代码的艺术

章淼

关于本课件

- 《代码的艺术》作为百度技术学院一年级生（面向新毕业同学）的必修课于2015年春推出
- 在百度内部，参加这门课学习的同学已经超过千人，受到很多同学的好评
- 2016年，《代码的艺术》被列入百度技术学院的精品课

个人简介

- 加入百度前
  - 清华(1994-2006)：博士，教师；1997年开始，网络方面的研究和开发
  - 用户产品研发(2006-2012)：搜狗，指南针，腾讯，木瓜
- 加入百度后(2012.11 -)
  - 百度运维部 **BFE团队** 技术负责人
  - 百度Python/Golang规范委员会成员
- 联系方式
  - zhangmiao02@baidu.com
- BFE团队的简介
  - We are hiring!
    http://www.newsmth.net/nForum/#!article/Career_Upgrade/500109
  - Golang在BFE的应用

http://www.infoq.com/cn/presentations/application-of-golang-in-baidu-frontend

Motivation

- 写代码，学校和公司有很大的不同
  - 学校：作业？实验室的项目？
  - 公司：工业级的产品
- 消除误解
  - 码农？35岁以上写不动代码？以后的出路是做管理？
  - 互联网公司就应该是加班加点？
- 建立正确的意识
  - 知和行，哪个更重要？
  - 很多人写了8-10年的代码，方法都还是错的
- 明确修炼的方向
  - 艺术品是由艺术家创造的
  - 艺术家是有道的
- 希望能够让大家对Software Engineer这个职业有一个新的认识

第一节 概念

主要内容：

- 代码
- 艺术
- 软件工程师(Software Engineer)

什么是艺术(Art)?

艺术（文化名词） 编辑

艺术是指用形象来反映现实但比现实有典型性的社会意识形态，包括文学、书法、绘画、雕塑、建筑、音乐、舞蹈、戏剧、电影、曲艺等。

中华奇石馆馆长李文科介绍：我们将“艺术”定义为人类通过借助特殊的物质材料与工具，运用一定的审美能力和技巧，在精神与物质材料、心灵与审美对象的相互作用下，进行的充满激情与活力的创造性劳动。可以说它是一种精神文化的创造行为，是人的意识形态和生产形态的有机结合体。

代码/编码可以成为艺术

- 借助的物质：计算机系统
- 工具：设计、编写、编译、调试、测试、...
- 编码需要激情
- 编码是非常具有创造性的工作

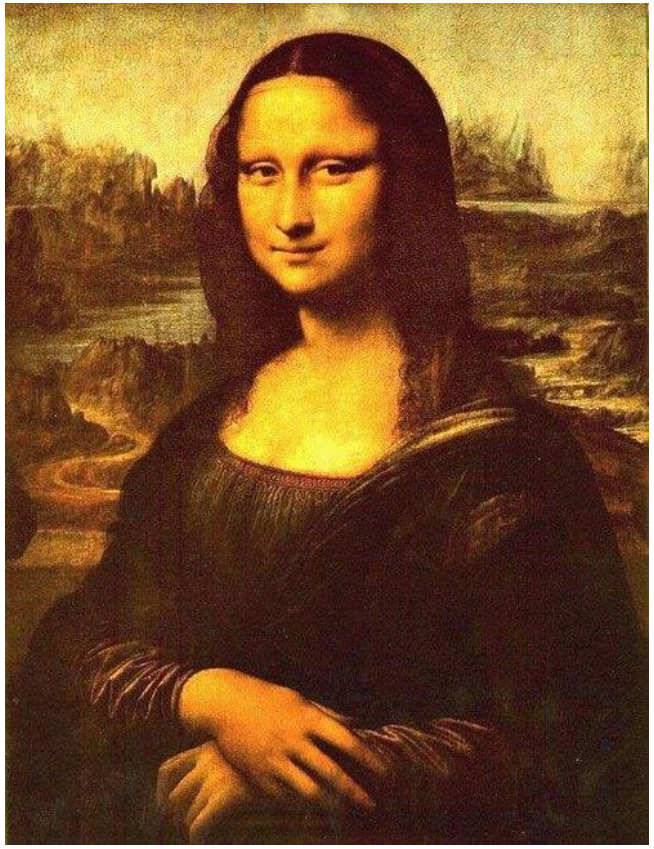
代码是人类智慧的结晶

代码反映了一个人/团队的精神

软件工程师 != 码农

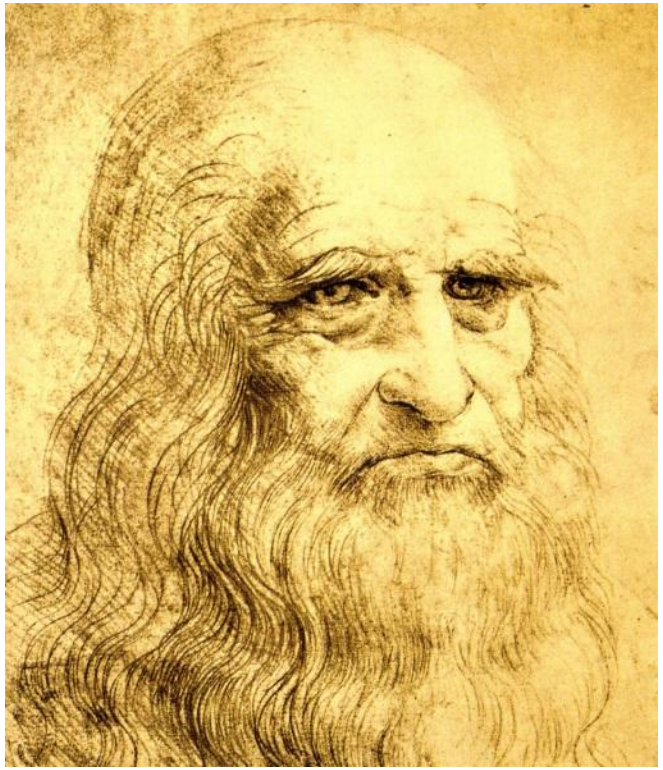
- 软件工程师只需要会写代码？
- No! 软件工程师应该具有综合的素质
- 技术
  - 编码能力，数据结构，算法
  - 系统结构，操作系统，计算机网络，分布式系统
- 产品
  - 对业务的理解，交互设计，产品数据统计，产品/业务运营
- 项目管理
- 研究和创新
  - R&D: Research and Development
- 一个系统工程师的培养需要至少8-10年的时间

