04讲理论一：当谈论面向对象的时候，我们到底在谈论什么

考虑到各个水平层次的同学，并且保证专栏内容的系统性、全面性，我会循序渐进地讲解跟设计模式相关的所有内容。所以，专栏正文的第一个模块，我会讲一些设计原则、设计思想，比如，面向对象设计思想、经典设计原则以及重构相关的知识，为之后学习设计模式做铺垫。

在第一个模块中，我们又首先会讲到面向对象相关的理论知识。提到面向对象，我相信很多人都不陌生，随口都可以说出面向对象的四大特性：封装、抽象、继承、多态。实际上，面向对象这个概念包含的内容还不止这些。所以，今天我打算花一节课的时间，先大概跟你聊一下，当我们谈论面向对象的时候，经常会谈到的一些概念和知识点，为学习后面的几节更加细化的内容做一个铺垫。

特别说明一下，对于今天讲到的概念和知识点，大部分我都是点到为止，并没有展开详细讲解。如果你看了之后，对某个概念和知识点还不是很清楚，那也没有关系。在后面的几节课中，我会花更多的篇幅，对今天讲到的每个概念和知识点，结合具体的例子，一一做详细的讲解。

什么是面向对象编程和面向对象编程语言？

面向对象编程的英文缩写是OOP，全称是Object Oriented Programming。对应地，面向对象编程语言的英文缩写是OOPL，全称是Object Oriented Programming Language。

面向对象编程中有两个非常重要、非常基础的概念，那就是类（class）和对象（object）。这两个概念最早出现在1960年，在Simula这种编程语言中第一次使用。而面向对象编程这个概念第一次被使用是在Smalltalk这种编程语言中。Smalltalk被认为是第一个真正意义上的面向对象编程语言。

1980年左右，C++的出现，带动了面向对象编程的流行，也使得面向对象编程被越来越多的人认可。直到今天，如果不按照严格的定义来说，大部分编程语言都是面向对象编程语言，比如Java、C++、Go、Python、C#、Ruby、JavaScript、Objective-C、Scala、PHP、Perl等等。除此之外，大部分程序员在开发项目的时候，都是基于面向对象编程语言进行的面向对象编程。

以上是面向对象编程的大概发展历史。在刚刚的描述中，我着重提到了两个概念，面向对象编程和面向对象编程语言。那究竟什么是面向对象编程？什么语言才算是面向对象编程语言呢？如果非得给出一个定义的话，我觉得可以用下面两句话来概括。

- 面向对象编程是一种编程范式或编程风格。它以类或对象作为组织代码的基本单元，并将封装、抽象、继承、多态四个特性，作为代码设计和实现的基石 。
- 面向对象编程语言是支持类或对象的语法机制，并有现成的语法机制，能方便地实现面向对象编程四大特性（封装、抽象、继承、多态）的编程语言。

一般来讲， 面向对象编程都是通过使用面向对象编程语言来进行的，但是，不用面向对象编程语言，我们照样可以进行面向对象编程。反过来讲，即便我们使用面向对象编程语言，写出来的代码也不一定是面向对象编程风格的，也有可能是面向过程编程风格的。这里听起来是不是有点绕？不过没关系，我们在后面的第7节课中，会详细讲解这个问题。

除此之外，从定义中，我们还可以发现，理解面向对象编程及面向对象编程语言两个概念，其中最关键的一点就是理解面向对象编程的四大特性。这四大特性分别是：封装、抽象、继承、多态。不过，关于面向对象编程的特性，也有另外一种说法，那就是只包含三大特性：封装、继承、多态，不包含抽象。为什么会有这种分歧呢？抽象为什么可以排除在面向对象编程特性之外呢？关于这个问题，在下一节课详细讲解这四大特性的时候，我还会再拿出来说一下。不过，话说回来，实际上，我们没必要纠结到底是四大特性还是三大特性，关键还是理解每种特性讲的是什么内容、存在的意义以及能解决什么问题。

而且，在技术圈里，封装、抽象、继承、多态也并不是固定地被叫作“四大特性”（features），也有人称它们为面向对象编程的四大概念（concepts）、四大基石（cornerstones）、四大基础（fundamentals）、四大支柱（pillars）等等。你也发现了吧，叫法挺混乱的。不过，叫什么并不重要。我们只需要知道，这是前人进行面向对象编程过程中总结出来的、能让我们更容易地实现各种设计思路的几个编程套路，这就够了。在之后的课程讲解中，我统一把它们叫作“四大特性”。

