软件体系结构的发展方向

* 各种软件体系结构语言之间的信息互换

1. 现有的各种软件体系结构描述语言大多是与领域相关的，所以不利于对不同领域体系结构的说明。但这些针对不同领域的软件体系结构描述语言在某些方面又大同小异，造成资源的冗余。
2. 其实，大多数软件体系结构描述语言具有一系列共同概念。
3. 如何用一种公共形式把各种语言综合起来，使得能够交换各种体系结构描述信息，将是今后软件体系结构研究和实践的重点之一。

* 设计工具和环境

1. 软件体系结构设计作为软件工程的一部分，它的计算机辅助实现手段是相当重要的。
2. 应当开发出一些软件工具来实现体系结构的描述和分析，开发阶段转换工具可以实现阶段成果的自动转换，例如，把需求规格说明自动转换为构件等。
3. 目前关于这方面的研究成果很少，特别是可以应用到实际项目开发中的工具和环境就更少。

* 体系结构再工程问题

1. 现在软件系统的规模变得越来越大，结构也越来越复杂，同时从头开始构建的大系统数量在急剧地减少，因而很多遗留系统正在被逐步地利用。
2. 从遗留系统软件代码和系统中抽取结构信息，经过描述、统一、抽象、一般化与实例化等处理，可总结出系统的体系结构。

## 软件体系结构的风格与模式

软件体系结构风格和模式的概念

* 软件体系结构风格Architectural Style

·一种体系结构风格以结构组织模式定义了一个系统家族

·关于构件和连接件类型的术语；一组约束对它们组合方式的规定；一个或多个语义模型，规定了如何从各成分的特性决定系统整体特性

·概括地说，一种软件体系结构风格刻划一个具有共享结构和语义的系统家族

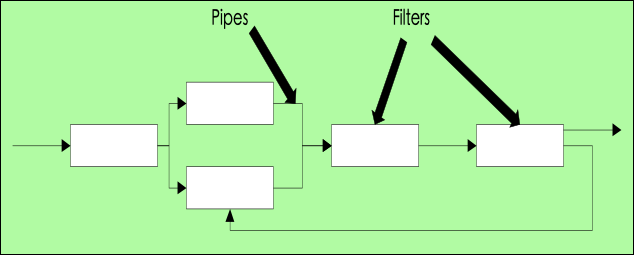
* 软件体系结构模式Architectural Pattern

·一种软件体系结构模式是对某个具体环境下问题的结构性解决方法

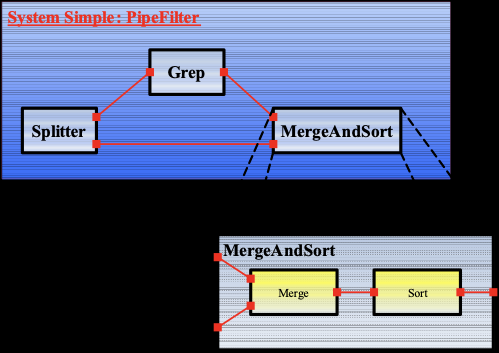
* 软件体系结构的构建风格  
  ：体系结构风格不是对软件进行分类的标准​它仅仅是表示描述软件的不同角度而已例如，一个系统采用了分层风格，但这并不妨碍它用面向对象的方法来实现。​同一个系统采用多种风格造成了所谓体系结构风格的异构组合。​

·管道-过滤器风格

* + 概述
    - 在管道-过滤器风格下，每个功能模块都有一组输入和输出
      * 功能模块称作过滤器（filters）
      * 功能模块间的连接可以看作输入、输出数据流之间的通路，所以称作管道（pipes）。
    - 管道-过滤器风格的特性之一在于过滤器的相对独立性，即过滤器独立完成自身功能，相互之间无需进行状态交互。
  + 过滤器是独立运行的构件
    - 非临近的过滤器之间不共享状态
  + 过滤器自身无状态
    - 过滤器对其处理上下连接的过滤器“无知”
  + 对相邻的过滤器不施加任何限制
    - 结果的正确性不依赖于各个过滤器运行的先后次序
    - 各过滤器在输入具备后完成自己的计算。
    - 完整的计算过程包含在过滤器之间的拓扑结构中。
* 一个管道-过滤器风格的图示