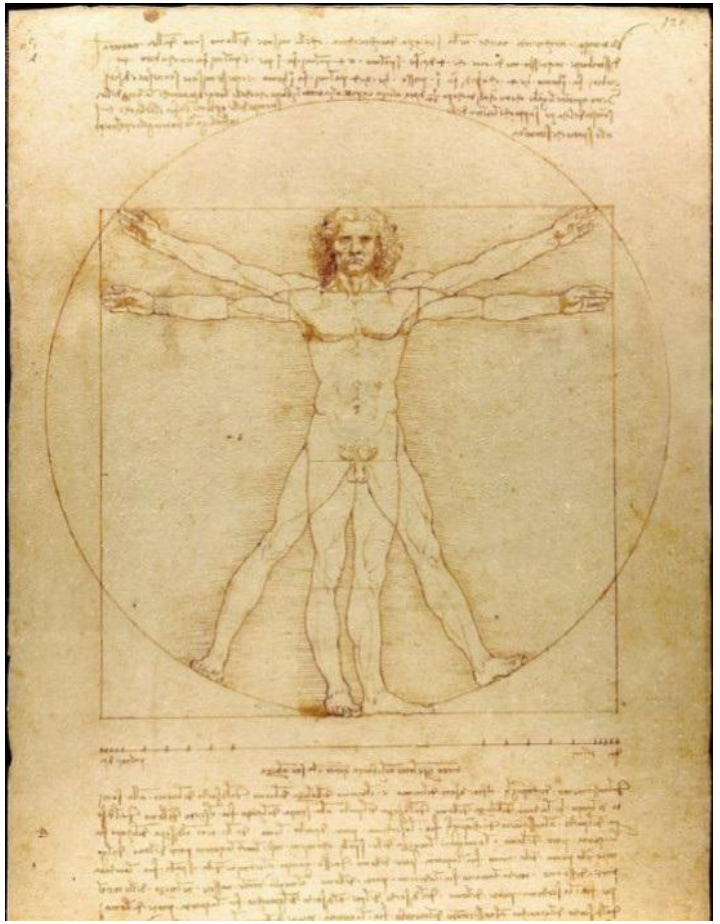
艺术，解读的角度



艺术品

艺术家





艺术家是怎么思考的

Coding is NOT so easy

- Coding的过程是：
  - 从 无序 变为 有序
  - 将 现实世界中的问题 转化为 数字世界的模型
  - 一个 认识 的过程(从 未知 变为 已知)
- 在Coding的过程中，需要
  - 把握问题 的能力
  - 建立模型 的能力
  - 沟通协作 的能力
  - 编码执行 的能力
- 写好代码需要首先 建立品味

第二节 实践

主要内容

- 什么是好的代码？
- 好的代码从哪里来？

一流代码的特性

- 高效 (Fast)
- 鲁棒 (Solid and Robust)
- 简洁 (Maintainable and Simple)
- 简短 (Small)
- 共享 (Re-usable)
- 可测试 (Testable)
- 可移植 (Portable)
- 可监控 (Monitorable)
- 可运维 (Operational)
- 可扩展(Scalable & Extensible)

正确和性能

可读和可维护

共享和重用

运维和运营

注：前7条来自《写好代码的十个秘诀》by 林斌, 2000

Bad Code的一些表现

- Bad function name
  - do(), myFunc(), …
- Bad variable name
  - a, b, c, l, j, k, temp
- No comments
  - 太多人缺乏“讲故事”的能力
- The function has no single purpose
  - LoadFromFileAndCalculate()
- Bad layout
  - 极端：把所有代码写为一行？
- None testable
- …

好的代码从哪里来？

- 代码不只是”写”出来的
  - 编码的时间一般只占10%
- 好的代码是多个环节工作的结果
- 编码前（经常被忽略或轻视的环节）
  - 需求分析，系统设计
- 编码中
  - 编写代码，单元测试
- 编码后
  - 集成测试，上线，持续运营/迭代改进
- 一个好的系统/产品是以上过程持续循环的结果

需求分析 & 系统设计

- 这是两个经常被忽视的环节
  - 有太多的人错误的认为，写代码才是最重要的事情
  - 有太多的项目，在没有搞清楚目标之前、就已经开始编码了
  - 有太多的项目，在代码基本编写完成后、才发现设计思路是有问题的
- 软件开发规律和人的直觉是相反的
  - 在前期更多的投入，收益往往最大
  - Why? 除了开发，测试、上线、调试等都是很大的成本
- 有太多人无法搞清两者的差别
- 需求分析：定义系统/软件 黑盒的行为 (external, what)
- 系统设计：设计系统/软件 白盒的机制(internal, how & why)

需求分析

- 问题：怎么用寥寥数语勾勒出一个系统的功能？
- 每个系统都有自己的定位，以GFS为例
  - GFS is designed to provide efficient, reliable access to data using large clusters of commodity hardware.
- 怎么描述GFS的需求？
  - 分布式文件系统
  - 文件数量；文件大小的分布；总的存储容量
  - 读写能力（读写文件次数，数据传输速率，读写延迟）
  - 容错的能力；一致性方面的定位（强一致，弱一致）
  - 对外的接口（用户怎么使用）
  - …
- 需求需要用精确的数字来刻画
  - 量变导致质变

