Slow6502：设计模式报告（初稿）

Meow

软件学院

版本：0.10

更新：December 5, 2022

目录

1 项目简介

1.1 工作内容简介

Slow6502 是我们小组编写的一个 MOS Technologies 6502 微处理器和兼容系统的通用模拟器。Slow6502 可以模拟 1 MHz NMOS 6502 或 CMOS 65C02、32KB RAM、16KB ROM，并且可以模拟 MOS 6551 Motorola 6850 ACIA、MOS 6522 VIA 和 6545 CRTC 等外设，从而模拟一个完整的系统。

该项目由 Java 8 写成，实现了共 30 个设计模式，包括 23 个 GoF 设计模式和 7 个非 GoF 设计模式。

1.2 选题背景

关于 6502 MOS Technology 6502 是一款 8 位微处理器，由 Chuck Peddle 领导的 MOS Technology 团队设计. 设计团队曾在摩托罗拉从事摩托罗拉 6800 项目；6502 本质上是该设计的简化、更便宜和更快的版本。

在 1975 年推出时，6502 是市场上价格最低的微处理器，遥遥领先。它最初的售价不到 6800 或 Intel 8080 等大公司竞争设计成本的六分之一。它的推出导致整个处理器市场的价格迅速下降。它与 Zilog Z80 一起引发了一系列项目，导致了 80 年代初期的家用电脑革命。

1980 年代和 90 年代初期流行的视频游戏机和家用电脑，例如 Atari 2600、Atari 8 位系列、Apple II、Nintendo Entertainment System、Commodore 64、Atari Lynx、BBC Micro 等，使用 6502 或 6502 的变体基本设计。6502 推出后不久，MOS Technology 被 Commodore International 彻底收购，Commodore International 继续向其他制造商销售微处理器和许可。在 6502 的早期，它由 Rockwell 和 Synertek 二次采购，后来授权给其他公司。

时至今日，依然有诸多爱好者仿制 6502 和用它搭建怀旧电脑。即使和同时代的英特尔 8080, 摩托罗拉 6800 相比，它也是平平无奇的：3510 个晶体管，56 个指令，最高 3 MHz 的主频。看似平庸的 6502 依靠着仅有对手六分之一的价格、亲民的姿态、和车库文化交融的理念构成了美国 60 到 70 一代年轻计算机爱好者的集体记忆。

MOS 6502 和我国的计算机产业 得益于 MOS 6502 廉价的设计思路和良好的生态，其可能是第一个能广泛被中国学生们用到的 CPU。

著名的中华学习机 CEC-I 就“使用”了 MOS 6502 作为其 CPU。这款由电子工业部计算机与信息局组织，清华大学主持联合设计，有电子部六所、国营 734 厂、陕西省计算机厂以及华明计算机有限公司参与研制的机型是 Apple II 的仿制品。

这款机器是芯片级仿制的 Apple II 计算机，如果你拆开机器，会发现除了某些早期试验型号，里面并没有使用 MOS 6502 CPU。而是一些标着 ZH-6、ZH-7、ZH3065 的芯片，而这些芯片就是对 Apple II 内部芯片的完全仿制，包括 MOS 6502。在那个靠 decapping 技术就可以了解 CPU 内部构造的年代，这款机器的诞生虽然不能说是容易的，但也是很神奇了。而且这款机器内置了汉卡和 80 列卡。这两款卡在一般 Apple II 上是需要单独购买的，有了这两块

卡，不但可以显示和输入汉字，还可以显示更多的列数，让一行显示的字符数变多一倍，由于中文通常要占用两个字符宽度，这对于中文使用是非常重要的。

这款机器的价格大约是当时中产阶级家庭一年的收入，虽然并不能做到家用，但是很多学校都有一两台这个机器。至少让很多人从小接触到了计算机。

除了中华学习机外，上世纪末红遍大江南北的 FC 兼容机/小霸王学习机们和后来几乎人手一台的文曲星都采用了 6502 CPU。从诞生至今的 40 多年来，6502 对个人电脑和家用游戏主机行业产生了极其深刻的影响，无数人的人生因此而改变。虽然小霸王和文曲星早已经不再流行，各类 NES 上的游戏也逐渐被人遗忘，但直到现在，6502 仍被运用于数以亿计的工业监测和控制计算机当中，为我们服务。

2 设计模式详述

2.1 单例（Singleton）

模式某个类只能生成一个实例，该类提供了一个全局访问点供外部获取该实例，其拓展是有限多例模式。

2.2 原型（Prototype）

模式将一个对象作为原型，通过对其进行复制而克隆出多个和原型类似的新实例。

2.3 工厂方法（Factory Method）

模式定义一个用于创建产品的接口，由子类决定生产什么产品。

2.4 抽象工厂（Abstract Factory）

模式提供一个创建产品族的接口，其每个子类可以生产一系列相关的产品。

2.5 建造者（Builder）

模式将一个复杂对象分解成多个相对简单的部分，然后根据不同需要分别创建它们，最后构建成该复杂对象。

2.6 代理（Proxy）

模式为某对象提供一种代理以控制对该对象的访问。即客户端通过代理间接地访问该对象，从而限制、增强或修改该对象的一些特性。

2.7 适配器（Adapter）

模式将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类能一起工作。

2.8 桥接（Bridge）

模式将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现，从而降低了抽象和实现这两个可变维度的耦合度。

2.9 装饰（Decorator）

模式动态的给对象增加一些职责，即增加其额外的功能。

2.10 外观（Facade）

模式为多个复杂的子系统提供一个一致的接口，使这些子系统更加容易被访问。

2.11 享元（Flyweight）

模式运用共享技术来有效地支持大量细粒度对象的复用。

2.12 组合（Composite）

模式将对象组合成树状层次结构，使用户对单个对象和组合对象具有一致的访问性。

2.13 模板方法（TemplateMethod）

模式定义一个操作中的算法骨架，而将算法的一些步骤延迟到子类中，使得子类可以不改变该算法结构的情况下重定义该算法的某些特定步骤。

2.14 策略（Strategy）

模式定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的改变不会影响使用算法的客户。

2.15 命令（Command）

模式将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。

2.16 职责链（Chain of Responsibility）

模式把请求从链中的一个对象传到下一个对象，直到请求被响应为止。通过这种方式去除对象之间的耦合。

2.17 状态（State）

模式允许一个对象在其内部状态发生改变时改变其行为能力。

2.18 观察者（Observer）

模式多个对象间存在一对多关系，当一个对象发生改变时，把这种改变通知给其他多个对象，从而影响其他对象的行为。

2.19 中介者（Mediator）

模式定义一个中介对象来简化原有对象之间的交互关系，降低系统中对象间的耦合度，使原有对象之间不必相互了解。

2.20 迭代器（Iterator）

模式提供一种方法来顺序访问聚合对象中的一系列数据，而不暴露聚合对象的内部表示。

2.21 访问者（Visitor）

模式在不改变集合元素的前提下，为一个集合中的每个元素提供多种访问方式，即每个元素有多个访问者对象访问。

2.22 备忘录（Memento）

模式在不破坏封装性的前提下，获取并保存一个对象的内部状态，以便以后恢复它。

2.23 解释器（Interpreter）

模式提供如何定义语言的文法，以及对语言句子的解释方法，即解释器。

2.24 MVP 模式

2.25 数据访问对象模式（Data Access Object Pattern）

2.26 传输对象模式

2.27 业务代表模式

2.28 前端控制器模式

3 项目总结