【软件构造】第一章第一节 软件构造的多维视角

第一章第一节 软件构造的多维视图

从八个维度解释软件构造的对象是什么，并简单介绍在每个维度的刻画技术。

Outline

- 描述软件系统的三个维度元素、关系和各种视角的模式
  - 阶段：build-time && run-time
  - 动态：moment && period
  - 级别：code && component
- 不同视角间的转换

Notes

【软件的构成要素】

- 软件=算法+数据结构
- 软件=程序+数据+文档（80年代）
- 软件=Modules（Components）+Data/Control Flow 模块（组件）+数据流/控制流

【软件系统的组成】

Software system =

Programs（UI，算法，utilities（实用工具 function），APIs，test cases）

+ Data（files，database）

+ Documents（SRS（需求规格声明），SDD（设计规格声明），user manuals）

+ Users（谁来使用）

+ Business Objective（为什么使用它）

+ Social Environment（法规）

+ Technological Environement（如何部署）

+ Hardware / Network（硬件）

（前三个是主要）

【软件构造的多维视角】

1.阶段：构建‖运行

2.动态：时刻‖周期

3.级别：代码‖组件

	Moment		Period	
	Code-level	Component-level	Code-level	Component-level
Build-time	Source code, AST, Interface-Class-Attribute-Method (Class Diagram)	Package, Source File, Static Linking, Library, Test Case (Component Diagram) Build Script	Code Churn (代码变化)	Configuration Item, Version
Run-time	Code Snapshot, Memory dump	Package, Library, Dynamic linking, Configuration, Database, Middleware, Network, Hardware (Deployment Diagram)	Execution trace	Event log
			Procedure Call Graph, Message Graph (Sequence Diagram) Parallel and multi-threads/processes Distributed processes	

Buildtime概述

想法 -> 需求 -> 设计 -> 代码 -> 可安装可执行的包

- 代码是如何组建起来的？（依赖关系）
- 体系架构 源代码如何组成文件
- 时间角度 源代码在特定的时间什么样，随着时间如何变化

【Code-level；Build-time；Moment】

三种相互关联的形式

- 面向词法 半结构化源代码
- 面向语法 （AST抽象语法树）半结构化的源代码变成语法树（编译器能够处理）
- 面向语义 UML（参考软件工程课程内容）

关于词法、语法、语义的更多内容

https://www.cnblogs.com/lightsong/p/6107310.html

【Code-level；Build-time；Period——Code Churn（代码变化）】

- Churn Trends
- 代码变化包括添加、修改、删除

【Component-level ；Build-time；Moment】

- 源代码如何组织成文件——通过类库
- 文件被压缩进package，逻辑上进入components（组件）and sub-systems（子系统）
- 链接技术（动态 / 静态）

【类库（Library）】

来源

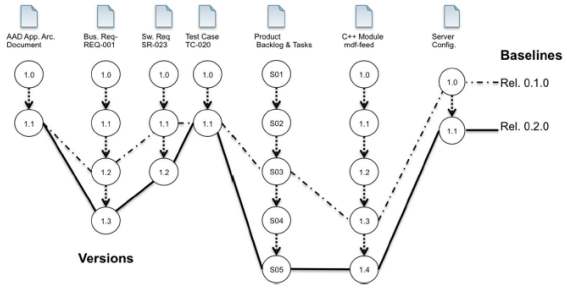
- 操作系统自带
- 语言自带的SDK
- 第三方
- 自己编写

链接到类库

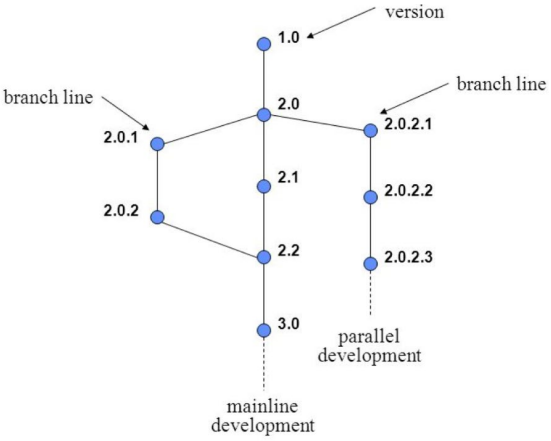
- 编译器形成关于外部库的链表，编译器找到库的目标文件，复制加到程序中

【Component-level；Build-time；Period】

- 版本控制（Git、SVN）



- 版本演化图（SCI）



- Software Configuration Item（软件生命周期各个阶段活动的产物，经审核后可称为软件配置项）
- version：major.minor.patch
- software evolution（软件演化）