系统设计

- 什么是系统架构(System Architecture)?
- A system architecture is the conceptual model that defines the structure, behavior, and more views of a system.(from wiki)
- 几个要素
  - 系统要完成哪些功能
  - 系统如何组成
  - 功能在这些组成部分之间如何划分
- 系统设计的约束
  - 资源的限制：计算，存储，IO/网络
- 需求是系统设计决策的来源
  - 在设计中，经常需要做Trade-Off
  - 需求是make decision的重要依据

系统设计的风格和哲学

- 在同样的需求下，可能出现不同的设计
- 电信网络 vs. Internet
  - 中心控制 vs 分布式控制
  - 可靠组件 vs 不可靠组件
- CISC(复杂指令集) vs RISC(精简指令集)
- 好的系统是在 合适假设 下的精准平衡
  - 一个通用的系统，在某个方面是不如专用系统的
- 每个组件（子系统/模块）的功能都应该足够的专注和单一
  - 功能的单一是复用和扩展的基础
- 子系统/模块之间的关系应该简单而清晰
  - 软件中最复杂的是**耦合**（为什么全局变量是要极力避免的？）

关于接口(interface)

- 系统对外的**接口**，比系统实现本身还要更重要
  - 这个问题被太多人所忽视
- Why?
  - 接口定义了功能：如果功能不正确，系统就没有用
  - 接口决定了外部关系：相对于内部，外部关系确定后非常难以修改
- 哪些是接口？
  - 模块对外的函数接口
  - 平台对外的API（很多是RPC或Web API）
  - 系统间通信的协议（问题：什么是协议？）
  - 系统间存在依赖的数据（比如：给另外一个系统提供的词表）
- 设计和修改接口，需要考虑的非常清楚
  - 合理，好用
  - 修改时需要尽量向前兼容

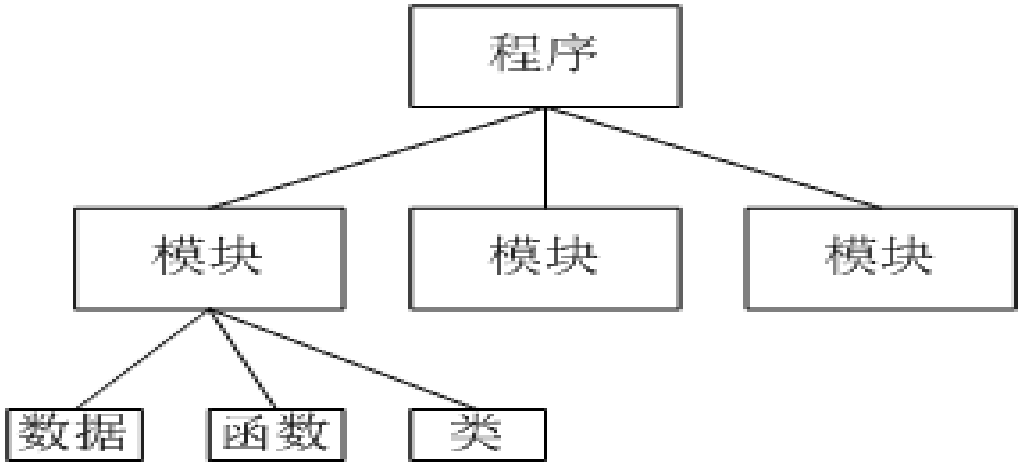
第三节 怎么写代码

- 代码也是一种表达方式
- 模块，函数，代码块，命名，...

代码也是一种表达方式

- 一个项目中，超过50%的时间用于沟通
- 软件的维护成本远高于开发成本
- 沟通有哪些形式？
  - 面对面交流，Email，文档，代码，…
- 现在，代码主要是写给人看的
  - 曾经要为机器考虑很多，但现在编译器已经做了很多
- 编程规范，一般包含的内容
  - 代码如何规范表达
  - 一些语言相关的注意事项
- 理想的场景
  - 看别人的代码感觉和看自己的代码一样
  - 看代码时能够专注于逻辑，而不是格式方面
  - Don’t make me think

模块(module)



- 模块是程序的基本组成单位
  - 一个c/py/go文件就是一个模块
- 模块需要有明确的功能
  - 紧内聚，松耦合
- 怎么切分模块是一个需要慎重考虑的事情
- 切分模块的一种角度
  - 数据类的模块
  - 过程类的模块

