【软件构造】第五章第一节 可复用性的度量、形态和外部观察

第五章第一节 可复用性的度量、形态和外部观察

本节探讨可复用软件的形态与特征，下一节学习“如何构造”。

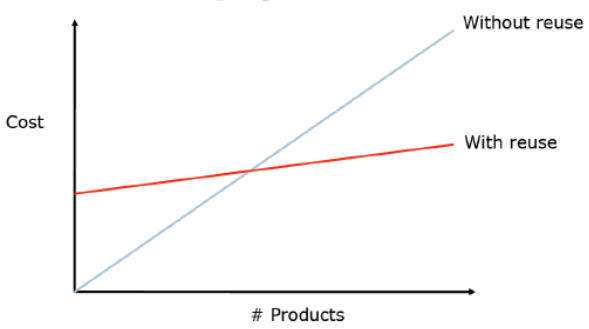
Outline

- 什么是软件复用
- 可复用实现的级别
  - 源代码级别的复用
  - 模块级别的复用：类、抽象类、接口
  - 库级别的复用：API、包
  - 系统级别的复用：框架
- 对可复用性的外部观察
- 白盒框架和黑盒框架

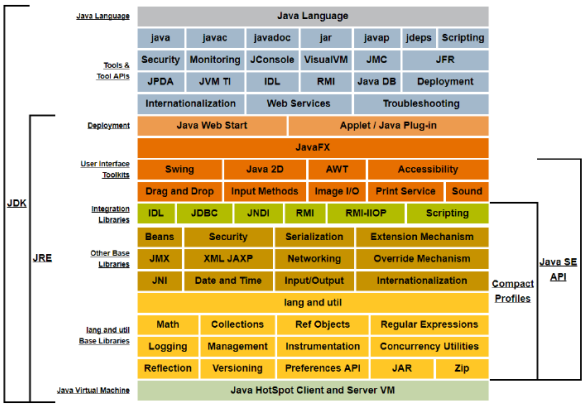
Notes

什么是软件复用

- 软件复用是使用现有软件组件实施或更新软件系统的过程。
- 软件复用的两个观点：
  - 面向复用编程(programming for reuse)：开发出可复用的软件
    - 开发成本高于一般软件的成本：要有足够高的适应性
    - 性能差些： 针对更普适场景，缺少足够的针对性
  - 基于复用编程(programming with reuse)：利用已有的可复用软件搭建应用系统
    - 可复用软件库，对其进行有效的管理
    - 往往无法拿来就用，需要适配
- 为什么需要复用：
  - 复用降低成本和开发时间
  - 复用的代码经过充分测试，可靠、稳定
  - 产出标准化，在不同应用中保持一致
- 软件复用的代价：
  - 软件可复用的部分需要设计在如下的标准上：明确的定义、开放的方法、简洁的交互规范、可理解的文档，并着眼于未来。
  - 不仅program for reuse代价高，program with reuse代价也高



- 代码复用的类型：
  - 白盒复用：源代码可见，可修改和扩展
    - 含义：复制已有代码到正在开发的系统，进行修改
    - 优点：可订制化程度高
    - 缺点：对其修改增加了软件的复杂度，且需要对其内部充分的了解
  - 黑盒服用：源代码不可见，不能修改
    - 含义：只能通过过API接口来使用，无法修改代码
    - 优点：清晰、简单
    - 缺点：适用性差
- 高复用性的软件应具有如下特性：
  - 小、简单
  - 与标准兼容
  - 灵活可变
  - 可扩展
  - 泛型、参数化
  - 模块化
  - 变化的局部性
  - 稳定
  - 丰富的文档和帮助
- 栗子：JDK中可复用的库和API



