计算思维与计算科学

控制复杂性

* 第十一章 控制复杂性的方法

控制复杂性是计算机编程的本质

Brian Kernighan

* 控制复杂性

* 复杂性与抽象
* 模块化
* 层次化
* 分散化
* 工程化

复杂性的来源

- 复杂系统
  - 体量巨大
  - 多路径连接和反馈
  - 非线性行为
  - 不一致、不均匀

软件的本质复杂性

- 问题域的复杂性
  - 外部复杂性
- 管理开发过程的困难性
  - 团队规模带来的挑战
- 通过软件可能实现的灵活性
  - 同一工作可以有多种不同的实现
- 大型离散系统的行为复杂性
  - 复杂系统

* 软件复杂性的分类

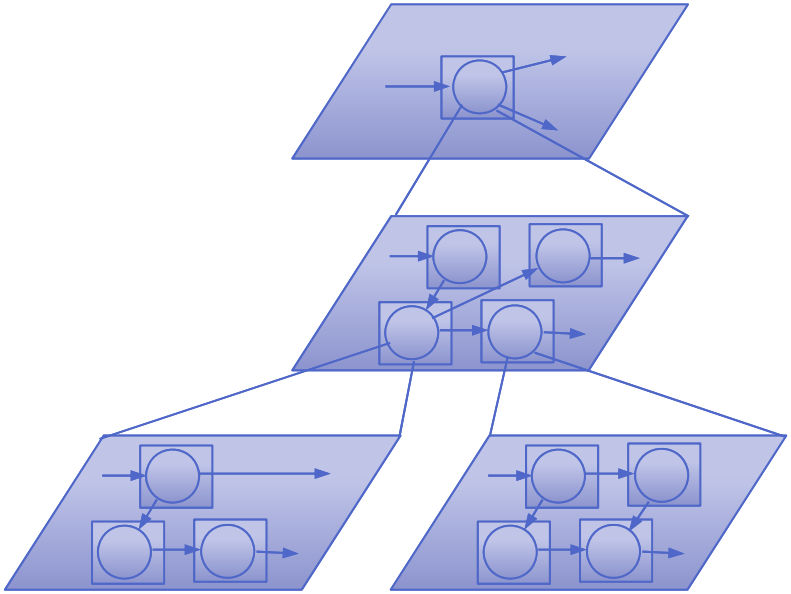
* 计算复杂性
  * 时间复杂性、空间复杂性
  * 算法
* 数据复杂性
  * 数据结构
  * 面向对象
* 结构复杂性
  * 抽象

抽象

- 抽象:把事物整体中某一方面的本质抽取出来，而不陷入细节。
- 这种本质特性就是抽象层次，它们是确定的，抽象用来局部化不确定性；
  - 典型如各种框架，如EJB框架、Spring Framework，HiveMind框架等。
- 自底向上：在系统中将某些确定的部分分离出来，然后对这些确定的部分先提供解决方法，用来局部化确定性部分
  - 典型如各种工具库，如面向C语言提供的标准IO库，面向Java的标准类库等，这些库在设计的时候并没有局限用于某个特定的领域。

* 抽象、分解与自顶向下

* 抽象：从作为整体的系统开始(顶层)，每一抽象层次上只关注于系统的本质特征（输入输出）
* 分解：将系统不断分解为子系统、模块……
* 随着分解层次的增加，抽象的级别越来越低，也越接近问题的解(算法和数据结构)



*控制结构复杂性

*水平轴：模块化
*垂直轴：层次化
*尺度上：分散化
*时间轴：工程化

*模块化

*将整个系统分解成若干子系统（模块），子系统可以继续分解，直到可以完全把握

*系统结构图是一棵树

*高内聚、低耦合

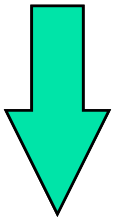
*模块间接口清晰规范

*模块具有通用性和可互换性

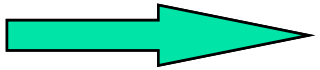
* 模块化(Modularity)

