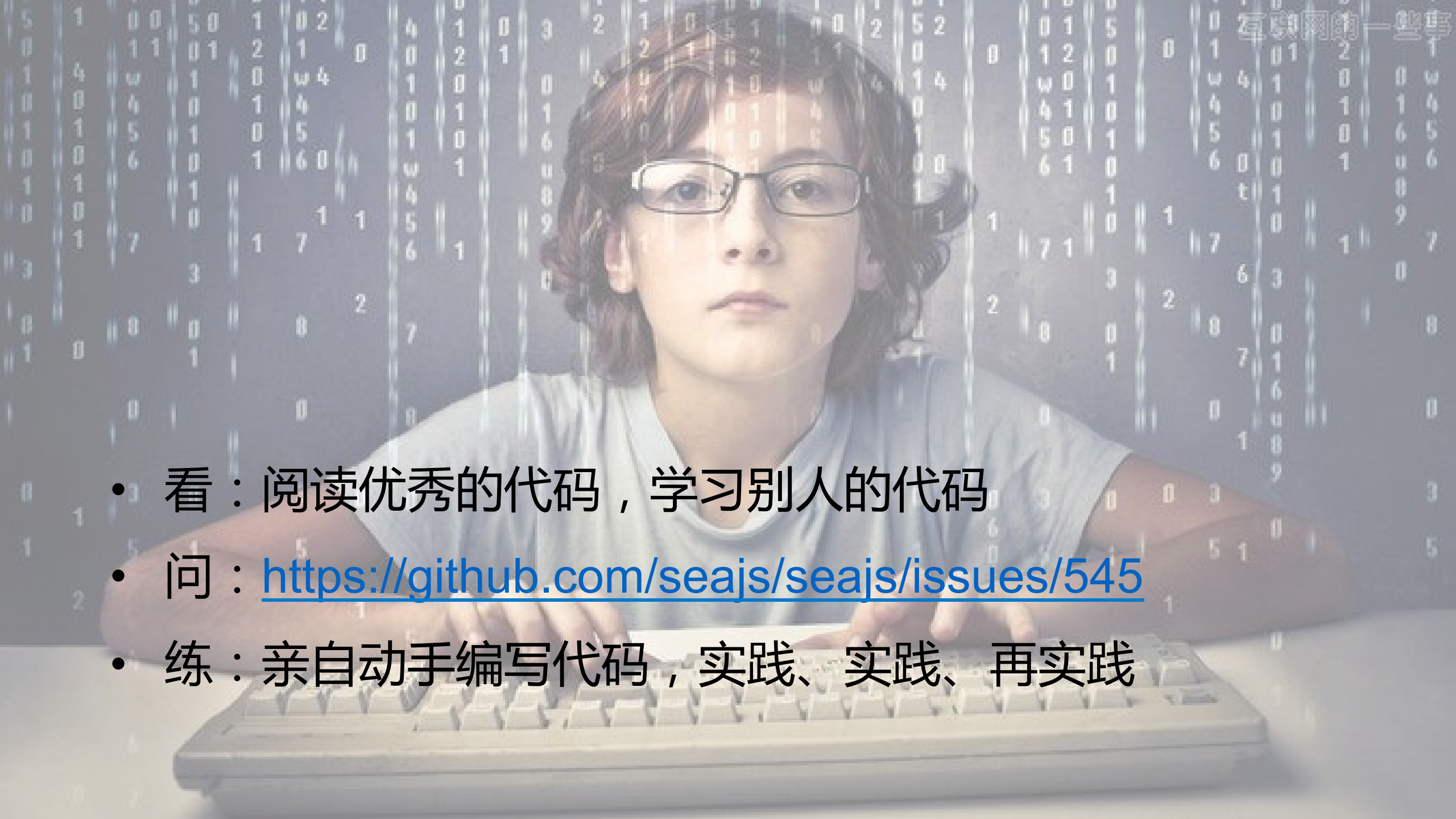




良好的编程实践

清华大学软件学院 刘强、陈华榕



- 
- 看：阅读优秀的代码，学习别人的代码
 - 问：<https://github.com/seajs/seajs/issues/545>
 - 练：亲自动手编写代码，实践、实践、再实践

