设计模式完全手册

讲师：丁宋涛 夏曹俊

结构型模式

- 在解决了对象的创建问题之后，对象的组成以及对象之间的依赖关系就成了开发人员关注的焦点，因为如何设计对象的结构、继承和依赖关系会影响到后续程序的维护性、代码的健壮性、耦合性等。对象结构的设计很容易体现出设计人员水平的高低。
- 适配器模式属于结构型的设计模式，它是结构型设计模式之首（用的最多的结构型设计模式）。

结构型模式之适配器模式

- 学习适配器模式
- 一：适配器模式的介绍-定义、结构、参考实现、场景问题
- 二：适配器模式的典型疑问与优缺点评价
- 三：适配器模式的应用案例与思考

场景

- 适配器设计模式也并不复杂，适配器它是主要作用是将一个类的接口转换成客户希望的另外一个接口这样使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。适配器模式有两种：1.类的适配器 2.对象适配器，对象适配器更多一些。

适配器在实际开发中的应用案例-STL中的适配器

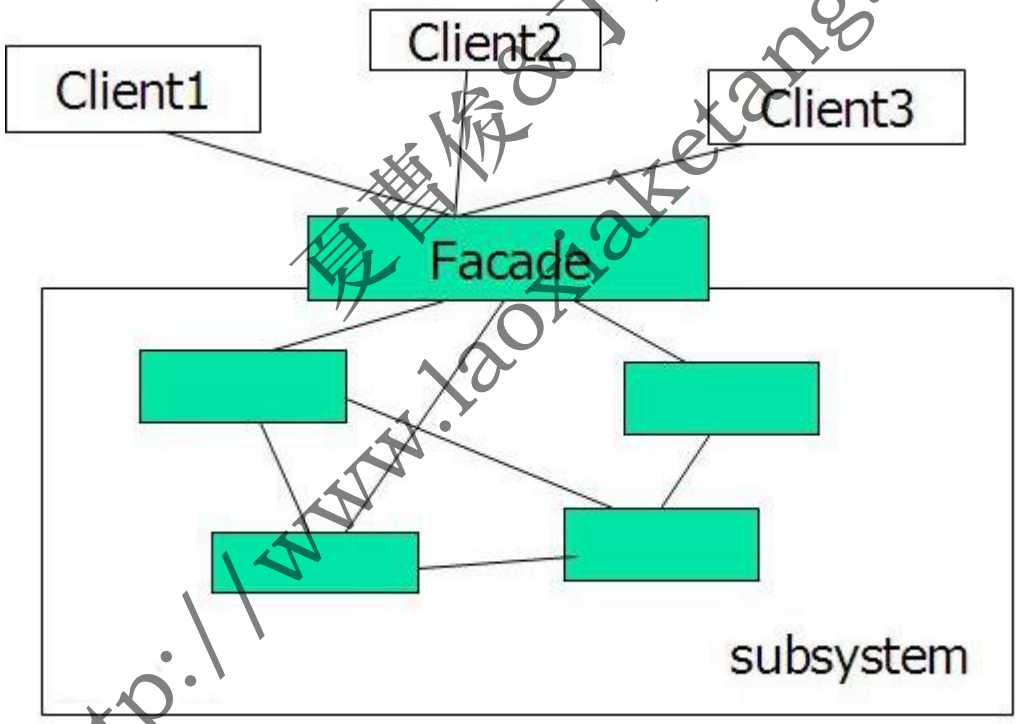
- 将容器放入栈或队列

结构型模式之门面模式

- 学习门面模式
- 一：门面模式的介绍-定义、结构、参考实现、场景问题