Runtime概述

【运行时软件的高级概念】

- 可执行程序：CPU能直接理解执行的指令序列（二进制文件）
- 库文件：可复用的代码，库文件本身不能执行

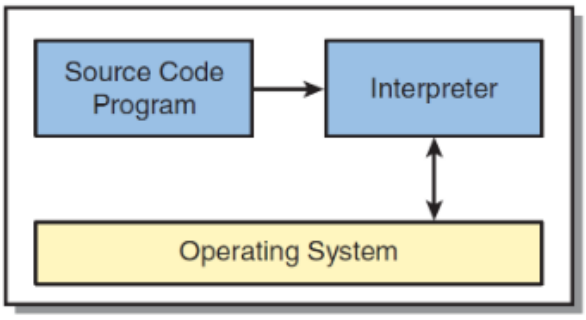
【可执行程序的四种形式】

【本地机器码】

- 载入内存——OS调用机器码
- 优点：CPU直接执行，速度快；
- 缺点：可移植性差；

【完全解释】

- Basic与UNIX中的shell
- 操作系统提供解析器，一边解析，一边运行

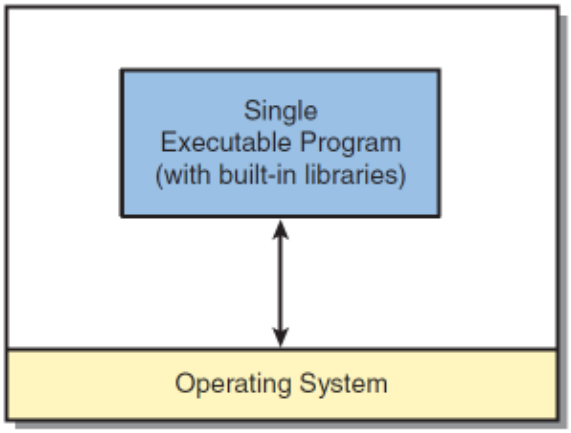


【自解码】

- 源代码编译为自解码，然后通过JVM变为机器码
- 或自解码通过解析器进行边解析边运行
- 优点：跨平台
- 缺点：速度慢

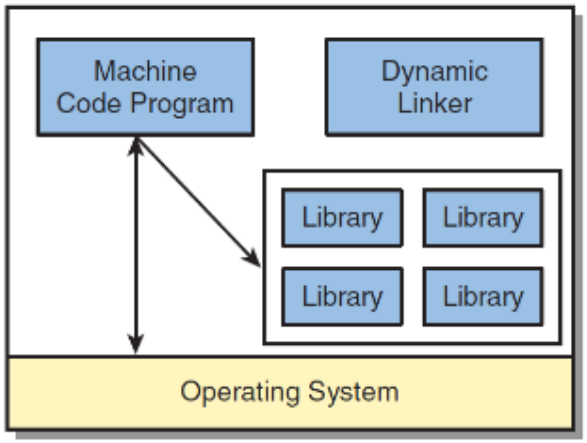
【静态链接】

- 类库就像是特别的对象文件的集合
- 编译前就需要知道方法对应的文件
- 构建时，从类库中提取文件并复制到可执行文件中



【动态链接】

- 操作系统为应用程序提供了丰富的函数调用，这些函数调用都包含在动态链接库中。在可执行文件装载时或运行时，由操作系统的装载程序加载库。
- 优点：多个程序可以共享同一个副本，减少内存占用；打包方便，方便库升级。



