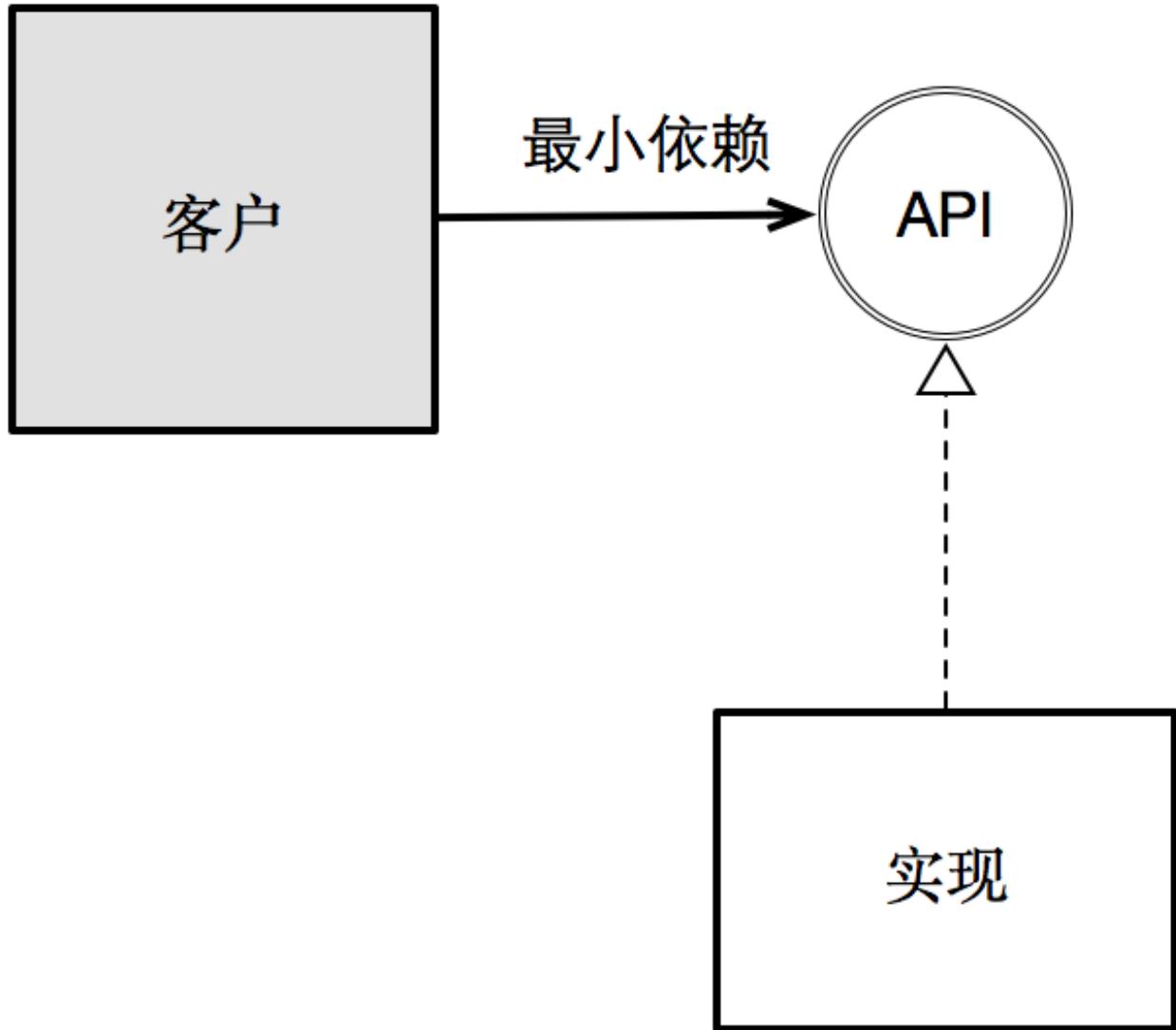
不依赖不稳定的 的依赖

遵循里氏替换原则，
子类实现不能破坏父
类被依赖的稳定接
口，向稳定的方向依
赖



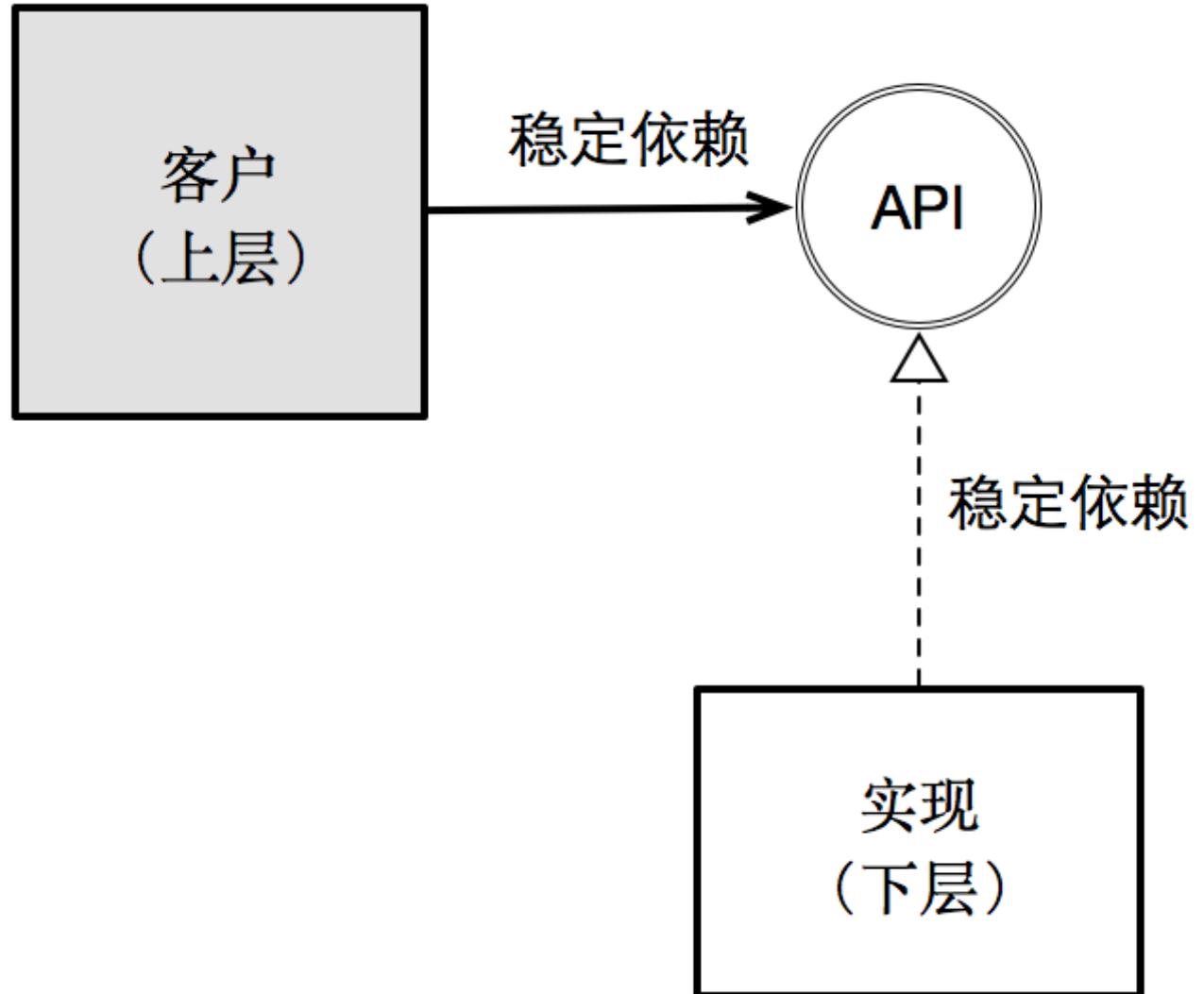
不依赖不必要的 的依赖

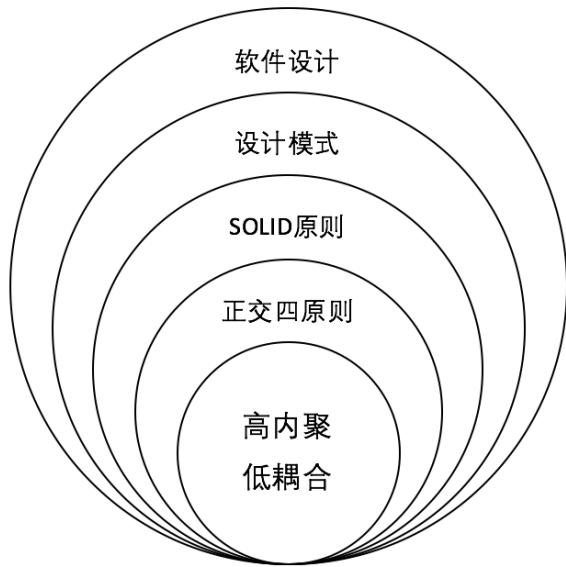
遵循接口隔离原则，
拆分接口，缩小依赖
的范围



不依赖不稳定的 的依赖

循依赖倒置原则，
从“上层”定义稳定接
口，“下层”实现向稳
定的方向依赖





正交设计四原则

- 1 消除重复（避免一个变化导致多处修改）
- 2 分离不同变化方向（避免多个变化导致一处修改）
- 3 缩小依赖范围（不依赖不必要的依赖）
- 4 向着稳定的方向依赖（不依赖不稳定的依赖）

案例展示

X项目软件设计演化之旅



需求一：在聊天界面支持选择本地图片进行发送 

实现

```
class ChooseLocalImageActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_choose_image)
        showImageList()
    }

    private fun showImageList() {
        //省略异步回调 . . .
        var imageList = getLocalImageList()
        var listAdapter = ListAdapter(imageList)
        lvList.adapter = listAdapter
    }

    private fun getLocalImageList():List<Image>{
        var imageList= mutableListOf<Image>()
        //从本地读出图片列表
        //. . .
        return imageList
    }
}
```