- 二：门面模式的典型疑问与优缺点评价
- 三：门面模式的应用案例与思考

场景

- 门面设计模式也并不复杂，指提供一个统一的接口去访问多个子系统的多个不同的接口，它为子系统中的一组接口提供一个统一的高层接口。使得子系统更容易使用。



结构型模式之门面模式

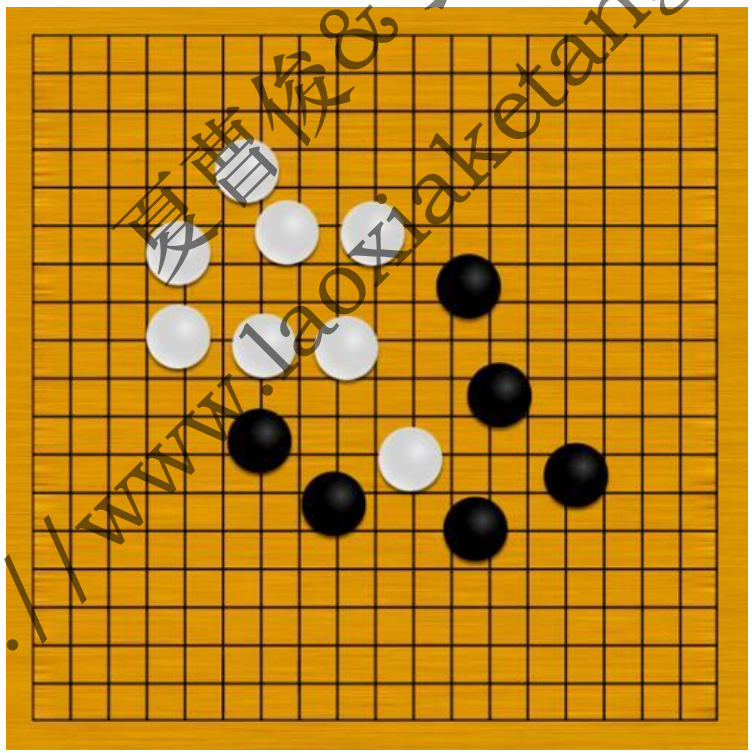
- LOD（迪米特法则）：
- 迪米特法则可以简单说成：**talk only to your immediate friends。** 对于OOD来说，又被解释为下面几种方式：一个软件实体应当尽可能少的与其他实体发生相互作用。每一个软件单位对其他的单位都只有最少的知识，而且局限于那些与本单位密切相关的软件单位。
- 迪米特法则的初衷在于降低类之间的耦合。由于每个类尽量减少对其他类的依赖，因此，很容易使得系统的功能模块功能独立，相互之间不存在（或很少有）依赖关系。
- 迪米特法则不希望类之间建立直接的联系。如果真的有需要建立联系，也希望能通过它的友元类来转达。因此，应用迪米特法则有可能造成的一个后果就是：系统中存在大量的中介类，这些类之所以存在完全是为了传递类之间的相互调用关系——这在一定程度上增加了系统的复杂度。