如何判定某编程语言是否是面向对象编程语言？

如果你足够细心，你可能已经留意到，在我刚刚的讲解中，我提到，“如果不按照严格的定义来说，大部分编程语言都是面向对象编程语言”。为什么要加上“如果不按照严格的定义”这个前提呢？那是因为，如果按照刚刚我们给出的严格的面向对象编程语言的定义，前面提到的有些编程语言，并不是严格意义上的面向对象编程语言，比如JavaScript，它不支持封装和继承特性，按照严格的定义，它不算是面向对象编程语言，但在某种意义上，它又可以算得上是一种面向对象编程语言。我为什么这么说呢？到底该如何判断一个编程语言是否是面向对象编程语言呢？

还记得我们前面给出的面向对象编程及面向对象编程语言的定义吗？如果忘记了，你可以先翻到上面回顾一下。不过，我必须坦诚告诉你，那个定义是我自己给出的。实际上，对于什么是面向对象编程、什么是面向对象编程语言，并没有一个官方的、统一的定义。而且，从1960年，也就是60年前面向对象编程诞生开始，这两个概念就在不停地演化，所以，也无法给出一个明确的定义，也没有必要给出一个明确定义。

实际上，面向对象编程从字面上，按照最简单、最原始的方式来理解，就是将对象或类作为代码组织的基本单元，来进行编程的一种编程范式或者编程风格，并不一定需要封装、抽象、继承、多态这四大特性的支持。但是，在进行面向对象编程的过程中，人们不停地总结发现，有了这四大特性，我们就能更容易地实现各种面向对象的代码设计思路。

比如，我们在面向对象编程的过程中，经常会遇到is-a这种类关系（比如狗是一种动物），而继承这个特性就能很好地支持这种is-a的代码设计思路，并且解决代码复用的问题，所以，继承就成了面向对象编程的四大特性之一。但是随着编程语言的不断迭代、演化，人们发现继承这种特性容易造成层次不清、代码混乱，所以，很多编程语言在设计的时候就开始摒弃继承特性，比如Go语言。但是，我们并不能因为它摒弃了继承特性，就一刀切地认为它不是面向对象编程语言了。

实际上，我个人觉得，只要某种编程语言支持类或对象的语法概念，并且以此作为组织代码的基本单元，那就可以被粗略地认为它就是面向对象编程语言了。至于是否有现成的语法机制，完全地支持了面向对象编程的四大特性、是否对四大特性有所取舍和优化，可以不作为判定的标准。基于此，我们才有了前面的说法，**按照严格的定义，很多语言都不能算得上面向对象编程**

语言，但按照不严格的定义来讲，现在流行的大部分编程语言都是面向对象编程语言。

所以，多说一句，关于这个问题，我们一定不要过于学院派，非要给面向对象编程、面向对象编程语言下个死定义，非得对某种语言是否是面向对象编程语言争个一清二白，这样做意义不大。

什么是面向对象分析和面向对象设计?

前面我们讲了面向对象编程（OOP），实际上，跟面向对象编程经常放到一块儿来讲的还有另外两个概念，那就是面向对象分析（OOA）和面向对象设计（OOD）。面向对象分析英文缩写是OOA，全称是Object Oriented Analysis；面向对象设计的英文缩写是OOD，全称是Object Oriented Design。OOA、OOD、OOP三个连在一起就是面向对象分析、设计、编程（实现），正好是面向对象软件开发要经历的三个阶段。

关于什么是面向对象编程，我们前面已经讲过了。我们现在再来讲一下，什么是面向对象分析和设计。这两个概念相对来说要简单一些。面向对象分析与设计中的“分析”和“设计”这两个词，我们完全可以从字面上去理解，不需要过度解读，简单类比软件开发中的需求分析、系统设计即可。不过，你可能会说，那为啥前面还加了个修饰词“面向对象”呢？有什么特殊的意义吗?

之所以在前面加“面向对象”这几个字，是因为我们是围绕着对象或类来做需求分析和设计的。分析和设计两个阶段最终的产出是类的设计，包括程序被拆解为哪些类，每个类有哪些属性方法，类与类之间如何交互等等。它们比其他的分析和设计更加具体、更加落地、更加贴近编码，更能够顺利地过渡到面向对象编程环节。这也是面向对象分析和设计，与其他分析和设计最大的不同点。

看到这里，你可能会问，那面向对象分析、设计、编程到底都负责做哪些工作呢？简单点讲，面向对象分析就是要搞清楚做什么，面向对象设计就是要搞清楚怎么做，面向对象编程就是将分析和设计的的结果翻译成代码的过程。今天，我们只是简单介绍一下概念，不展开详细讲解。在后面的面向对象实战环节中，我会用两节课的时间，通过一个实际例子，详细讲解如何进行面向对象分析、设计和编程。

什么是UML？我们是否需要UML?