软件开发的工程思维

高楼大厦可能是这样建成的

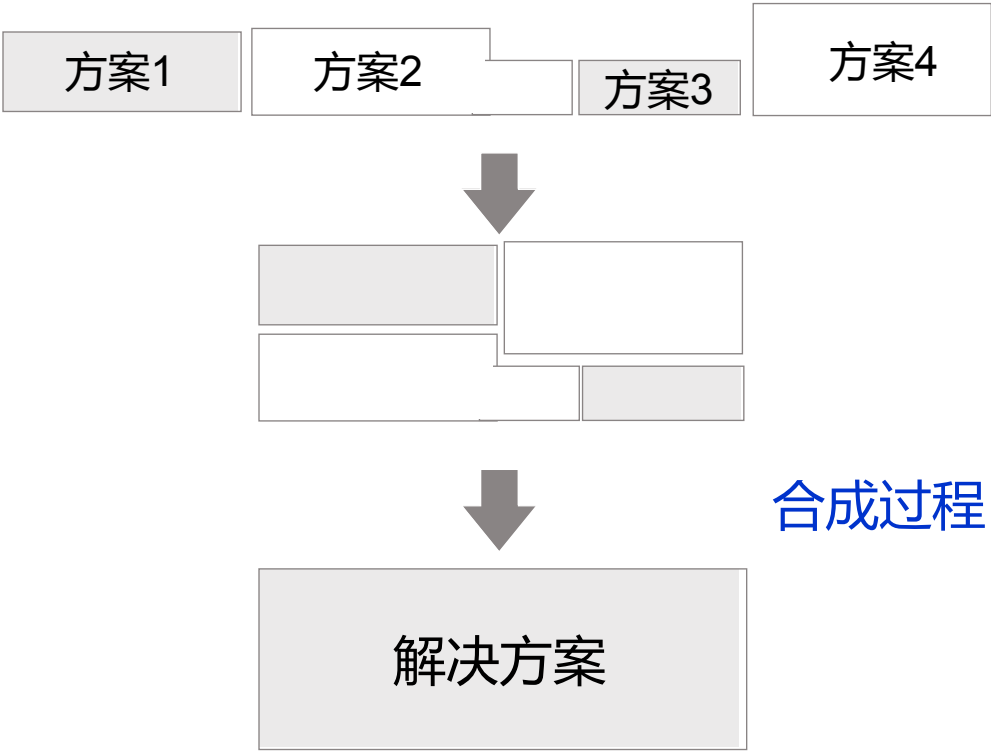
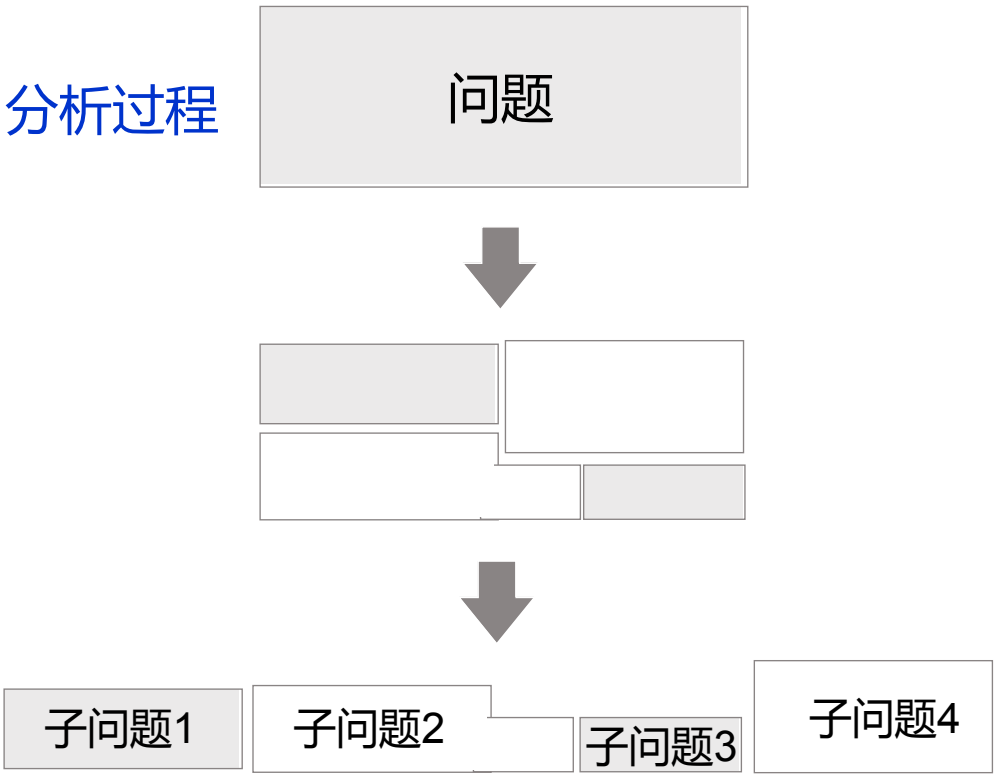
1. 分析问题
2. 初步设计
3. 模型评估与测试
4. 搭建框架
5. 建造
6. 测试与验收