享元模式

- 学习享元模式
- 一：享元模式的介绍-定义、结构、参考实现、场景问题
- 二：享元模式的典型疑问与优缺点评价
- 三：享元模式的应用案例与思考

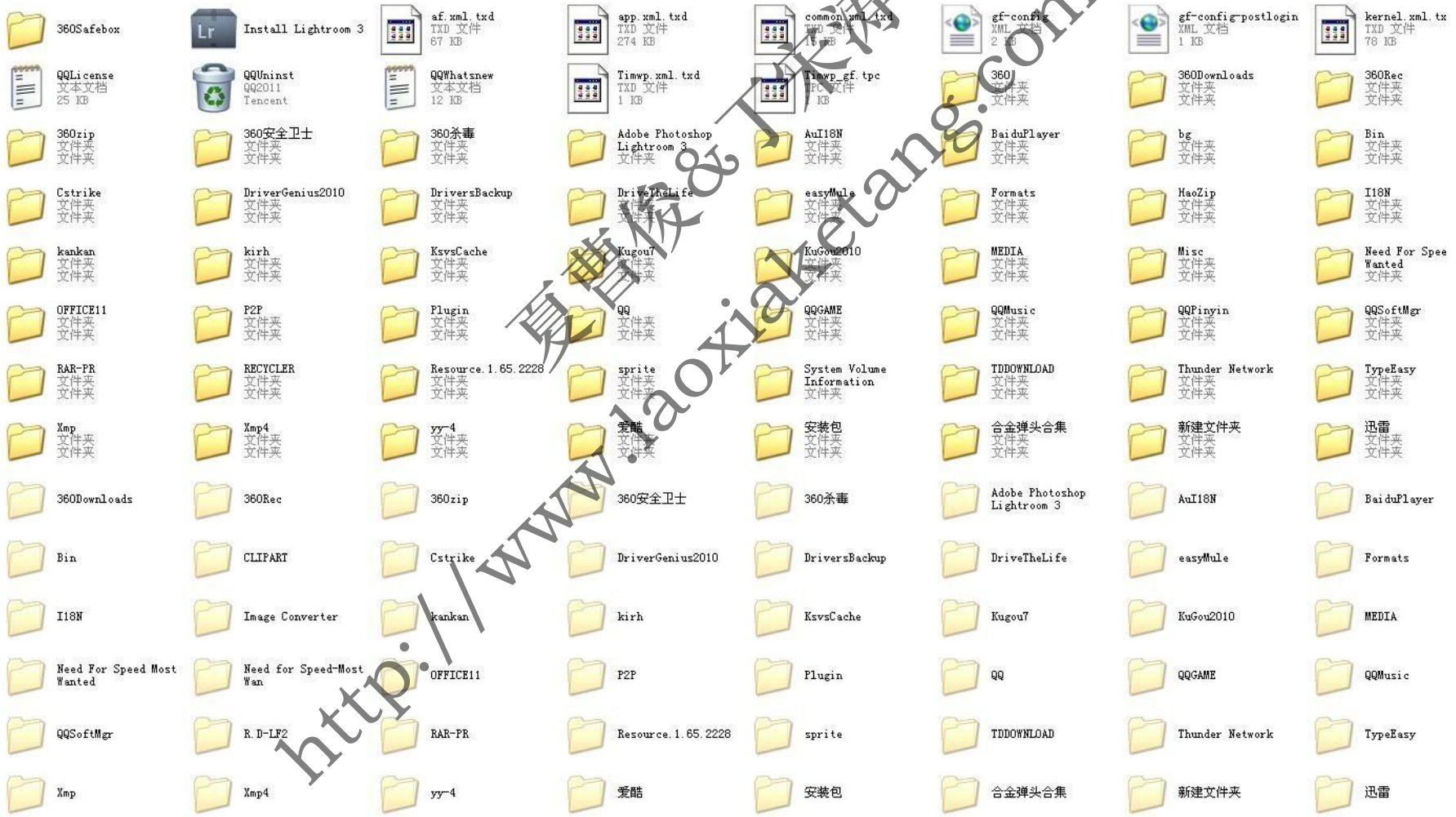
场景

- 考虑这样一个实际应用：五子棋游戏，五子棋盘是方形的，由纵横各15条线组成。15×15形成了225个交叉点。五子棋的棋子分为黑白两色，黑子113枚，白子112枚，黑白子加起来是225枚。



享元模式

- 运用共享技术有效地支持大量细粒度的对象 。



享元模式思想的应用-数据库连接池

- 1：数据库连接池：“数据库连接”是一种稀缺的资源，应该对其进行妥善管理。其实我们查询完数据库后，如果不关闭连接，而是暂时存放起来，当别人使用时，把这个连接给他们使用。就避免了一次建立数据库连接和断开的操作时间消耗。
- 数据库连接池的基本思想：就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。我们可以通过设定连接池最大连接数来防止系统无尽的与数据库连接
- 2：资源池、对象池：着重在对象的复用，池中的每个对象都是可替换的，从同一个池中获得A对象和B对象对客户端来讲是完全相同的，主要解决复用。字符串不变性

代理模式

- 学习代理模式
- 一：代理模式的介绍-定义、结构、参考实现、场景问题
- 二：代理模式的典型疑问与优缺点评价
- 三：代理模式的应用案例与思考

场景（软件分层思想与代理）

- 考虑这样一个场景：有一个日志读取需求?