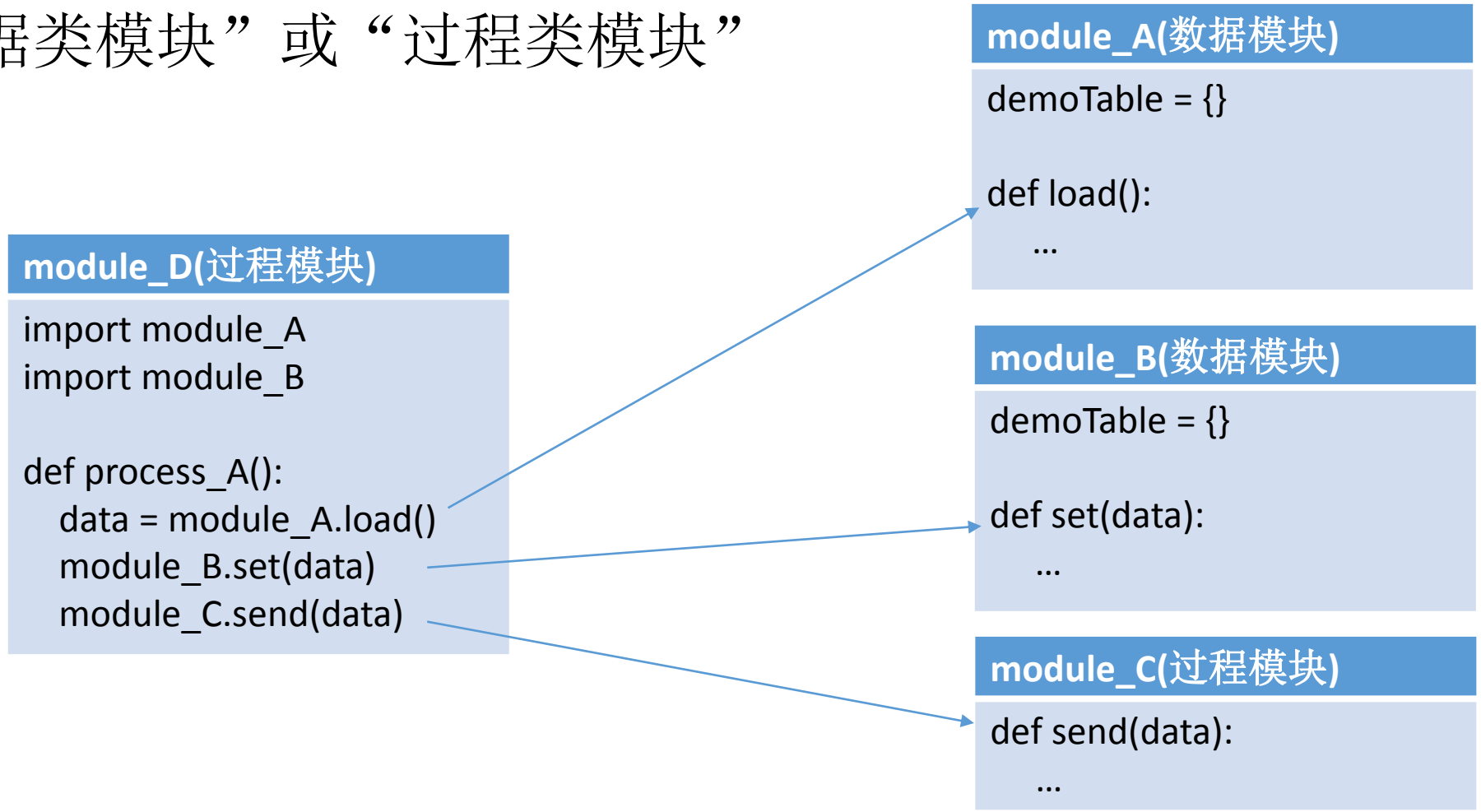
数据类的模块

- 主要完成对数据的封装
  - 模块内部变量；
  - 类的内部变量
- 对外提供明确的数据访问接口
  - 数据结构和算法属于模块内部的工作
- 写程序要以数据为中心考虑
  - 很多程序最后不好维护都是数据封装没做好

```
class demoTable(object):
    def __init__(self):
        self.table = {}

    def set(key, value):
        self.table[key] = value

    def get(key):
        return self.table[key]
```

过程类的模块

- 本身不含数据
- 调用“数据类模块”或“过程类模块”



模块的重要性

- 好的模块划分是软件架构稳定的基础
  - 稳定的模块功能/接口 =>稳定的软件架构
- 减少模块间的耦合 => 降低软件复杂性
- 清晰的模块有利于代码的复用

类和函数

- 类和函数是两种不同的模型，有它们各自适用的范围
  - 尽力想用一种方式来描述整个世界，这可能不是一个好主意
  - Java的设计是一个反例：试图用对象来统一描述
- 推荐：和**类的成员变量**无关的函数，作为一个**独立的函数**
  - 不建议实现为**类的成员函数**来写
  - 便于未来的复用

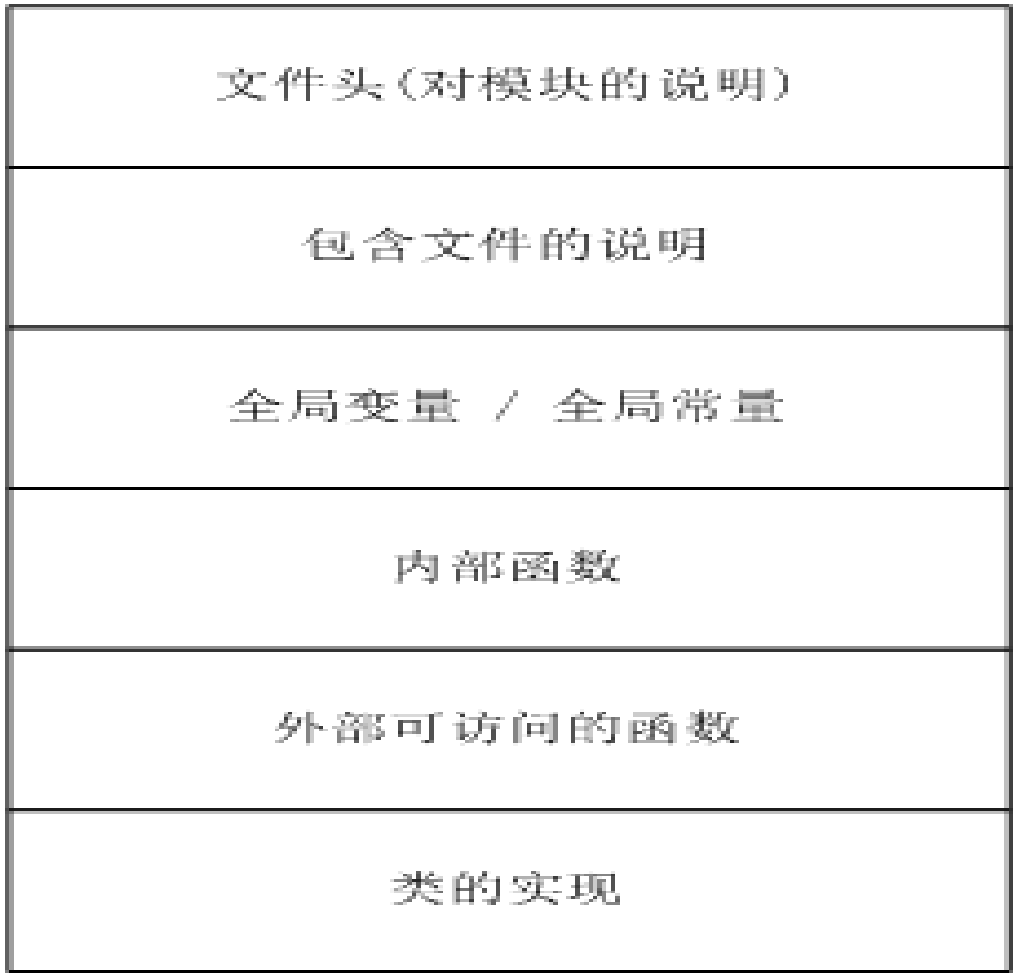
```
def func(a):
    return a * 2

class SortTable(object):
    def _internal_get(self):
        return self.value

    def get(self):
        value = self._internal_get()
        return func(value)
```