模块化是控制复杂性一个基本方法

高层模块 ——从整体上把握问题，隐蔽细节



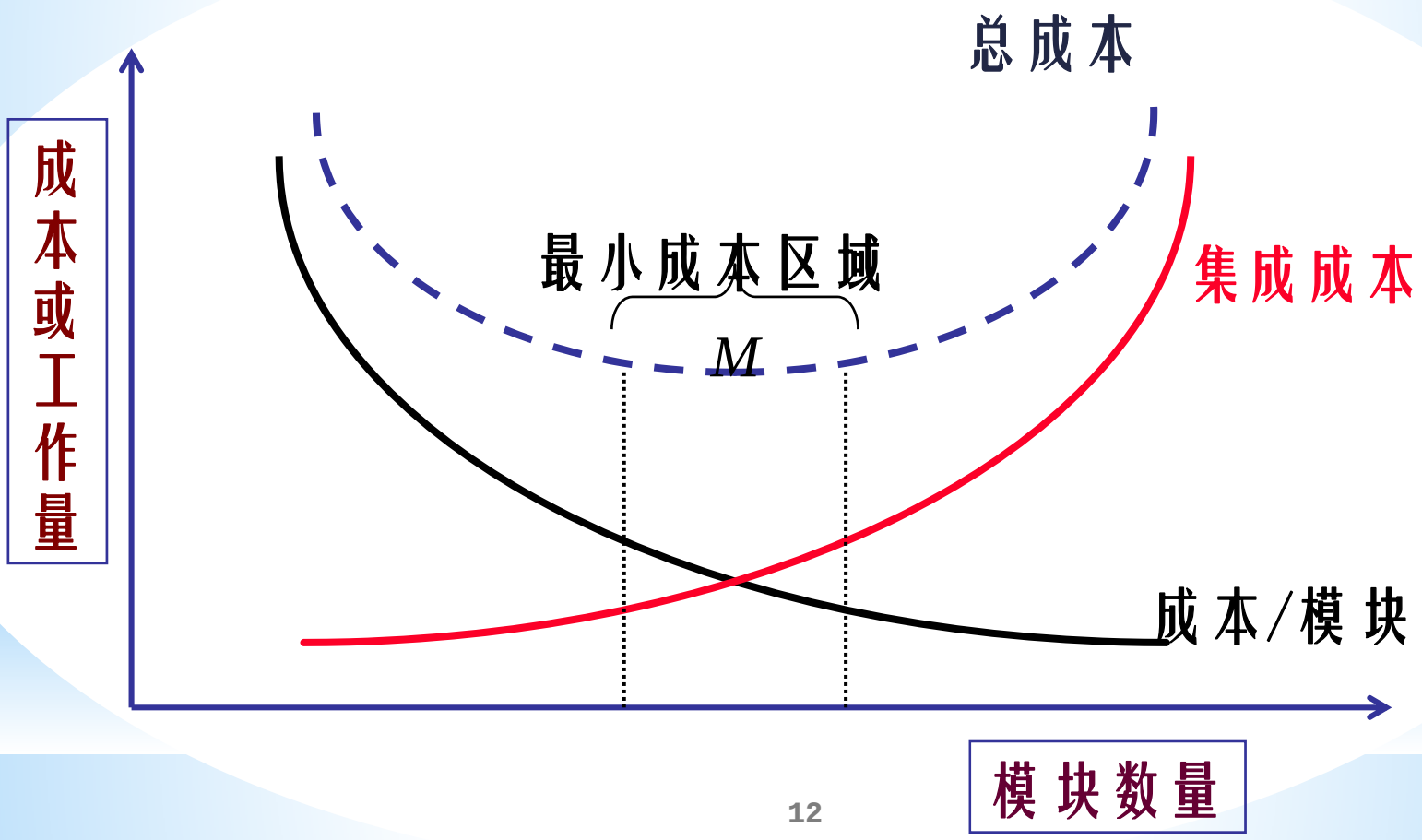
分解

复杂问题  较小问题

分解

可减小解题所需的总的工作

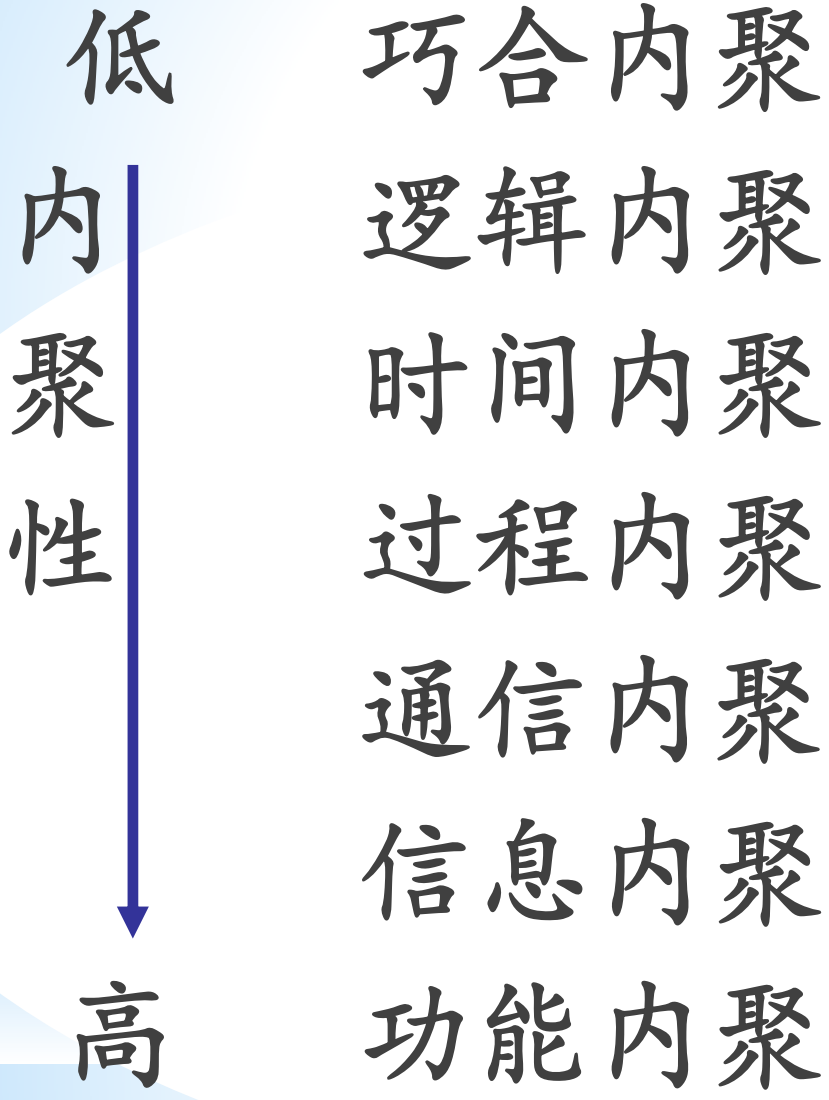
* 模块数与工作量的关系

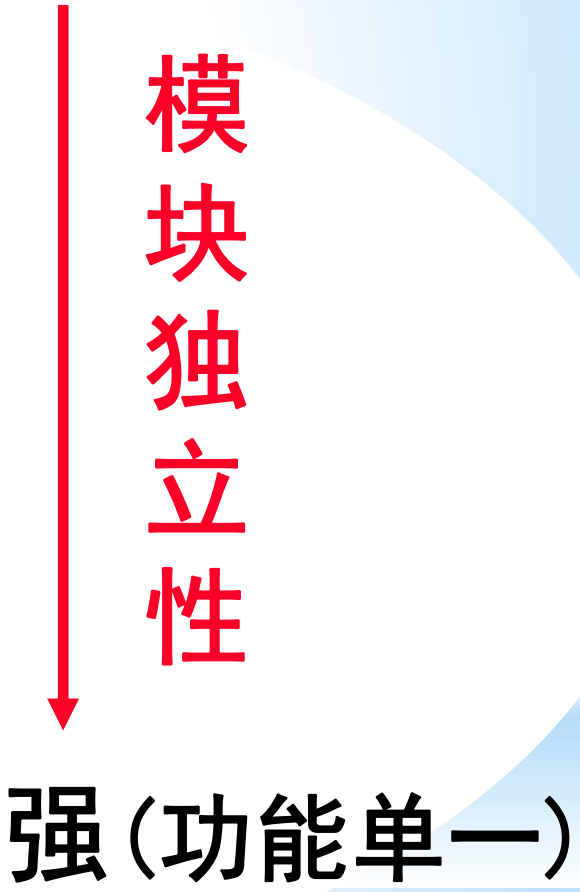


模块的独立性

- 独立性
  - 模块完成独立的功能
  - 符合信息隐蔽和信息局部化原则
  - 模块间关连和依赖程度尽量小
- 度量
  - 模块自身的内聚性