如何设计

- 从系统加载的角度来说，懒加载可以提高系统资源的利用。这是一个典型的时间换空间的思想

代理模式定义

- **1：代理模式的功能**
- **代理模式是通过创建一个代理对象，用这个代理对象去代表真实的对象，客户端得到这个代理对象过后，对客户端没有什么影响，就跟得到了真实对象一样来使用。**

代理模式的本质-工程应用AOP与智能指针

包装模式（Decorate)

- 学习包装模式
- 一：包装模式的介绍-定义、结构、参考实现、场景问题
- 二：包装模式的典型疑问与优缺点评价
- 三：包装模式的应用案例与思考

场景

- 考虑这样一个实际应用：如何设计一个饮品的计费模块。

如何设计

- 组合大于继承，大量子类对象的类爆炸结构将无法完成设计。

Decorate模式

- **1：模式的功能**
- **透明的给一个对象增加功能，在给一个对象增加功能的同时，不能让这个对象知道，也就是不能去改动这个对象。而实现了能够给一个对象透明的增加功能，自然就能够实现功能的动态组合。**
- **。**

Decorate模式的工程应用——MFC中Cview与Java IO流

- 1：模式的功能
- 以一个CTextView为例，我们现在有一个文本视图类，该类提供一个基本的文本编辑框，当我们渴望拥有一个带滚动条的文本视图时，我们也许会从CTextView类中派生出一个CScrollTextVIew，而当我们想要一个带边框的文本视图时，我们也许会派生出一个CBorderTextView，可是如果我们想要一个既带滚动条，又拥有边框的文本视图呢？CScrollBorderTextView？也可以，但这并不灵活如果现在需求发生变动，我想要有一个边框，当文本过多时，它有垂直滚动条，当文本过长时，它有水平滚动条，当客户希望时，它又可以有阴影等效果，如果继续使用继承的方式来满足需求，那是非常痛苦的
- 2：Java IO流

组合模式（Composite）

- 学习组合模式
- 一： 组合模式的介绍-定义、结构、参考实现、场景问题
- 二：组合模式的典型疑问与优缺点评价
- 三：组合模式的应用案例与思考

场景

- 考虑这样一个实际应用：显示文件目录

如何设计

- 组合模式就是面向对象的树形结构，简单的讲就是递归结构的面向对象实现。

组合模式的应用

- **1：模式的功能**
- **组合模式的核心就是构建树形结构，一个重要的应用就是二叉树形式的计算表达式构建**

桥接模式（ bridge ）

- 学习桥接模式
- 一：桥接模式的介绍-定义、结构、参考实现、场景问题
- 二：桥接模式的典型疑问与优缺点评价
- 三：桥接模式的应用案例与思考

场景

- 考虑这样一个实际应用：考虑操作系统的组装

如何设计

- 类膨胀?

桥接模式的思想

- 现需要提供大中小3种型号的画笔，能够绘制5种不同颜色，如果使用蜡笔，我们需要准备3*5=15支蜡笔，也就是说必须准备15个具体的蜡笔类。而如果使用毛笔的话，只需要3种型号的毛笔，外加5个颜料盒，用3+5=8个类就可以实现15支蜡笔的功能。
- 实际上，蜡笔和毛笔的关键一个区别就在于笔和颜色是否能够分离。即将抽象化(Abstraction)与实现化(Implementation)脱耦，使得二者可以独立地变化"。关键就在于能否脱耦。蜡笔的颜色和蜡笔本身是分不开的，所以就造成必须使用15支色彩、大小各异的蜡笔来绘制图画。而毛笔与颜料能够很好的脱耦，各自独立变化，便简化了操作。在这里，抽象层面的概念是："毛笔用颜料作画"，而在实现时，毛笔有大中小三号，颜料有红绿蓝黑白等5种，于是便可出现3×5种组合。每个参与者（毛笔与颜料）都可以在自己的自由度上随意转换。
- 蜡笔由于无法将笔与颜色分离，造成笔与颜色两个自由度无法单独变化，使得只有创建15种对象才能完成任务。
- Bridge模式将继承关系转换为组合关系，从而降低了系统间的耦合，减少了代码编写量。

桥接模式的应用——MFC的Carchive与CFile

- **桥什么，接什么?**

调停者模式（Mediator）

- 学习调停者模式
- 一：调停者模式的介绍-定义、结构、参考实现、场景问题
- 二：调停者模式的典型疑问与优缺点评价
- 三：调停者模式的应用案例与思考

场景

- 考虑这样一个实际应用：电脑的组成与微软 Win95划时代的“即插即用”