关于Object Oriented

- OO是一个好主意，但是真正理解的人不多
  - 有太多的人，使用OO的语言，写着非OO思想的程序
- OO的本质是数据封装
  - 写程序应该从Data开始想问题
  - But, 周围有太多人是从执行过程开始考虑的
- C是面向过程的，C++是面向对象的？
  - 这是一个普遍的认知错误
  - C是**基于对象(Object Based)**的，主要是没有**多态**和**继承**
- C++是一个被滥用的语言
  - 编程语言重要的是实现功能，而不是咬文嚼字（“回”字有几种写法？）
  - **多态**和**继承**都是应该被谨慎使用的
  - 一个悖论：很好的继承模型需要对需求的准确把握；而在初始设计阶段往往对需求不会有很好的理解
  - 问题：系统是从开始就设计好的，还是逐步发展和改造好的？

模块内部的组成

- 文件头
  - 模块的说明：模块功能的简要说明
  - 修改历史：时间，修改人，修改内容
  - 其它必要的、模块级的说明
- 模块内，内容的顺序尽量保持一致
  - 降低阅读的成本



文件头的例子

```
#!/usr/bin/python
#-*- coding: utf8 -*-
"""
Unit testing for rss_hdb

Description:
This file implements unit-testing cases for rss_hdb. It is using the
framework provided by PyUnit.
"""
# modification history:
# ---------------------
# 2012/12/15, by Zhang Miao, Create
#
# Structure of files on disk:
# /rss_hdb
#         rss_feeds.dat: store urls of all feeds
#         /(md5 of feed1): store all information for a rss feed
#                  rss.feed : feed information
#                  rss.idx  : an index for rss data
#                  rss.items: all rss items of this feed
#         /(md5 of feed2)
#         ...
#
```

函数

- 系统 => 子系统/程序 => 模块 => 函数
  - 函数的切分也很重要
- 函数描述的3要素
  - 功能描述：这个函数是做什么的
  - 传入参数描述：含义，限制条件
  - 返回值描述：各种可能性
- 函数的规模
  - 要足够的短小
    - Python：尽量在一屏内（约30行）完成
    - C/C++：尽量在两屏内完成
- 写好程序的一个秘诀：把函数写的短一些
  - Bug往往出现在哪些非常长的函数里
  - 出现危险的根源，往往是过于自信

函数的返回值

- 每个函数应该有足够明确的语义
  - 基于函数的语义，函数返回值有3种类型
- True, False
  - “逻辑判断型”的函数，表示“真”或“假”
  - 如：is_white_cat()
- OK, ERROR
  - “操作型”的函数，表示“成功”或“失败”
  - 如： data_delete()
- Data, None
  - “获取数据型”的函数，表示“有数据”或“无数据 / 获取数据失败”
  - 如： data_get()

函数头的例子

- 函数头：对函数的语义做出清晰和准确的说明

```
def fetch_bigtable_rows(big_table, keys):
    """Fetches rows from a Bigtable.

    Retrieves rows pertaining to the given keys from the Table instance
    represented by big_table.

    Args:
        big_table: An open Bigtable Table instance.
        keys: A sequence of strings representing the key of each table row
            to fetch.

    Returns:
        A dict mapping keys to the corresponding table row data
        fetched. Each row is represented as a tuple of strings. For
        example:

        {'Serak': ('Rigel VII', 'Preparer'),
         'Zim': ('Irk', 'Invader'),
         'Lrrr': ('Omicron Persei 8', 'Emperor')}

    Raises:
        IOError: An error occurred accessing the bigtable.Table object.
    """
    pass
```

函数：单入口单出口

- 单入口、单出口是一种推荐的方式
- 多线程下数据表的实现
  - 使用一个内部函数来实现”单入口单出口”

```
class SortTable(object):
    def __init__(self):
        self.lock = threading.Lock()

    def get(self):
        self.lock.acquire()
        if ***:
            self.lock.release()
            return ***
        elif ***:
            self.lock.release()
            return ***
        else:
            self.lock.release()
            return ***
```

```
class SortTable(object):
    def __init__(self):
        self.lock = threading.Lock()

    def _get(self):
        if ***:
            return ***
        elif ***:
            return ***
        else:
            return ***

    def get(self):
        self.lock.acquire()
        value = self._get()
        self.lock.release()
        return value
```