软件开发的工程思维



分析过程



合成过程

软件开发的工程思维



高质量软件
开发之道



高质量的
设计

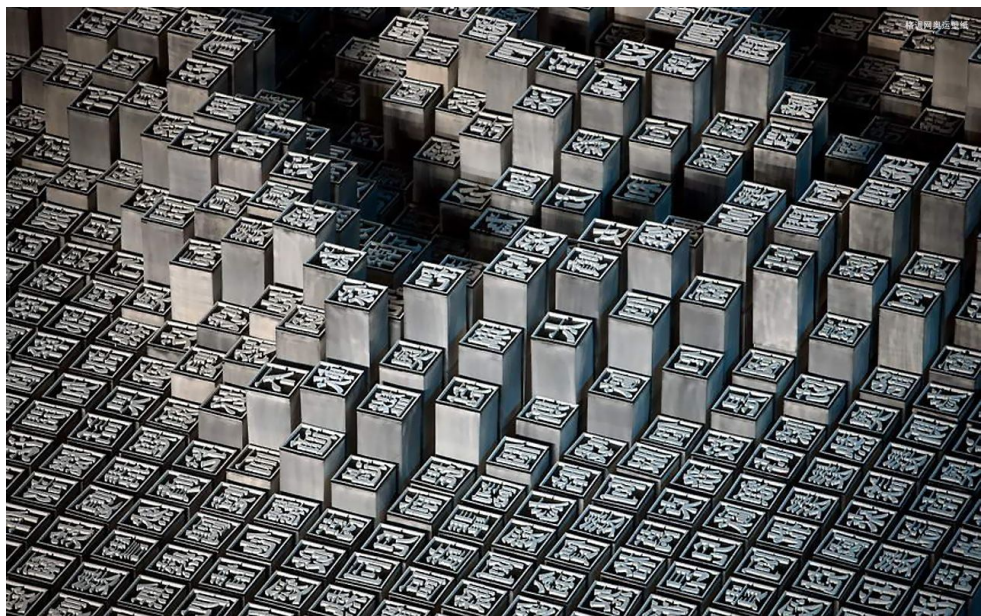
- 模块化设计
- 面向抽象编程
- 错误与异常处理

规范的
编码

有效的
测试

模块化设计

活字印刷术可以称得上是古代产品领域的模块化设计



- 每个汉字代表一个文字“模块”，具有特定的功能和含义
- 语法是连接汉字模块的“接口”，通过它构成了最终的文章“产品”