调停者模式

- 1：模式的功能
- 调停者的功能非常简单，就是封装对象之间的交互。如果一个对象的操作
- 会引起其它相关对象的变化，或者是某个操作需要引起其它对象的后续或连带操
- 作，而这个对象又不希望自己来处理这些关系，那么就可以找调停者，把所有的
- 麻烦扔给它，只在需要的时候通知调停者，其它的就让调停者去处理就可以了。
- 反过来，其它的对象在操作的时候，可能会引起这个对象的变化，也可以
- 这么做。最后对象之间就完全分离了，谁都不直接跟其它对象交互，那么相互的
- 关系，全部被集中到调停者对象里面了，所有的对象就只是跟调停者对象进行通
- 信，相互之间不再有联系。
- 把所有对象之间的交互都封装在调停者当中，无形中还得到另外一个好
- 处，就是能够集中的控制这些对象的交互关系，这样有什么变化的时候，修改起
- 来就很方便。

调停者模式的应用-多对多关系的解耦合

状态模式（state）

- 学习状态模式
- 一：状态模式的介绍-定义、结构、参考实现、场景问题
- 二：状态模式的典型疑问与优缺点评价
- 三：状态模式的应用案例与思考

场景

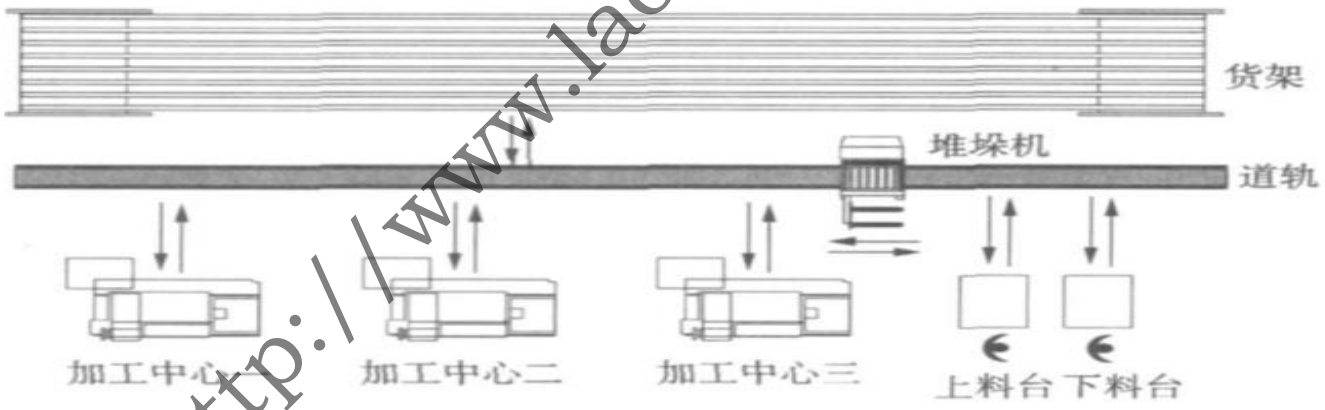
- 考虑这样一个实际应用：考虑一个在线下载的应用，要实现控制在1s内同一个用户只能单线程下载一个文件，如果一个用户反复下载，而且同时下载的并发数超过3，则判定为恶意下载，要取消该用户的资格。如果一个用户的下载个数超过5，将进入黑名单，禁止再登录本系统。”

