前言

相对学习UML的符号含义而言，掌握他们背后的方法和思想是更为重要的。古人将只是分为“技”和“道”，习技固然可以为人杰，而吾道才能羽化升仙。

UML创始人之一Grady Booch对面向对象的理解：我对面巷对象编程的牧鞭从来就不是复用／相反，对我来说，对象提供了一种处理复杂问题的方式，这个问题可以追述到亚里士多德，

您把这个世界视为过程还是对象？在面相对象兴起运动之前，编程以过程为中心，例如结构化设计方法，然而，系统已经到达了超越其处理能力的复杂性极点。有了对象，我们能够

通过提神抽象级别来构建更大的，更复杂的系统。我认为，这才是面向对象编程运动的真正胜利。

**面向过程**

将世界视为过程的这个方法本身蕴含者的一个前提假设，就是这个过程是稳定的，这样我们才有分析的基础。所有的过左成果都依赖于对这个过程的一步步分析。同事，这种步步分析的

过程分析方法还导致另一个结果，即过程中的每一步都是预设好的，有着严谨的因果关系。只可惜我们这个世界从来都不是一成不变的。尤其到了信息化时代，一切都无时无刻不在发生着

变化，系统所依赖的因果关系变得越来越脆弱。IBM的口号：On-Demand Business（随机应变的商务）

其实并非面向过程的方法不正确，只是因为构成一个系统的因素太多，要把所有可能的因素都考虑到，把所有因素的因果关系都分析清楚，再把这个过程模拟出来实在是太困难了。

我们的精力有限，计算能力有限，只能放弃对整个过程的了解。重新寻找一个方法，能够将复杂的系统转化成一个个我们可以控制的小单元。这个转化，正如完成一个成型的汽车

太过困难，我们可以将汽车分解为很多零件。分布制造。在一句预先设计好的接口把这些产品拼装起来。

temp

**面向对象**

面向对象方法将世界看作一个个相互独立的对象，相互之间并没有因果关系，他们平时是“鸡犬之声相闻，老死不相往来”。只有在某个外部力量的驱动下，对象之间才会依据某种规律

相互传递信息。这些交互构成了这个生动世界的一个“过程”。在没有外力作用的情况下，对象则保持者”静止”的状态。对于外界来说，对象内部就像一个黑匣子，这个就是封装。对象

可以聚合在一起形成一个新的对象。这个叫做聚合。对象可以繁育，产生的孩子拥有父辈的全部本领。这个叫做继承。对象可以是多面派，可以根据具体的要求。展现具体的能力，这个

就是接口。多个对象可能长着相同的脸，而这张脸背后却有着不同的行为。这个就是多态。对象是短视的，他不知道他自身处的整个世界是怎么回事儿。也不知道他的行为是如何贡献

给这个世界的。他只知道与他有着联系的身边的一群伙伴（依赖），并于伙伴间保持着信息交流的关系（耦合）。同事对象也是“自私”的，即使在伙伴之间，每个对象也依然顽固的保护

着自己的领地。只允许其他人通过小小的窗口（方法）进行交流，从不会像对象敞开心扉。

**面向对象的困难：**

对象是如何被抽离出来的，现实世界和对象世界看上去差别如此大，为什么要这么抽象而不那么抽象呢？（why）

对象世界极其灵活，我们怎么知道要采用那种组合来组合成需要的产品，什么样的组合是好的，什么样的组合是差的呢？（how）

对象世界这么难理解，怎样才能理解一个对象的真正含义（what）

箴言：（经验是宝贵的，但是很遗憾经验也是靠不住的，经验换一种说法，又叫拍脑袋）

现实世界和对象世界存在着一道鸿沟，这道鸿沟的名字叫做抽象，而抽象正是面向对象的根本所在。要跨越这个我们需要

1 一种把现实世界映射到对象世界的方法

2 一种从对象世界描述现实世界的方法

3 一种验证对象世界行为是否能反映现实世界的方法

而这个方法就是UML

**UML的优势：**

目前，随着软件工程的不断成熟，软件开发越来越朝着专业化和横向分工话发展。以前人们认为从需求到代码是一个紧密联系的过程，是不可分割的。一旦分开就会导致高成本和高技术风险

。然而与现代工业的分工越来越细枝和专业化的趋势一样，软件行业的需求，分析，设计，开发这些过程也被分离开来并专业化了。需求由专门的需求团队来做，甚至回委托给一个咨询公司，

分析由专门的系统分析团队来做。设计由专门的设计团队来做。以往，开发人员是项目的中心，一个开发人员常常从需求一直做大编码。现在，程序猿只负责根据设计结果来编码。设计师只负

责根据需求分析结果来做设计，项目组还有架构师，质量保证小组等等担任着格子的指责。在软件工程的约束下相互协作来完成一个项目。软件开发工作被横向分工化的一个最显著额例子就是