模块化设计

集装箱酒店

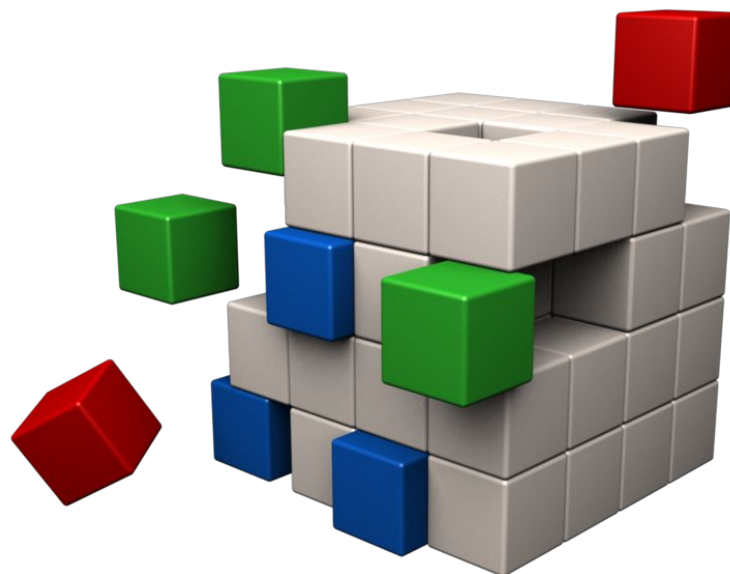
迪拜集团在伦敦西部建立的
第一家模块化酒店



模块化程序设计

基本思想： 将一个大的程序按功能分拆成一系列小模块

- 降低程序设计的复杂性
- 提高模块的可靠性和复用性
- 缩短产品的开发周期
- 易于维护和功能扩展

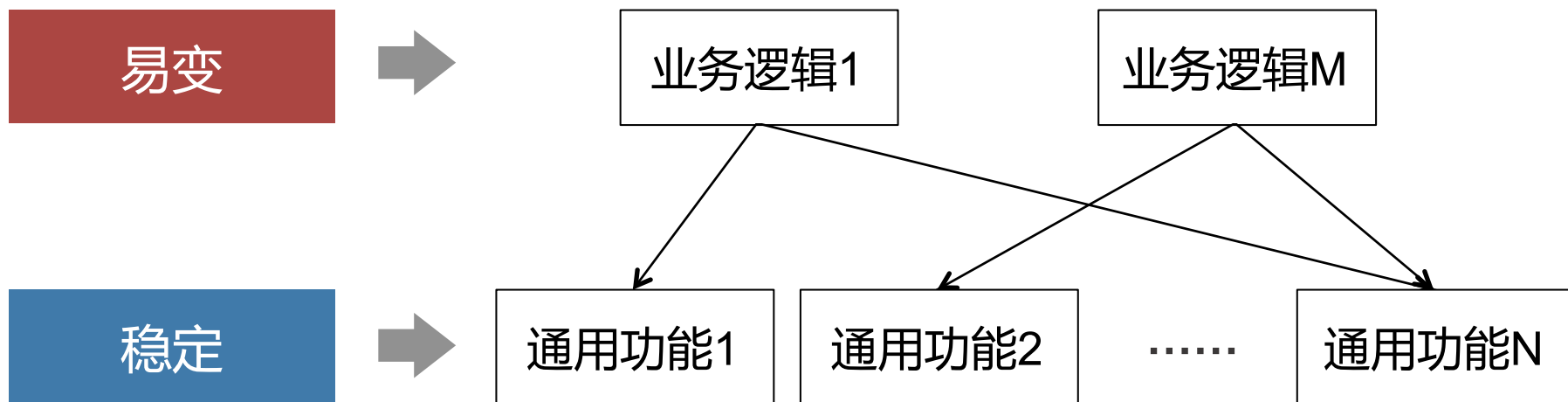


模块化程序设计



模块化程序设计

基于易变与稳定：认识和识别变与不变的部分，并将之科学地分离开。



模块化程序设计

基于单一职责：类或者函数应该只做一件事，并且做好这件事。



- 单一职责 \neq 单一功能
- 单一职责：只有一个引起变化的原因

Python的模块化设计



函数

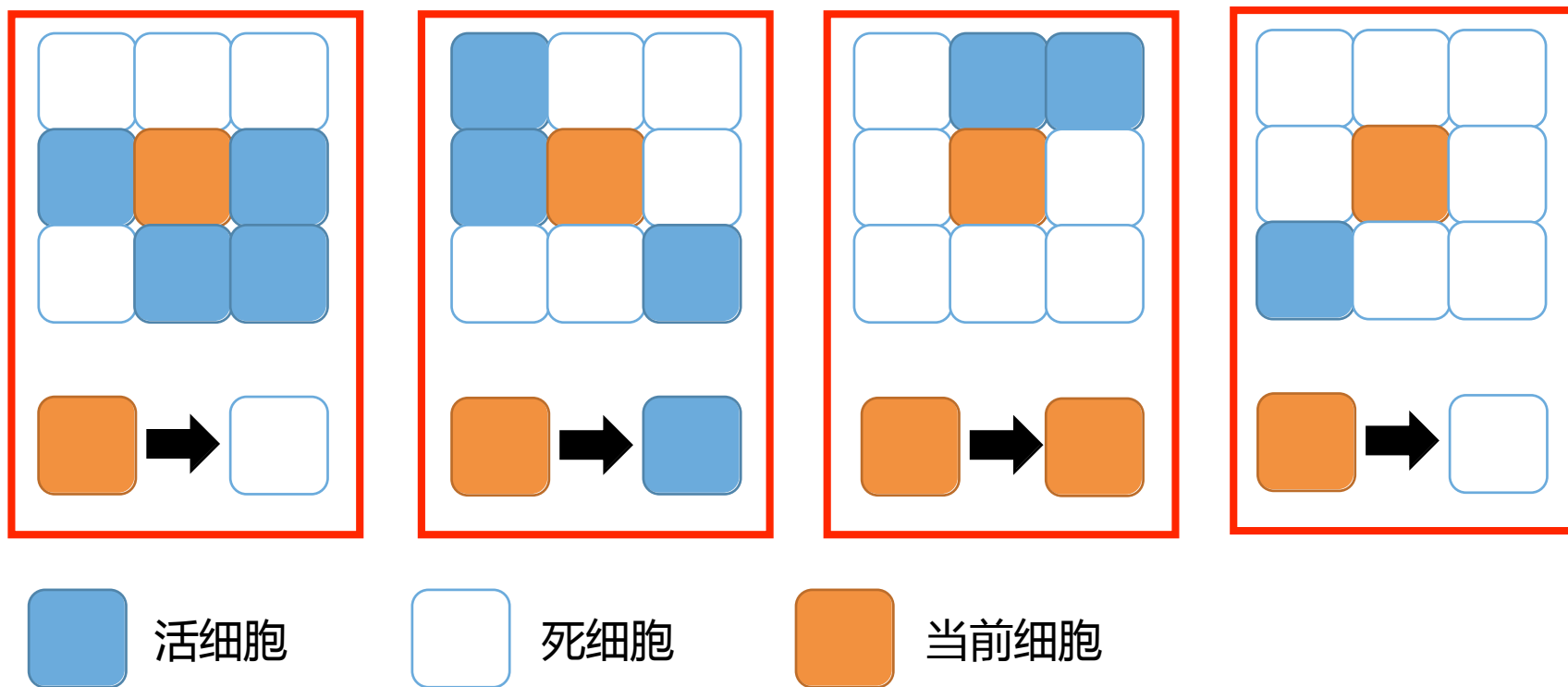
类

模块

包

案例：生命游戏

生命游戏是英国数学家约翰·何顿·康威在1970年发明的细胞自动机。



案例：生命游戏

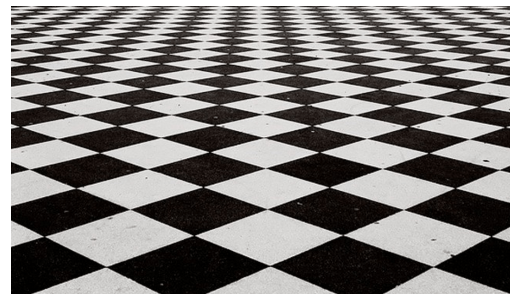
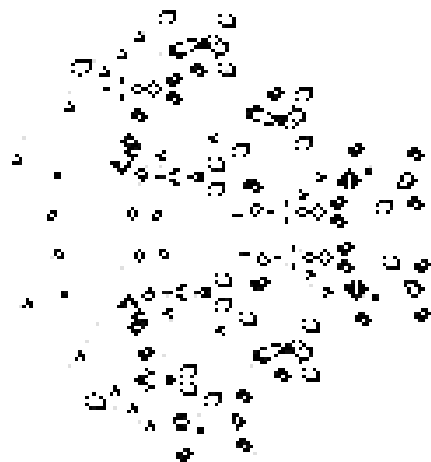


动态图来自维基百科

<https://zh.wikipedia.org/wiki/%E5%BA%B7%E5%A8%81%E7%94%9F%E5%91%BD%E6%B8%B8%E6%88%8F>

案例：生命游戏的模块划分

想一想，生命游戏有哪些要素？



地图

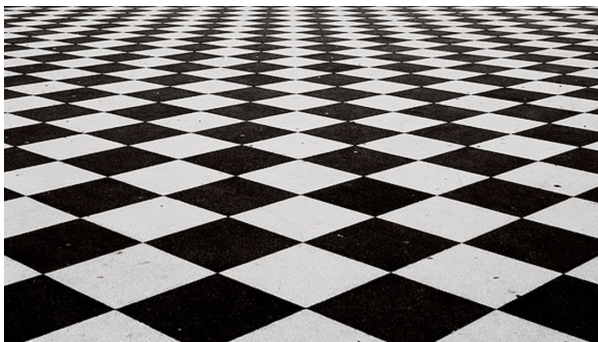