    - 一个模块内部各成分之间相互关联的强度
    - 高内聚(一模块的所有成分都直接参与并且对于完成同一功能来说都是最基本的)
  - 模块之间的耦合性

* 模块的内聚性





模块间的耦合性

- 耦合性是模块间相互依赖程度的度量
  - 耦合的强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。
- 耦合性越高，模块独立性越弱

* 耦合性级别

耦合性降低，独立性提高

耦合级别	解释
内容耦合	直接使用另一个模块的内部数据，或通过非正常入口而转入另一个模块内部
共享耦合	通过公共全局数据相互耦合
外部耦合	通过外部信息（如文件）相互耦合
控制耦合	通过传递控制变量（如开关、标志等），选择被调用模块内部的功能
特征耦合/标记耦合	使用复杂的数据结构作为参数
数据耦合	使用简单的数据类型作为参数
消息耦合	使用消息完成功能调用，控制分离、可延迟、可并发

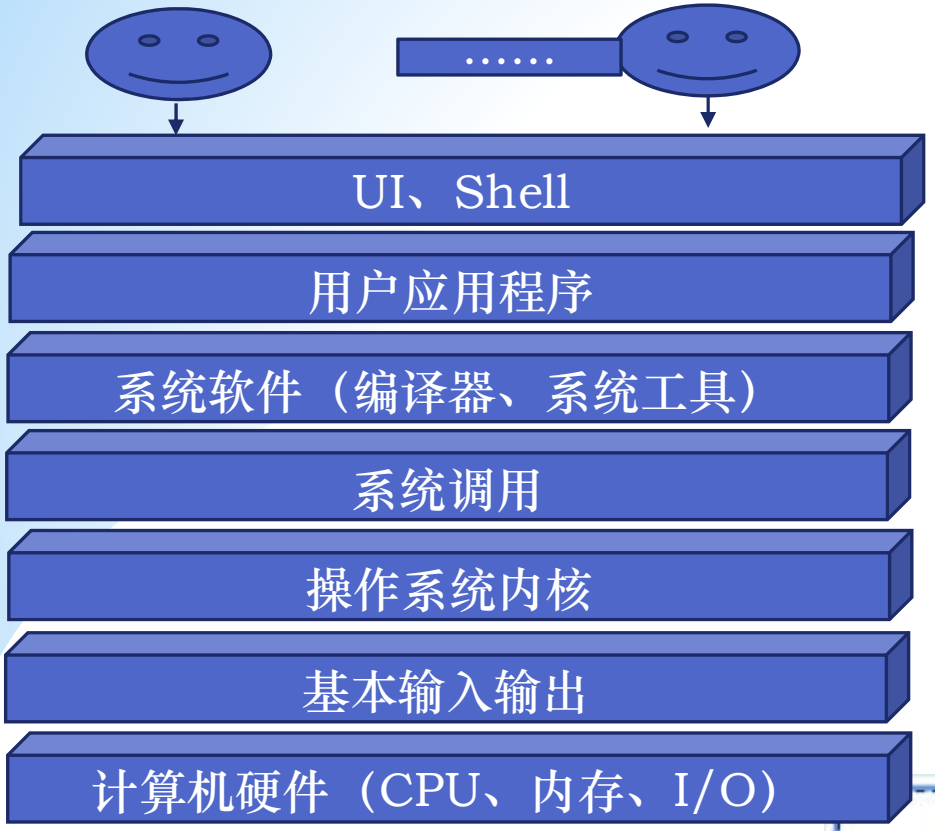
层次化

- 透明性
  - 某个内容虽然存在，但从某个角度看不到。
  - 对于上层使用者来说，无需了解下层的具体实现方式和细节
- 优点
  - 层次之间不相互干扰，不同层次关注不同的目的
  - 降低研发、学习的复杂性
  - 可扩展、兼容性强