代码块：Bad case

```
if len(buffer) < (LOG_HEAD_LEN + LOG_TIME_LEN):
    # no enough data to get log_head and log_time
    return (False, None, buffer)
logHeader, buffer = _logHeadRead(buffer)
if logHeader == None:
    # fail to read logHead from buffer
    return (True, None, buffer)
if logHeader[LOG_HEAD_POS_COMPRESS_LEN] != 0:
    # some code is omitted
offset = LOG_HEAD_LEN + LOG_TIME_LEN + logHeader[LOG_HEAD_POS_UNCOMPRESS_LEN]
if len(buffer) < offset:
    # no enough data, wait for the next time
    return (False, None, buffer)
recordStr = buffer[(LOG_HEAD_LEN + LOG_TIME_LEN):offset]
buffer = buffer[offset:]
```

代码块：Good case

```
if len(buffer) < (LOG_HEAD_LEN + LOG_TIME_LEN):
    # no enough data to get log_head and log_time
    return (False, None, buffer)

# read loghead
logHeader, buffer = _logHeadRead(buffer)
if logHeader == None:
    # fail to read logHead from buffer
    return (True, None, buffer)

# check whether it is compressed record
if logHeader[LOG_HEAD_POS_COMPRESS_LEN] != 0:
    # some code is omitted

# check whether record is completely in the buffer
offset = LOG_HEAD_LEN + LOG_TIME_LEN + logHeader[LOG_HEAD_POS_UNCOMPRESS_LEN]
if len(buffer) < offset:
    # no enough data, wait for the next time
    return (False, None, buffer)

# get record out of the buffer
recordStr = buffer[(LOG_HEAD_LEN + LOG_TIME_LEN):offset]
buffer = buffer[offset:]
```

代码块

- 讨论范围：一个函数内的代码实现
- 思路：把代码的段落分清楚
- 形式的背后是逻辑（划分，层次）
  - 千万不要认为那些 空行 / 空格 是可有可无的
- Don’t make me think
  - 一眼看过去，如果无法看清逻辑，这不是好代码
  - 好的代码不需要你思考太多
  - 一定记住：代码更是写给别人看的
- 注释不是 补出来的
  - 我的习惯是先写注释，后写代码
- Btw
  - 一个在代码上表达不好的同学，在其他表达上一般也存在问题
  - 其他表达：文档、email、ppt、口头沟通、...

命名

- 命名的范围：系统，子系统，模块，函数，变量/常量，...
- 为什么命名如此重要?
  - “望名生义”是人的自然反应
  - 概念是建立模型的出发点 （概念，逻辑推理 => 模型体系）
- 普遍存在的问题：
  - 名字不携带信息：do, a, b, ...
  - 名字携带的信息是错误的：体会一下

    set() vs update(), isXXX() vs check()
- 命名不是一件很容易的事情
  - 要求：准确，易懂
  - 起一个好名字很多时候需要经过推敲
- 名字的可读性：下划线，驼峰
  - ClassName, GLOBAL_CONSTANT_NAME, module_name

在互联网时代，系统是运营出来的

- 系统的可监测性非常重要
- 如果没有足够的数据收集，系统等于没有上线
  - 举例：提供了跨集群重试的功能，到底生效了多少次？
  - 类比：产品的上线
- 对一个系统来说，数据和功能同等重要
  - 功能可以依靠线下测试来验证
  - 数据只能依靠线上的运转和运营
- 在设计和编码阶段，就要考虑系统的运营
  - 提供足够的状态记录
  - 提供方便的对外接口

第四节 修身

- 怎么才能成为优秀的软件工程师？

怎样修炼成为优秀的软件工程师？

- 一个好的程序员和以下因素没有必然联系
  - 写了多少年程序
  - 写了多少行代码
  - 曾经在哪里上学，曾经在哪里工作
- 希望大家注意以下三个方面
  - 学习－思考－实践
  - 知识－方法－精神
  - 基础乃治学之根本

学习 – 思考 – 实践

- 学习
  - 软件编写的历史已经超过半个世纪，有太多的经验可以被借鉴
  - 途径：书，开源代码
  - 有hungry和foolish的感觉才会去学习
  - 最忌井底之蛙、夜郎自大
- 思考
  - 学而不思则罔
  - 不经过思考，不能形成自己的思想，等于白学和白干
- 实践
  - 知行合一 谓之 善
  - 所有重要的进步，都来源于失败和挫折的经历

知识 - 方法 – 精神

- 知识是过时最快的
  - 只学知识的人，总是感觉世界变化太快
  - 编程语言、新的系统、…
- 方法
  - 这个词总是让某些人感觉很虚
  - 但是这个虚(方法)可能比那个实(知识)更有价值：道可道，非常道
  - 分析问题、解决问题的能力才是最重要的
  - Research: To Identify the Fundamental Problem, and Solve it.
- 精神
  - 即使有了知识和能力，To be or not to be永远都是一个问题
  - 前进的路上往往不是鲜花和掌声，而是困难和荆棘
  - 人类总是在神性和兽性间不断斗争，进步往往来自于对理想的追求
  - 自由精神，独立思想；Don’t Follow；对完美的不懈追求