逻辑



计时

案例：生命游戏的模块划分



地图

管理与地图相关一切数据的初始化、获取、更新等



逻辑

控制完整游戏逻辑，根据地图数据依照逻辑进行相应更新



计时

负责时间相关的功能，在适当的时机触发游戏逻辑模块对地图的更新

模块化设计的好处：**思路清晰，易于测试，适应未来可能的变化**

案例：生命游戏的模块划分



实际上，模块化设计可以有多种不同的方案，应该选择利于理清思路、方便测试、容易调整的方案，同时避免“过度设计”。



地图



逻辑



计时



UI

例如：如果你愿意的话，加一个UI模块专门负责将游戏状态呈现给用户也是可以的。

面向抽象编程



在模块化设计的基础上，我们可以先设计出各个模块的“骨架”，或者说对各个模块进行“抽象”，定义它们之间的“接口”。

定义各个模块互相关联的部分，这些部分在未来开发中不应该发生改变。



燕尾榫

注：这里所说“接口”与Java的interface类似，但不一定需要显式地定义出来，也可以是开发人员之间的约定。

案例：生命游戏



地图

```
# game_map.py
class GameMap(object):

    def __init__(self, rows, cols):
        """地图将在逻辑模块进行初始化"""
        pass

    def reset(self, life_ratio):
        """重置地图并按life_ratio随机地填充一些活细胞"""
        pass

    def get_neighbor_count(self, row, col):
        """地图上一个方格周围活细胞数是游戏逻辑里的重要数据"""
        pass

    def set(self, row, col, val):
        """当游戏进行中，需要常常更新地图上方格的状态"""
        pass

    def get(self, row, col):
        """当需要将游戏状态呈现给用户时，就需要获取地图上方格的状态"""
        pass
```

案例：生命游戏



```
# life_game.py
```

```
class LifeGame(object):
```

```
    def __init__(self, map_rows, map_cols, life_init_ratio):
```

```
        """将在主程序中初始化实例"""
```

```
        pass
```

```
    def game_cycle(self):
```

```
        """
```

```
        进行一次游戏循环，将在此完成地图的更新
```

```
        将在计时器触发时被调用
```

```
        """
```

```
        pass
```

```
    def print_map(self):
```

```
        """由于暂时没有UI模块，因此先在逻辑模块进行地图的呈现"""
```

```
        pass
```