更多关于动态链接与静态链接

【配置文件和数据文件】

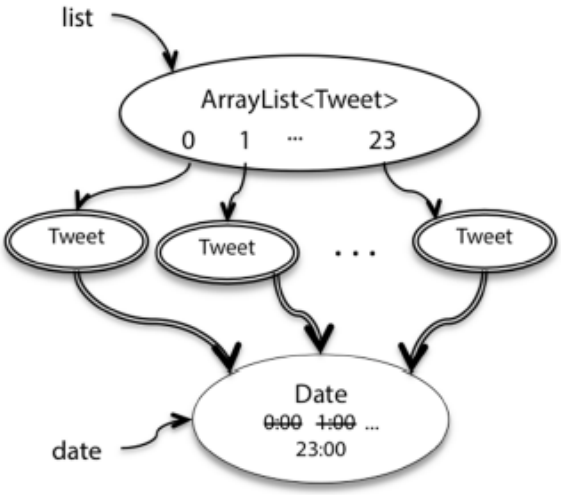
- 程序调用操作系统，来请求将数据读入内存；
- configuration：保存程序的参数
- Data：保存程序中如位图图形图像、数字化波形音频等文件

【分布式程序】

- 多端口或者多线程
- 如：QQ通过客户端访问服务器（client & server）
- 健壮性要求很高

【Code level；Run-time；Moment】

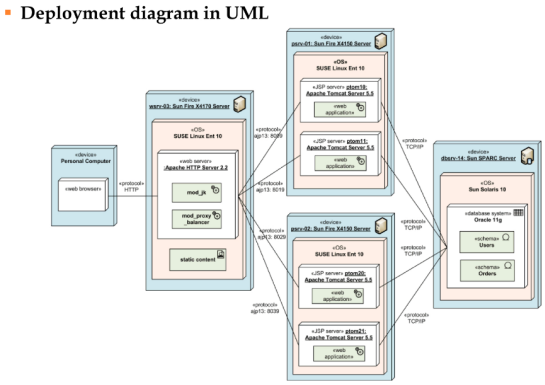
- 快照图：着眼于目标计算机内存中的变量级执行状态，体现某时刻内存中变量的情况。



- 内存转储（Memory dump）：常发生在异常退出时，把内存中信息写到文件中（常用来调试）

【Code level；Run-time；Period】

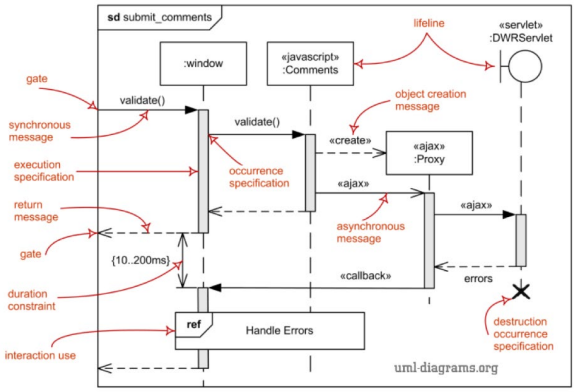
- UML时序图（类之间的段落关系）



- 执行跟踪：根据跟踪日志里的信息进行调试或诊断软件问题

【Component level；Run-time；Moment】

- UML部署图：程序中的各个模块在物理上如何分布；表明客户端、服务器之间的关系。



【Component level；Run-time；Period】

- 事件日志：每个事件有唯一编号
- 比较“执行跟踪”和“事件日志”

执行跟踪	事件日志
有开发人员使用	由系统管理员使用
记录低级信息（如抛出异常）	记录高级别信息（如程序的失败安装）
可以包含左述事件和信息	不能包含许多重复的事件或信息对其目标受众没有帮助
输出格式没有限制	需要基于标准的输入输出，有时是必须的
很少考虑本地化问题	事件日志常常是本地化的
一定是敏捷的	添加新类型的事件、消息不一定是敏捷的

【Transformations between views】

- 从无到有：
  - ADT/OOP
  - 可理解性
- 从代码变为组件：
  - Design
  - Build
- 构建阶段到运行阶段
  - inatall