基础乃治学之根本

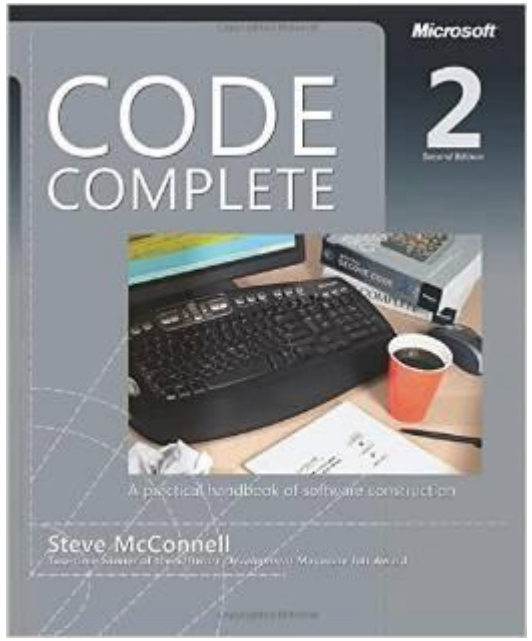
- 于敏，2015年国家最高科技奖得主，中国氢弹元勋
- 于敏特别喜欢诸葛亮在《诫子书》中的格言：非宁静无以致远，视之为座右铭
- 于敏本性沉静好思，喜欢寻根问底。在耀华中学时期他逐渐形成了自己的思维模式和学习方法，善于薄书厚读，再由博而约，由厚而薄，融会贯通，得其精髓。他非常喜欢魏征谏唐太宗的两句话：“求木之长者，必固其根本；欲流之远者，必浚其泉源”，深知基础乃治学之根本。
- 软件工程师的基础
  - 数据结构，算法；操作系统，系统结构，计算机网络；
  - 软件工程，编程思想
  - 逻辑思维能力，归纳总结能力，表达能力
  - 研究能力：分析问题，解决问题
- 这个基础的建立，至少也要5-8年之功

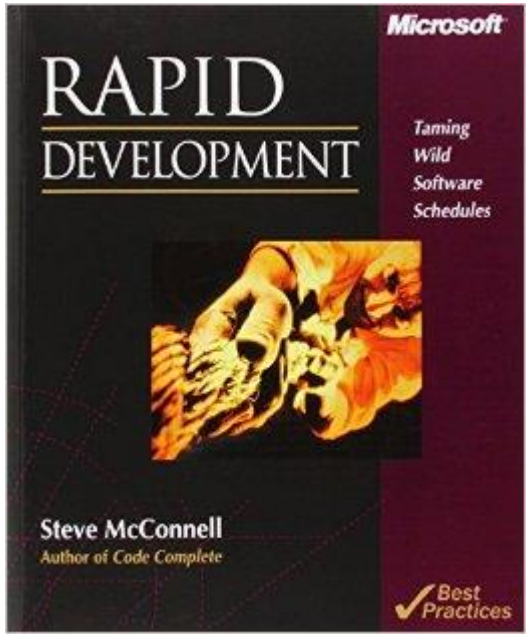
Summary

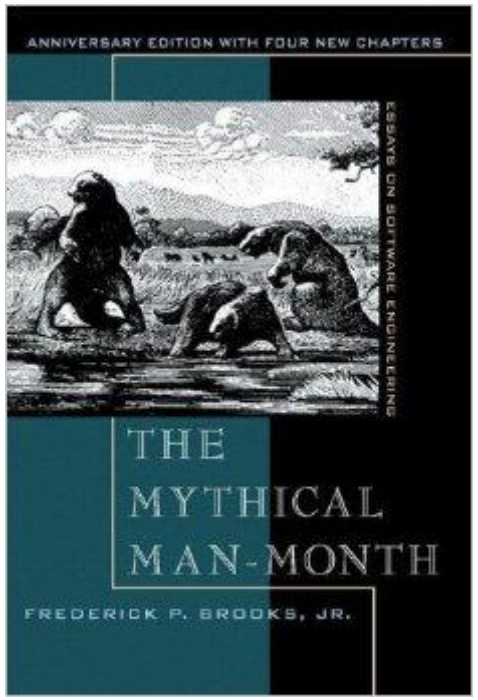
- 软件工程师 != 码农
- 代码 可以是 艺术品，也可以是 垃圾
- 不要忘记我们为什么出发
  - 我们的目的是改变世界，而不是学习编程、或炫耀技术
- 好代码的来源 不是 写好代码
  - 好代码是一系列工作的结果
- 代码是写给别人看的
  - 别人看不懂的代码是失败的
- 写好代码是有道的
  - 系统工程师至少需要 8 – 10年的积累

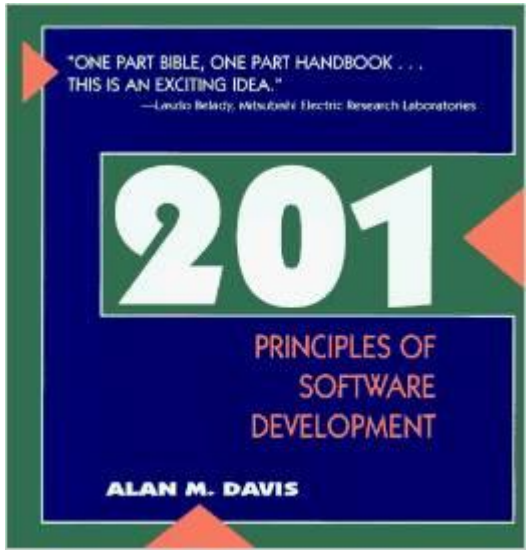
推荐阅读(1)

- Code Complete
- Rapid Development
- The Mythical Man-Month
- 201 principles of software development