案例：生命游戏



计时

```
# game_timer.py
```

```
class GameTimer(object):
```

```
    def __init__(self, trigger, interval):
```

```
        """
```

将在主程序中初始化实例

计时器以interval秒的频率触发

trigger是个函数，计时器被触发时调用该函数

```
        """
```

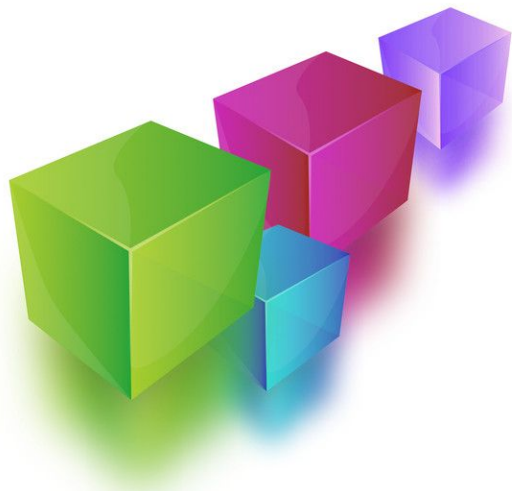
```
        pass
```

```
    def start(self):
```

"""启动计时器，之后将以interval秒的间隔持续触发"""

```
        pass
```

案例：生命游戏的实现



在模块化分解之后，开发人员可以分别实现各个模块。根据函数单一职责的原则，各个模块内部还会定义更多的函数。

与此同时，模块测试的设计工作也可以开始进行。

关键

前面抽象得到的各模块关键函数在后续开发中不应发生改变，这些函数一旦参数列表/名称/返回值等发生变化，可能造成连带性的一系列修改。

错误与异常处理



错误是导致程序崩溃的问题，例如Python程序的语法错误（解析错误）或者未捕获的异常（运行错误）等。



异常是运行时期检测到的错误，即使一条语句或者表达式在语法上是正确的，当试图执行它时也可能会引发错误。

异常处理是用于管理程序运行期间错误的一种方法，这种机制将程序中的正常处理代码和异常处理代码显式地区别开来，提高了程序的可读性。

Python程序异常处理

```
try:
    do_something()
except KeyboardInterrupt:
    exit(0)
except IOError as e:
    print("IO failed:", e)
```

- 细化具体的异常类型，更有针对地处理
- 减少 try/except 块中的代码量

- 在关键部分应该检查变量合法性，包括类型和取值范围等，避免“雪球效应”

```
def sqrt(n):
    assert isinstance(n, (float, int, ))
    assert n > 0
    return math.sqrt(n)
```

案例：生命游戏

- 目前给出的版本是通过命令行的方式运行并且用控制台打印出地图状态。
- 可以通过常见的Ctrl+C方式来终止命令行程序，但这会在Python中引发异常。

```
1 0 1 0 1 0 0 0 1 1
0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0 0
0 1 0 0 1 1 1 1 0 0
0 1 1 1 1 0 1 1 0 0
0 0 0 0 0 1 0 0 0 0
1 1 1 0 1 1 0 0 0 1
0 0 0 1 1 1 0 0 1 0
```

```
^CTraceback (most recent call last):
  File "main.py", line 38, in <module>
    exit(main(sys.argv[1:]))
  File "main.py", line 32, in main
    timer.start()
  File "/Users/Epsirom/MyCode/Project/LifeGame/game_timer.py", line 32, in start
    time.sleep(math.ceil(time.time() / self.interval) * self.interval - time.time())
```

案例：生命游戏

游戏启动部分



```
if __name__ == '__main__':  
    try:  
        exit(main(sys.argv[1:]))  
    except KeyboardInterrupt:  
        exit(0)
```

GameMap初始化



```
class GameMap(object):
```

```
    MAX_MAP_SIZE = 100
```

```
    def __init__(self, rows, cols):  
        assert isinstance(rows, int)  
        assert isinstance(cols, int)  
        assert 0 < rows <= self.MAX_MAP_SIZE  
        assert 0 < cols <= self.MAX_MAP_SIZE
```



谢谢大家！

THANKS