状态模式

- **1：模式的功能**
- **状态模式的功能就是分离状态的行为，通过维护状态的变化，来调用不同的状态对应的不同的功能。**

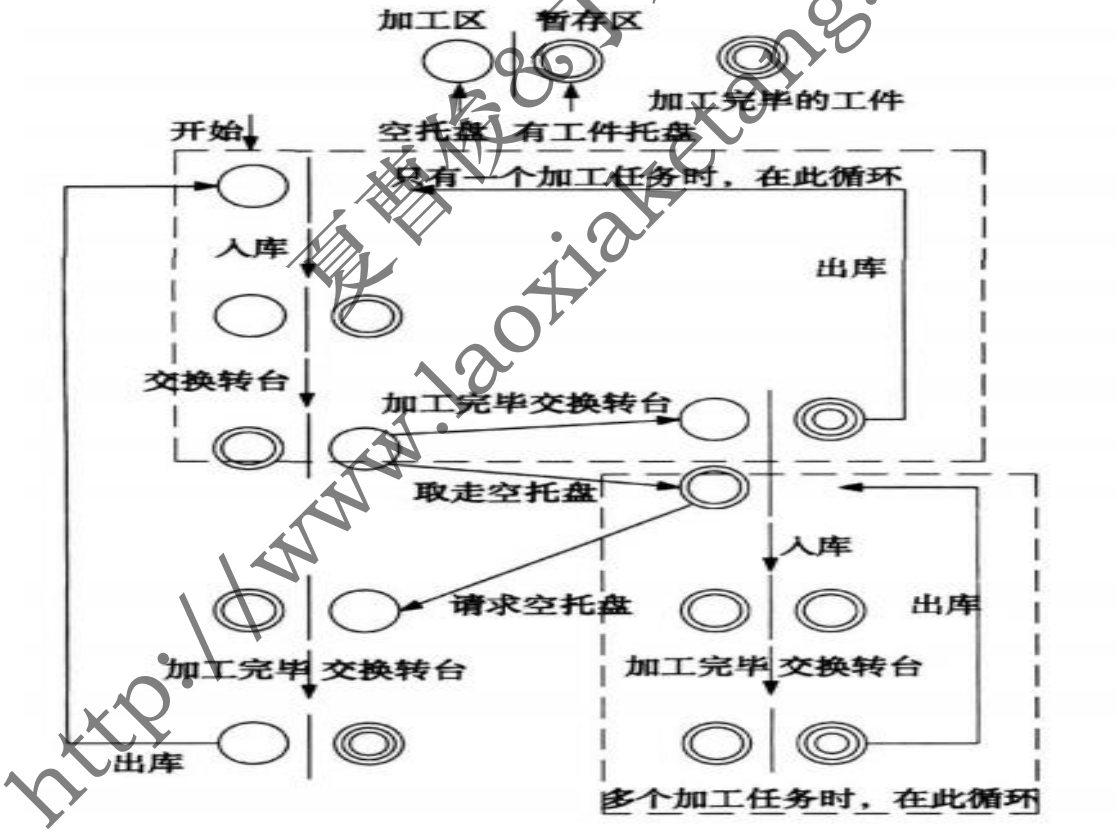
状态模式的应用-复杂if-else结构的解耦合

- 柔性生产中的if-else解耦加工中心共有 5个状态 :空闲 、急停 、故障 、关闭与运行 。其中运行状态只处理工件的正常加工及加工结束时的机床状态调整工作 ， 相应的物流动作发生在机床空闲状态 ,需要注意的是 ,这里的空闲状态只是定义的一个状态 ,实际机床可能在做一些动作