可复用实现的级别

【源代码级别的复用】

相关研究1：如何从互联网上快速找到需要的代码片段？

反向研究：如何从源代码中检测出克隆代码(clone code)？

我们可以在如github和searchcode的网站上搜索代码，实现复用。

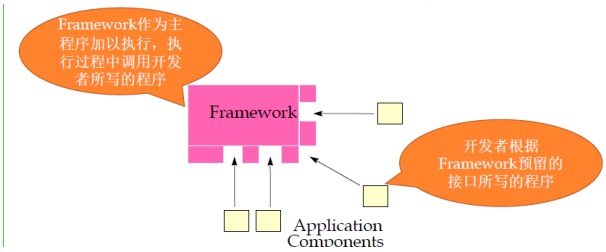
【模块级别的复用：类、抽象类、接口】

- 复用类：
  - 源码并非是必要的，可能只需要类文件或jar
  - 只需要将这个类加入到类路径
  - 可以使用工具javap获得一个类的public方法
- 使用复用类的注意事项：
  - 文档十分重要
  - 压缩会有助于复用
  - 管理更少的代码
  - 版本兼容性
  - 需要和类相关的包
- 复用类的方法：继承和委派
  - 继承（Inheritance）：
    - 类扩展了现有类的属性/行为；
    - 另外，他们可能会`Override`现有的行为；
    - 通常需要在实施之前设计继承层次结构；
  - 委派（Delegation）：
    - 根本没有父子关系的类中使用继承是不合理的，可以用委派的方式来代替。
    - 委托是简单的将一个对象连接到另一个对象上，使另一个对象获得这个对象方法的子集（一个实体将某个事物传递给另一个实体）。
    - 明确的委托：明确将需要传的对象传到目标对象上
    - 含蓄的委托：委托可以被描述为一种共享代码数据的低级别机制
    - 委派的类型：
      - Use（A uses B）
      - Composition/aggregation (A owns B)
      - Association (A has B)

【库级别的复用：**API**/包】

- 方法：Libaray、framework
  - library：
    - 库定义：一组提供可重用功能的类和方法（API）
    - 开发者构造可运行软件实体，其中涉及到对可复用库的调用
    - Java中有很多的库可以复用，例如Guava：Google的Java核心库；Apache Commons等
  - framework：
    - 框架定义：一组具体类、抽象类、及其之间的连接关系
    - 作为主程序加以执行，执行过程中调用开发者所写的程序
    - 开发者根据 framework的规约，填充自己的代码进去，形成完整系统

【系统级别的复用：框架】



将framework看作是更大规模的API复用，除了提供可复用的API，还将这 些模块之间的关系都确定下来，形成了整体应用的领域复用。开发者的任务就是增加新代码、对抽象类进行具体化。展开来说就是以下几点：

- 通常通过选择性覆盖来扩展框架; 或者程序员可以添加专门的用户代码来提供特定的功能—定义继承了抽象类祖先操作的具体类
- 设计模式(Hook方法)，它被应用程序覆盖以扩展框架。 Hook方法系统地将应用程序域的接口和行为与应用程序在特定上下文中所需的变体解耦。
- 控制反转，由第三方的容器来控制对象之间的依赖关系，而非传统实现中由代码直接操控。由第三方的容器来控制对象之间的依赖关系，而非传统实现中由代码直接操控。
- 不可修改的框架代码：在接受用户实现的扩展时，框架代码不应该被修改。 换句话说，用户可以扩展框架，但不应修改其代码。

对可复用性的外部观察

- Type Variation 类型可变
  - 能够复用的部分应该类型参数化，以适应不同的数据类型
  - 复用的部分应该一般化
  - 适应不同的类型，且满足LSP
- Implementation Variation 实现可变
  - ADT 有多种不同的实现，提供不同的representations 和abstract function ，但具有同样的specification (pre-condition, post-condition, invariants)，从而可以适应不同的应用场景
- Routine Grouping 功能分组
  - 提供完备的细粒度操作，保证功能的完整性，不同场景下复用不同的操作(及其组合)
- Representation Independence 表示独立
  - 内部实现可能会经常变化，但客户端不应受到影响。
- Factoring Out Common Behaviors 共性抽取
  - 将共同的行为（共性）抽象出来，形成可复用实体

白盒框架和黑盒框架

框架也可分为白盒框架和黑盒框架两类。

- 白盒框架：
  - 通过继承和动态绑定实现可扩展性。
  - 通过继承框架基类并重写预定义的hook方法来扩展现有功能。
  - 通常使用模板方法模式等设计模式来覆盖hook方法。
- 黑盒框架：
  - 通过为可插入框架的组件定义接口来实现可扩展性。
  - 通过定义符合特定接口的组件来复用现有功能。
  - 这些组件通过委派（Delegation）与框架集成。