需求二：在聊天界面支持选择网盘图片进行发送



快速实现

```
class ChooseRemoteImageActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_remote_image)
        showImageList()
    }

    private fun showImageList() {
        var imageList = getRemoteImageList()
        var listAdapter = ListAdapter(imageList)
        lvList.adapter= listAdapter
    }

    private fun getRemoteImageList():List<Image>{
        //省略异步回调 ...
        var imageList = mutableListOf<Image>()
        //从通过API读出图片列表
        //...
        return imageList
    }
}
```

讨论：有没有同学遇到这种情况，你是如何解决？

重复设计 

Copy-Paste是最快的实现方法，但会产生「重复设计」

散弹式修改

需求：页面除了支持列表布局，还需要支持九宫格布局

1 消除重复

| 避免一个变化导致多处修改

实现

```
class ChooseImageActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_choose_image)
        showImageList()
    }

    private fun showImageList() {
        var imageList= listOf<Image>()
        //省略异步回调 ...
        if (isLocal) {
            imageList = getLocalImageList()
        } else {
            imageList = getRemoteImageList()
        }
        var listAdapter = ListAdapter(imageList)
        lvList.adapter = listAdapter
    }

    fun getLocalImageList():List<Image>{
        var imageList= mutableListOf<Image>()
        //从本地读出图片列表
        //...
        return imageList
    }

    fun getRemoteImageList():List<Image>{
        var imageList= mutableListOf<Image>()
        //从通过API读出图片列表
        //...
        return imageList
    }
}
```

问题 ?

- 开放封闭原则，对扩展是开放的，而对修改是封闭的。

2 分离不同变化

| 避免多个变化导致一处修改

思考



- 1、显示部分不会经常变
- 2、获取数据源的方式会不断扩展

分离(单一职责原则)

```
interface ImageDataSource{
    fun getImageList():List<Image>
}

class LocalImageSource :ImageDataSource{
    override fun getImageList(): List<Image> {
        var imageList= mutableListOf<Image>()
        //从本地读出图片列表
        //....
        return imageList
    }
}

class RemoteImageSource :ImageDataSource{
    override fun getImageList(): List<Image> {
        var imageList= mutableListOf<Image>()
        //从通过API读出图片列表
        //....
        return imageList
    }
}
```

3 缩小依赖范围

| 不依赖不必要的依赖

抽象(依赖倒置原则)

```
class ChooseImageActivity : AppCompatActivity() {  
    var imageDataSource:ImageDataSource? = null  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_choose_image)  
        showImageList()  
    }  
  
    private fun showImageList() {  
        //省略异步回调 ... ...  
        var imageList = imageDataSource?.getImageList()  
        imageList?.let {  
            var listAdapter= ListAdapter(imageList)  
            lvList.adapter= listAdapter  
        }  
    }  
}
```

需求三：在聊天界面支持选择本地及网盘文件进行发送 

类型重复

按照既有的代码结构，可以通过Copy Paste快速地实现这个功能

```
interface FileDataSource{
    fun getFileList():List<FileInfo>
}

class LocalFileSource :FileDataSource{
    override fun getFileList(): List<FileInfo> {
        var fileList= mutableListOf<FileInfo>()
        //从本地读出文件列表
        //...
        return fileList
    }
}

class RemoteFileSource :FileDataSource{
    override fun getFileList(): List<FileInfo> {
        var fileList= mutableListOf<FileInfo>()
        //从通过API读出文件片列表
        //...
        return fileList
    }
}
```

类型参数化

```
data class BaseFileInfo(val id:String, val name:String, val path:String, val size:String)

interface FileDataSource<T:BaseFileInfo>{
    fun getFileList():List<T>
}
```

实现

```
class ChooseFileActivity: AppCompatActivity() {  
  
    var fileDataSource: FileDataSource<BaseFileInfo>?=null  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_choose_file)  
    }  
  
    private fun showFileList() {  
        var imageList = fileDataSource?.getFileList()  
        imageList?.let {  
            var listAdapter = ListAdapter(imageList)  
            lvList.adapter = listAdapter  
        }  
    }  
}
```

| 需求四：选择页面需要支持按时间或大小进行排序 

接口修改

增加排序字段及排序类型

```
interface FileDataSource<T:BaseFileInfo>{
    fun getFileList(orderKey:String,orderDesc:Boolean):List<T>
}
```



需求五：选择页面需要同时支持排序规则及按文件类型过滤  & 

4 向着稳定的方向依赖

| 不依赖不稳定的依赖

条件动态配置

```
class Condition{
    //字段
    val field:String = ""
    //类型
    val type:OptType = OptType.NULL
    //值
    val value:String = ""
}

interface FileDataSource<T:BaseFileInfo> {
    fun getFileList(conditions: List<Condition>):List<T>
}
```

? 结束



需求六：除了聊天，OA、考勤等其他模块也需要支持选择文件功能



讨论：支持多模块调用，你会怎样进行设计？

建造者模式

```
ChooseFileDialogue.Builder(this)
    .setCondition(mutableListOf())
    .setDataSource(RemoteFileSource())
    .onCreate()
```

一切围绕着变化

由变化驱动，反过来让系统演进的更容易应对变化