软件外包。承包商采集需求，设计团队进行设计，然后将编码工作外包给另一个公司完成。

＝》问题 ： 专业化分工导致沟通出现问题。而UML就是来规范统一交流语言的保证性语言。不同的人员都开的懂UML即可，不会产生理解误差。

**1 统一化。**

**2 可视化（图形）**

**从现实世界到业务模型**

**抽象现实世界的最高层 ：人 事 物 规则**

人是一切的中心，人要做事，做事儿就会使用到一些物，同事做事需要使用到一些规则。人驱动系统，事体现过程，物存储结果，规则是控制。

建立模型的关键，就是要弄明白有什么人，什么人做什么事儿，什么事儿产生什么物，中间有什么规则。再把人，事，物之间的关系定义出来。一个模型就基本成型了。

对应的UML提供的元素：

参与者：



UML采用称之为参与者的元模型作为信息来源提供者。代表着现实世界的人。（核心）



第二个UML采用被称之为用力的一种元模型来表示驱动者的业务目标。也就是参与者想要做什么并且获得什么。 也就是现实中的事

**从业务模型到概念模型**

UML通过被称为为概念化的过程，来建立适合计算机理解和实现的模型，这个模型称为分析模型。分析模型介于原始需求和计算机实现之间，是一种过渡模型。分析模型向上映射了原始需求。计算机的可执行代码可以通过分析模型追述到原始需求。同时，分析模型乡下为计算机实现规定的一种高层次的抽象。这种抽象是一种指导。也是一种约束。

**分析模型用到的主要元模型：**



参与者与系统的交互，系统与系统的交互，模块与模块的交互。

完成业务目标所涉及的事物。



边界类和实体都是静态的，本身并不会动作。边界类与边界类之间，实体类实体类之间，都不能直接访问，都需要实体类来代理访问。这样就把动作和物体分开了。

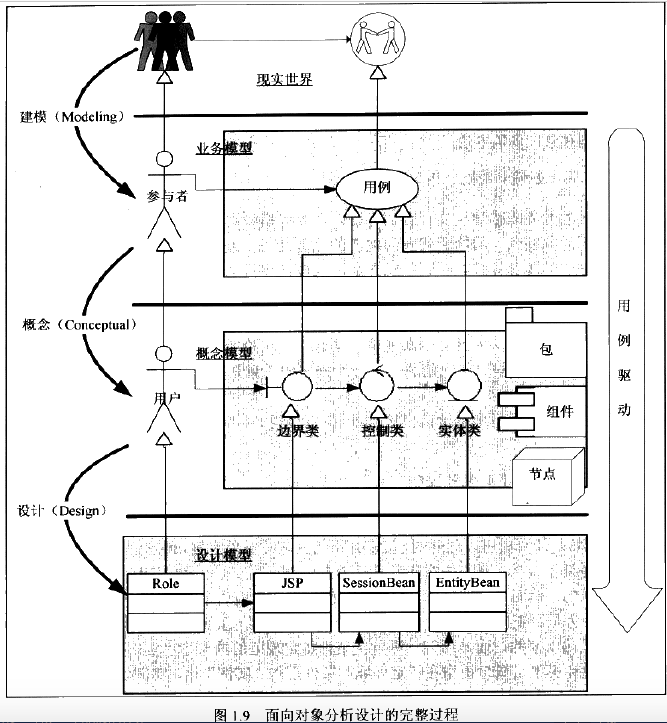
从概念模型到设计模型

设计模型的工作就是建造零部件，组装汽车的过程。在大多数情况下，实现类可以简单的从分析类映射出来。在设计模型中，概念模型中的边界类可以被转化为操作界面或者系统接口。控制类可以被转化为计算程序或者控制程序。例如工作六，算法体等。实体类可以转化为数据库表，xml文档，或者其他具有持久性特征的类。这个过程是有规则可循的：

软件架构和框架

编程语言

规范或者中间件



**建模**

建模，是指通过对客观事物建立一种抽象的方法用以表征事物并获得对事物本身的理解。同时把这种理解概念化。将这些逻辑概念组织起来。构成一种对所观察的对象的内部结构和工作原理的便于理解的表达。

当你试图为现实世界建模的时候，首先要决定的事抽象角度，即建立这个模型的目的是什么。一旦抽象角度确定，剩下的事情就变得顺理成章，而不再是杂乱无章。

不论再需求分析，系统分析，还是系统设计上，一定要学会采用面相对象的方法，首先不要决定去通盘考虑，而是找出问题领域里包含的抽象角度。如果你把抽象角度找全了，问题领域也就解决了。

做需求的时候，首要目标不是要弄清楚业务是如何一步步完成的。而是要弄清楚有多少业务的参与者。每个参与者的目标是什么。参与者的目标就是你的抽象角度。与一个复杂的业务流程相比，单独的分析参与者的一个个目的要简单的多。实际上，这就是用例。这也就是为什么用例会成为业务建模的方法之一。

模就是人，事，物，规则

找准抽象层次

by the way ：How Locking Works

A READ lock does not prevent other READers from also acquiring a READ lock and reading. A READ lock cannot be obtained if there is an outstanding WRITE lock. It will queue. A WRITE lock cannot be obtained while there are outstanding READ locks. It will queue. In each case the lock should be released after use to avoid locking problems. The lock release should be in a "finally" block. If before each read you acquire a READ lock and then before each write you acquire a WRITE lock, then an isolation level akin to READ\_COMMITTED is achieved.

有数据的地方就涉及到事物和锁对数据的处理

用例粒度与抽象级别要一致。