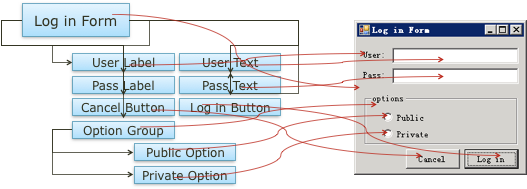
* 一个采用了嵌套的管道过滤器的系统示例



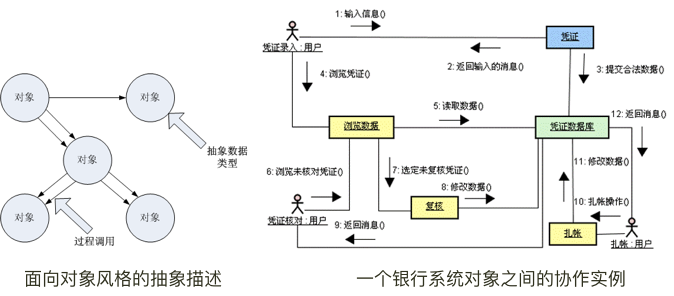
* 实例
  + Unix系统中的管道过滤器结构
    - ls –al | grep my
  + DOS 中的管道命令
    - DOS允许在命令中出现用竖线字符“|”分开的多个命令，将符号“|”之前的命令的输出，作为“|”之后命令的输入，这就是“管道功能”，竖线字符“|”是管道操作符。
    - 例如命令dir | more使得当前目录列表在屏幕上逐屏显示。dir的输出是整个目录列表，它不出现在屏幕上而是由于符号“|”的规定，成为下一个命令more的输入，more命令则将其输入，more命令则将其输入一屏一屏地显示，成为命令行的输出。
* 优点
  + 设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成
    - 设计人员将整个系统的输入输出行为理解为单个过滤器行为的叠加与组合。
    - 这样可以将问题分解，化繁为简。
    - 将系统抽象成一个“黑箱”，其输入是系统中第一个过滤器的输入管道，输出是系统中最后一个过滤器的输出管道，而其内部各功能模块的具体实现对用户完全透明。
  + 支持功能模块的复用
    - 任何两个过滤器，只要它们之间传送的数据遵守共同的规约，就可以相连接。
    - 每个过滤器都有自己独立的输入输出接口，如果过滤器间传输的数据遵守其规约，只要用管道将它们连接就可以正常工作。
  + 系统具有较强的可维护性和可扩展性
    - 旧的过滤器可以被替代，新的过滤器可以添加到已有的系统上。
    - 软件的易于维护和升级是衡量软件系统质量的重要指标之一，在管道-过滤器模型中，只要遵守输入输出数据规约，任何一个过滤器都可以被另一个新的过滤器代替，同时为增强程序功能，可以添加新的过滤器。
    - 这样，系统的可维护性和可升级性得到了保证。
  + 支持一些特定的分析
    - 如吞吐量计算和死锁检测等
    - 利用管道-过滤器风格的视图，可以很容易的得到系统的资源使用和请求的状态图。
    - 然后，根据操作系统原理等相关理论中的死锁检测方法就可以分析出系统目前所处的状态，是否存在死锁可能及如何消除死锁等问题。
  + 具有并发性
    - 每个过滤器作为一个单独的执行任务，可以与其它过滤器并发执行。
    - 过滤器的执行是独立的，不依赖于其它过滤器的。在实际运行时，可以将存在并发可能的多个过滤器看作多个并发的任务并行执行，从而大大提高系统的整体效率，加快处理速度。
* 缺点
  + 交互式处理能力弱
    - 管道-过滤器模型适于数据流的处理和变换，不适合为与用户交互频繁的系统建模。
    - 在这种模型中，每个过滤器都有自己的数据，这些数据或者是从磁盘存储器中读取来，或者是由另一个过滤器的输出导入进来，整个系统没有一个共享的数据区。
    - 这样，当用户要操作某一项数据时，要涉及到多个过滤器对相应数据的操作，其实现较为复杂。由以上的缺点，可以对每个过滤器增加相应的用户控制接口，使得外部可以对过滤器的执行进行控制。
  + 导致系统处理过程的成批操作。
    - 设计者也许不得不花费精力协调两个相对独立但又存在某种关系的数据流之间的关系
      * 例如多过滤器并发执行时数据流之间的同步问题等。
    - 根据实际设计的需要，设计者也需要对数据传输进行特定的处理（如为了防止数据泄漏而采取加密等手段），导致过滤器必须对输入、输出管道中的数据流进行解析或反解析，增加了过滤器具体实现的复杂性。

·面向对象风格

* 概述
  + 面相对象模式集数据抽象、抽象数据类型、类继承为一体，使软件工程公认的模块化、信息隐藏、抽象、重用性等原则在面向对象风格下得以充分实现。
* 应用场合
  + 面向对象的体系结构模式适用于数据和功能分离的系统中，同样也适合于问题域模型比较明显，或需要人机交互界面的系统。大多数应用事件驱动风格的系统也常常应用了面向对象风格
  + 图示



* 基本原则
* 将逻辑上的实体映射为对象，实体之间的关系映射为对象之间的应用关系。
* 对象利用应用关系来访问对方公开的接口，完成某个特定任务；一组对象之间相互协作，完成总体目标。
* 图示



* 优点
  + 高度模块性
    - 数据与其相关操作被组织为对象， 成为模块组织的基本单位
  + 封装功能
    - 一组功能和其实现细节被封装在一个对象中，具有功能的接口被暴露出来
  + 代码共享
    - 对象的相对独立性可被反复重用，通过拼装形成不同的软件系统
  + 灵活性
    - 对象在组织过程中，相互关系可以任意变化，只要接口兼容
  + 易维护性
    - 对象接近于人对问题和解决方案模型的思维方式，易于理解和修改
* 缺点
* 面向对象风格最大的不足在于如果一个对象需要调用另一个对象，它就必须知道那个对象的标识（对象名或对象引用），这样就无形之中增强了对象之间的依赖关系。
* 如果一个对象改变了自己的标识，就必须通知系统中所有和它有调用关系的对象，否则系统就无法正常运行。
* 图示