层次结构模型

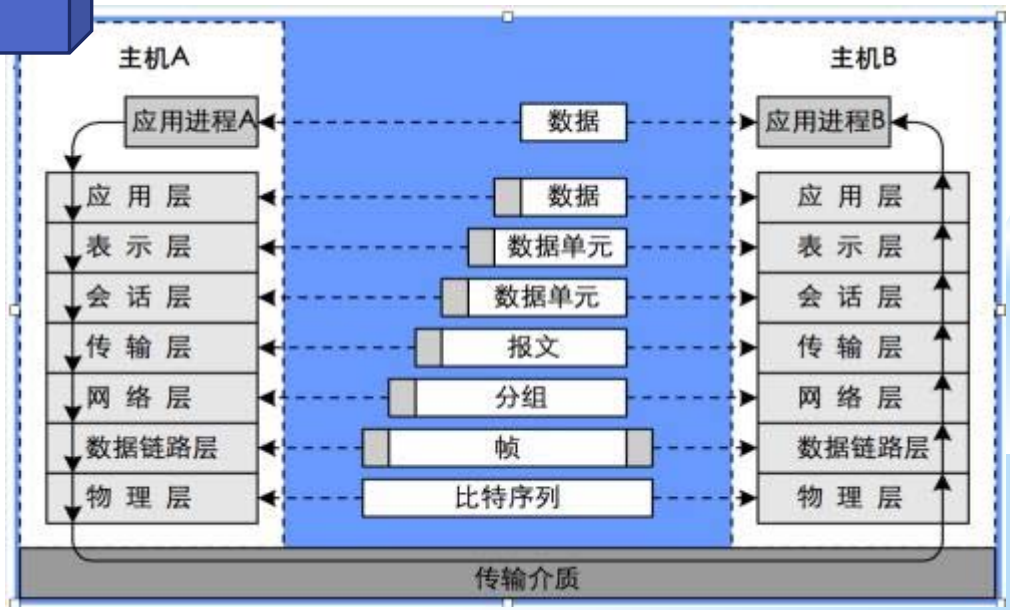
- 层次性一直都是软件的问题分析和设计实施的基本和具有普遍适用的思想方法
  - OS
  - Network
- 层次系统（Layered Systems）是一种体系结构风格