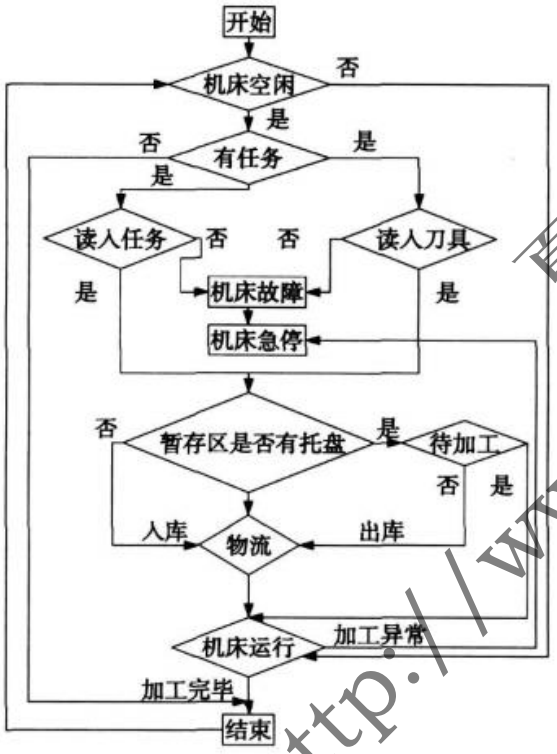
状态模式重构if-else结构

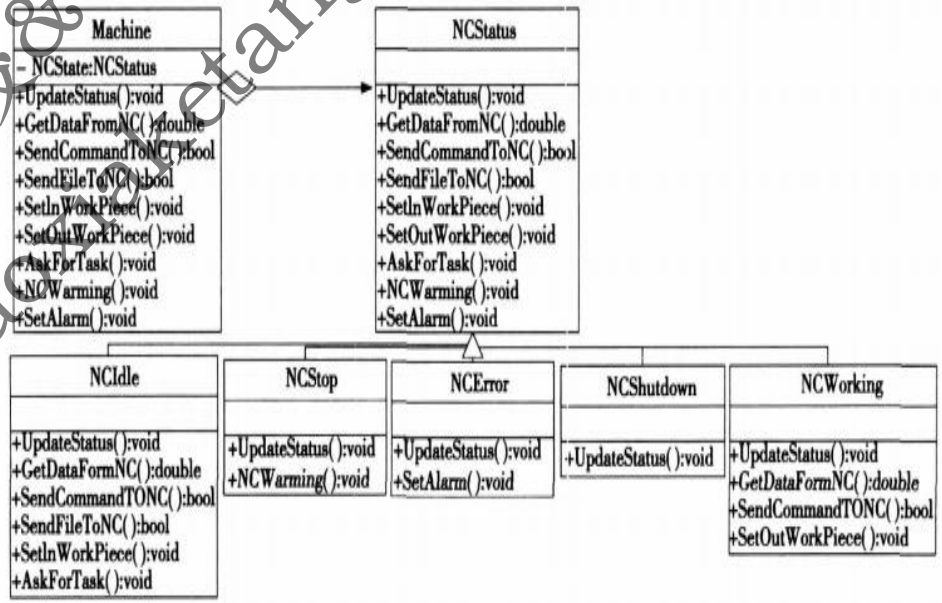
- **机床的行为取决于它的状态,在机床的运行时刻,物流的行为也是根据机床的状态做出改变,取走暂存区的空托盘或加工完毕的工件可以发生在机床加工区加工结束时或者是加工中,两者的改变带来了状态的变化,状态的变化依赖大量的分支判断来实现,在需求变化时~~~~**