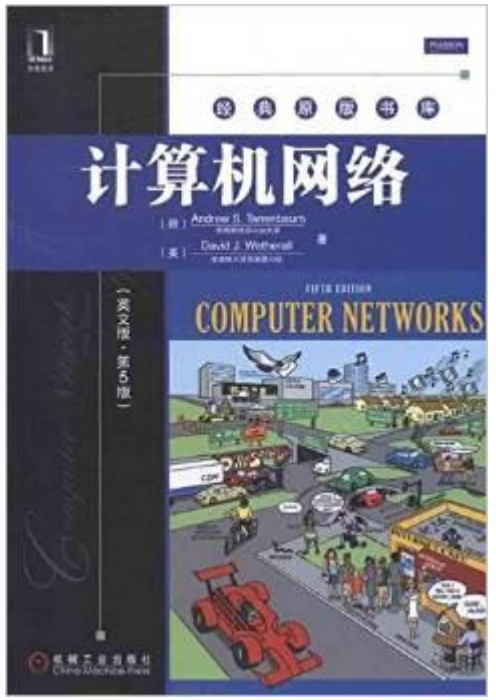


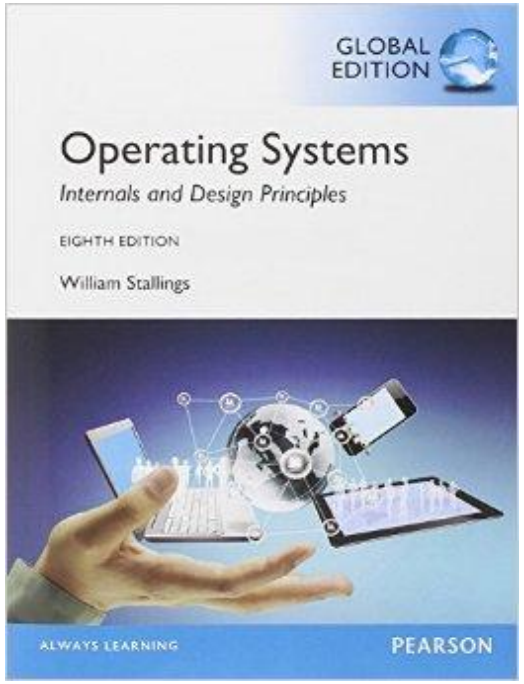


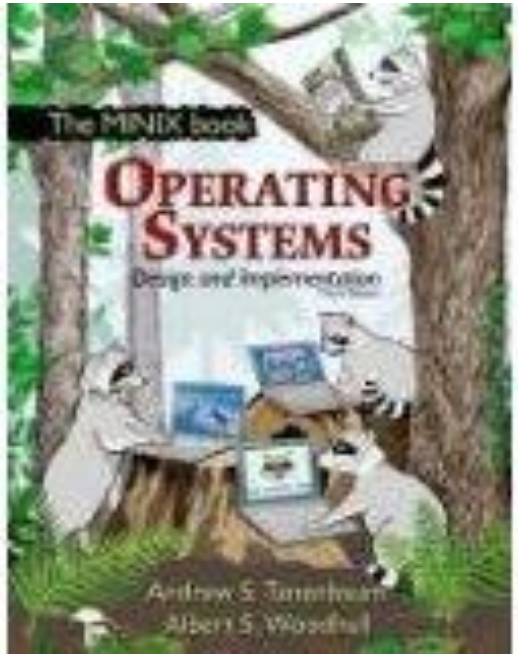


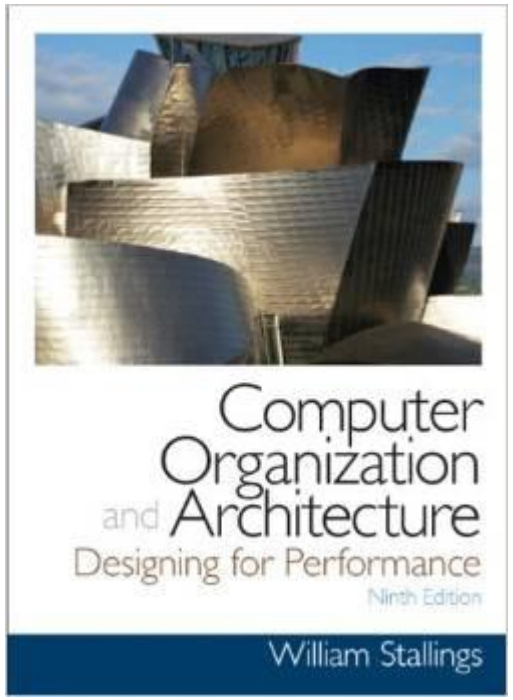
推荐阅读(2)

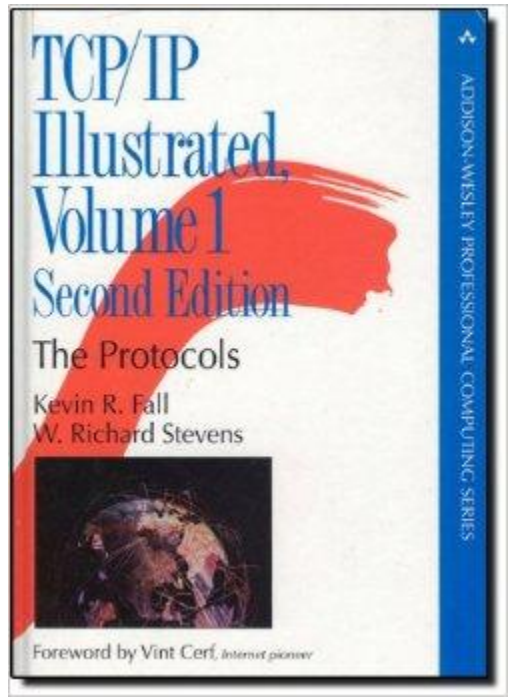
- Computer Organization and Architecture
- Operating Systems Internals and Principles
- Operating Systems Design and Implementation
- Computer Networks
- TCP/IP Illustrated











伟大的代码

永远是

伟大团队 **精神**

的反映