* 层次模型举例

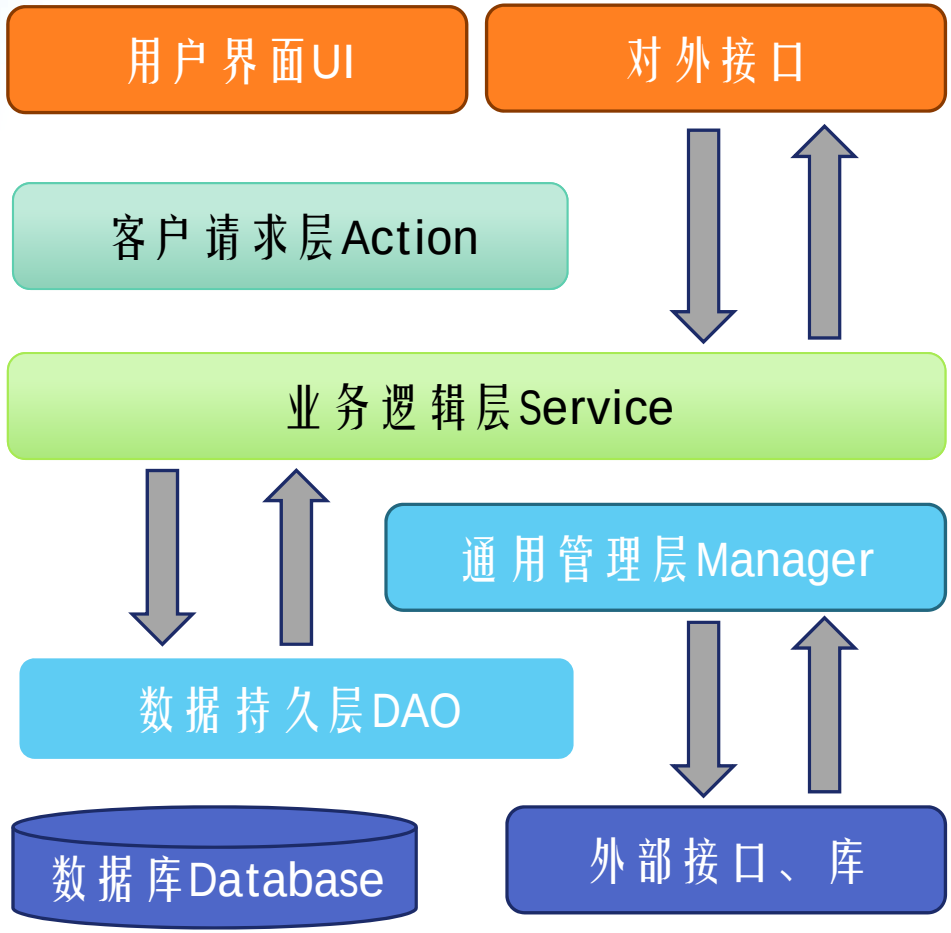


计算机操作系统的体系结构

计算机网络的体系结构



*一个常见的软件层次模型



层次结构

- 层次体系结构的对比、总结
  - 都是从基本实现（硬件的构成）开始
  - 系统设计中都考虑道了系统的升级和扩展性、兼容性
  - 建立在各基础层服务之上的系统，对于性能可以建立可追踪的分析估计
  - 区别：
    - 上层对下层的隔层之间是否完全隔离，是否允许跨层的使用

层次化的特点

- 优点
  - 集中注意力处理单一层次的内部业务
  - 只和上下两层之间发生接口，易于标准化
  - 具有可扩展、可移植性
- 缺点
  - 可能会带来额外的通信成本
  - 错误传播复杂

* 分散化

* 将功能分散到不同的实现单元中，各个工作单元并行或者并发工作
* 目的：
  * 提升计算效率
  * 提高吞吐量
  * 提高容错性
* 分类
  * 有中央控制单元，形成层级控制体系
  * 无中央控制单元，各单元完全平权

分散化带来的挑战

- 可能出现并发异常，实现成本高
- 负载均衡
- 数据安全和隐私保护
- 数据不一致（副本、Cache）
- 存在集中环节

CAP理论

- 一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项
- 一致性：
  - 更新操作成功并返回客户端完成后，所有节点在同一时间的数据完全一致。
- 可用性：
  - 服务一直可用，而且是正常响应时间
- 分区容错性
  - 在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性和可用性的服务

CAP取舍

- CA：优先保证一致性和可用性，放弃分区容错。回到了集中式系统。适用场景：高可靠性业务，如银行
- CP：优先保证一致性和分区容错性，放弃可用性。当网络故障时不可用。适用场景：分布式存储、计算
- AP：优先保证可用性和分区容错性，放弃一致性。适用场景：普通Web应用

* 工程化

* 大规模软件开发是一个复杂工程
* 关注人在软件开发中的作用
* 软件质量与可维护性

* 拥抱变化
  * 系统的熵总是向最大化发展，如果没有外部负熵的话
  * 重构降低熵
  * 为未来的变化做好准备

*理解与预测

*软件的复杂性包含了空间和时间两个方面

*空间上：结构复杂性

  *Simple

  *Complicated

*时间上：可预测性

  *ORDERED

  *COMPLEX

  *CHAOTIC