讲到面向对象分析、设计、编程，我们就不得不提到另外一个概念，那就是UML（Unified Model Language），统一建模语言。很多讲解面向对象或设计模式的书籍，常用它来画图表达面向对象或设计模式的设计思路。

实际上，UML是一种非常复杂的东西。它不仅仅包含我们常提到类图，还有用例图、顺序图、活动图、状态图、组件图等。在我看来，即便仅仅使用类图，学习成本也是很高的。就单说类之间的关系，UML就定义了很多种，比如泛化、实现、关联、聚合、组合、依赖等。

要想完全掌握，并且熟练运用这些类之间的关系，来画UML类图，肯定要花很多的学习精力。而且，UML作为一种沟通工具，即便你能完全按照UML规范来画类图，可对于不熟悉的人来说，看懂的成本也还是很高的。

所以，从我的开发经验来说，UML在互联网公司的项目开发中，用处可能并不大。为了文档化软件设计或者方便讨论软件设计，大部分情况下，我们随手画个不那么规范的草图，能够达意，方便沟通就够了，而完全按照UML规范来将草图标准化，所付出的代价是不值得的。

所以，我这里特别说明一下，专栏中的很多类图我并没有完全遵守UML的规范标准。为了兼顾图的表达能力和你的学习成本，我对UML类图规范做了简化，并配上了详细的文字解释，力图让你一眼就能看懂，而非适得其反，让图加重你的学习成本。毕竟，我们的专栏并不是一个讲方法论的教程，专栏中的所有类图，本质是让你更清晰地理解设计。

重点回顾

今天的内容讲完了，我们来一起总结回顾一下，你需要重点掌握的几个概念和知识点。

1.什么是面向对象编程?

面向对象编程是一种编程范式或编程风格。它以类或对象作为组织代码的基本单元，并将封装、抽象、继承、多态四个特性，作为代码设计和实现的基石 。

2.什么是面向对象编程语言?

面向对象编程语言是支持类或对象的语法机制，并有现成的语法机制，能方便地实现面向对象编程四大特性（封装、抽象、继承、多态）的编程语言。

3.如何判定一个编程语言是否是面向对象编程语言?

如果按照严格的的定义，需要有现成的语法支持类、对象、四大特性才能叫作面向对象编程语言。如果放宽要求的话，只要某种编程语言支持类、对象语法机制，那基本上就可以说这种编程语言是面向对象编程语言了，不一定非得要求具有所有的四大特性。

4.面向对象编程和面向对象编程语言之间有何关系?

面向对象编程一般使用面向对象编程语言来进行，但是，不用面向对象编程语言，我们照样可以进行面向对象编程。反过来讲，即便我们使用面向对象编程语言，写出来的代码也不一定是面向对象编程风格的，也有可能是面向过程编程风格的。

5.什么是面向对象分析和面向对象设计?

简单点讲，面向对象分析就是要搞清楚做什么，面向对象设计就是要搞清楚怎么做。两个阶段最终的产出是类的设计，包括程序被拆解为哪些类，每个类有哪些属性方法、类与类之间如何交互等等。

课堂讨论

今天我们要讨论的话题有两个：

1. 在文章中，我讲到UML的学习成本很高，沟通成本也不低，不推荐在面向对象分析、设计的过程中使用，对此你有何看法?
2. 有关面向对象的概念和知识点，除了我们今天讲到的，你还能想到其他哪些吗?

欢迎在留言区发表你的观点，积极参与讨论。你也可以把这篇文章分享给你的朋友，邀请他一起学习。

精选留言

王争

在这篇文章中，“面向对象编程”一词多义，不同的场景、语境下，解释不同。文章中没有点到这一点，我这里稍微补充说明一下：

1. 文章前半部分，面向对象编程指的是一种编程风格或者范式。

2. 文章后半部分，在讲到面向对象分析、设计、编程的时候，面向对象编程是一种行为。

2019-11-11 06:13

王争

UML中定义了类之间的关系：泛化、实现、关联、聚合、组合、依赖，试问下小伙伴们，你们都能搞清楚这几个的区别吗？能否准确的用不同的箭头、图线来画出来吗？即便你能画出来，团队里的小伙伴都能看懂吗？ 不过，关于类之间的关系，我后面会在实战篇中讲到的，但是，我会简化成四种关系，更好理解。

2019-11-11 09:43

辣么大

关于uml类图引起了大家的广泛讨论。我同意老师的观点，uml类图还是太复杂了。我给大家一个链接。Uml类图是不用记的。用的时候看一下cheat sheet就行。https://github.com/gdhucoder/Algorithms4/blob/master/designpattern/pic/umlcheatsheet.jpg