状态模式的应用-复杂if-else结构的解耦合

- 加工流程





解释器模式（Interpretor）

- 学习解释器模式
- 一：解释器模式的介绍-定义、结构、参考实现、场景问题
- 二：解释器模式的典型疑问与优缺点评价
- 三：解释器模式的应用案例与思考

场景

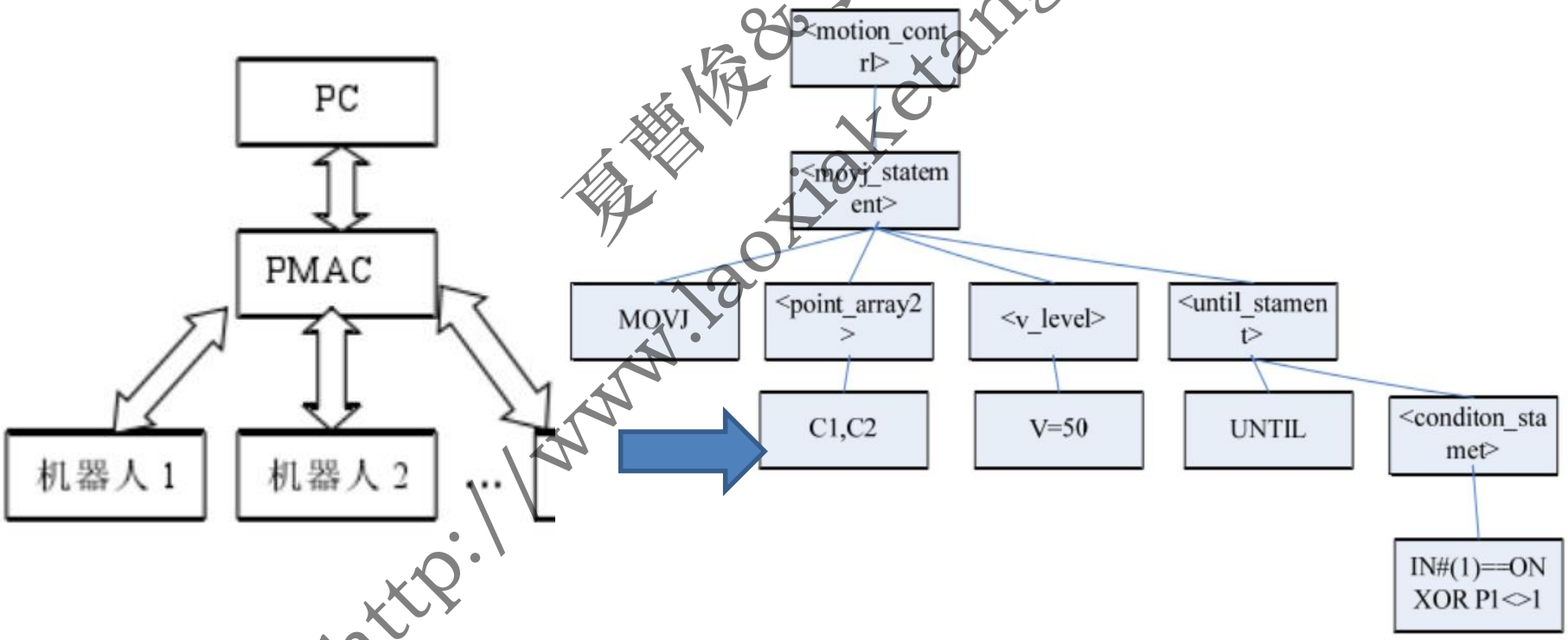
- 考虑这样一个应用：制作一个个位数加减法的计算表达式解析

解释器模式

- **1：模式的功能**
- **给定一个语言，定义它的文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释语言中的句子。**

解释器的应用案例-开放式工业机器人运动语言解释器

控制系统可以对不同的机器人进行伺服控制。可以用PMAC单独的控制一个机器人,也可以用PMAC同时对多个不同的机器人进行伺服控制，使其协同工作。这对于工业生产具有非常重大的意义: 对不同的工作,更换不同的机器人来完成工作；在工业自动化生产流水线上，可以让多台不同的机器人协同工作，完成不同的工作



访问者模式（Visitor）

- 学习访问者模式
- 一：访问者模式的介绍-定义、结构、参考实现、场景问题
- 二：访问者模式的典型疑问与优缺点评价
- 三：访问者模式的应用案例与思考

场景

- 考虑这样一个应用：假设我们有一系列的手机需要进行管理、测试，编写应用…

Visitor模式

- **1：模式的功能**
- **表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。**

Visitor的应用案例-工单派送系统

由于业务受理和工单施工两大业务流程在时间上是允许异步的,并且业务受理和工单施工在业务上应该是没有太多逻辑关联的。那么如何在不影响业务受理的情况下将业务受理与工单施工进行有效的衔接,这是系统设计需要考虑的首要问题。因为,对于业务受理来说,受理产生的受理单本身不知道对应的施工工单是什么样的格式,也不知道施工的流程,也不知道要拆分成如何样的施工工单同样对于施工受理方,并不关心业务受理的情况,只关心发送过来的工单格式。

所以为了要达到受理单和工单在不受“污染”的情况下各自工作,就要求系统的派单系统来处理这种转换。

此外,业务受理产生的受理单种类繁多,每种单对应的施工单的格式和消息都很不一样,并且对应的流转方式也不尽相同如一些审核的工单可以流转到系统的数据库表中等待审批、而一些购买信息则需要做远程接口实时发送到商平台。所以,工单派送子系统应该有为每种不同类别的受理单处理转换的能力。

随着业务的扩展越来越多的受理业务和管理部门的加入,施工过程也是个相对变化较大的模块,这样就要求施工过程能灵活的扩展,尽量达到在施工业务扩展的情况下不影响受理业务。从而使业务的扩展影响的范围缩小在最小范围内。

Visitor的应用案例-模块管理的双重分发