2019-11-11 12:04

极客不落

Day007 04

关于 UML 推荐一本书《Java Modeling In Color With UML》和一个神器：https://app.zenuml.com

2019-11-11 23:21

卢爱飞

我理解的是要因场景而异，但是最终的目的都是降低沟通的成本。

场景1：在大多数人对UML不是很熟练的情况下，如果采用UML来进行沟通，大家在理解上一定会存在Gap，无形之中会提高学习和沟通的成本，在这种情况下，建议不使用UML。举个例子，《实现领域驱动》的作者一开始是使用UML和领域专家沟通，作者认为UML很简单，但是许多领域专家或开发人员并不能很好地理解，最后又出现了ES（事件风暴）的形式来替代。

场景2：如果需要准确传达设计意图，还是需要UML这样的通用设计工具的，目的也是降低沟通的成本。例如，架构师的设计理念想准确传达给工程师，如果使用UML工具，可以避免模糊意图，带来额外的沟通成本。

敏捷宣言的第一条就是“个体和沟通”高于“流程和工具”。所以要因人而异，因场景而异，在专栏里“很多类图我并没有完全遵守 UML 的规范标准”的策略，我想是一个不错的折中。

2019-11-11 16:30

确认过眼神

对于uml来说，简单点是可以的，但是对于规范还是要有的。如果不规范，会的人看不习惯，不会的人容易被带入误区。想学的人，画得再难也会去看，不想学的人，画得再简单易懂，也不会去学。

2019-11-11 08:12

黄林晴

打卡~

我觉得在工作中，如果完成一个功能需要30分钟，其实25分钟都在思考，25分钟在设计，实际编码时间只需要5分钟，而前面25分钟就是编码设计

2019-11-11 00:30

丁丁历险记

下班后发现争哥让我出代码示例和说明区别，赶紧做。

写点简单粗暴的个人理解。

— show me the code ..

泛化（Generalization）class BaseComponent { ... } class Dingdding extend BaseComponent { .. }

实现(Realization) 类实现接口 虚线加三角interface crud { func create(); func update();func get(); func() del{ } }

class DingdingModel implements crud {

func create(){ ...}

func update(){ ...}

func get(){ ...}

func del(){ ...}

}

关键是后面四个 （关联，聚合，组合，依赖）先说关联关系。 （A has B）

class DingdingUser {

privte $account; //有一个账号对象，

}

再说聚合，是一种特殊的关联。

聚合，组合， 一对多的关联

聚合关系是“has-a”关系，组合关系是“contains-a”关系，少一个宿主对象死掉没。

uml2.x 已合并这种无聊的区分

聚合示例class birdsGroup(){

private $birds;

//聚合往往可以干增减相关的操作

public func addBird($bird) { ... }

public func removeBird($bird) { ... }

}

组合示例。class bird (){public $wing; //鸟由翅膀 组成.

}

最后说关联合依赖。

泛化 = 实现 > 组合 > 聚合 > 关联 > 依赖

2 然后说些个人理解：我回顾了一下，oop 的过程。

在框架的辅助下，数据库建模一作，其实文件放哪，啥关系就出来了，画uml 图反正了一个体力活。
往往是去实现一个需求，将一个业务流走通，写了一段代码后，发现这里写死了，于是做点配置管理（中心控制原则），一个类权责过多，于是将其支解。（类的单一职责原则），重复的代码出现了，赶紧抽离出来，先简单粗爆的用一个类的静态方法抽（很多大神不建议这样做，我不明其理，慢慢研究），某个操作，有几种不同的类可以去做实施。例如日志。（redis 写日志，文本日志，数据库日志，控制台输出）于是搞个工厂模式，遵守下dip原则，让Log::log($log_msg, $type,$tags) 成为一种面向抽象开始，而不是面向具体实现。老板看到gmail 的undo 很酷闲着蛋疼的让你对所有操作，都要求在半分钟类允许undo，上个command 应付下，不出意外 没多久，老板的redo 需求来了，就顺着扩展，进一步的有时候，一个主体业务完成后，要做一堆关联的杂七杂八的事，于是搞个观察者模式，这样将主体业务和后序操作业务解耦了。 继续扯到解藕(decouple)了，我粗浅的来看，折腾设计模式，本质是解藕，找到合适的方法，在合适的场景下做对应的解藕操作。
这么一折腾下来，类和类啥关系，好像压根没太在意到。。。。 但类之前又确实有关系 。 挺想想知道其它伙伴们是如何做oop 的。
自我总结，套路包确实掌握了几个，但总感觉是在很浅的层面上折腾，上述错的乱七八糟的，烦请指正，个人平时就这么想的，这八个月就跟着争哥好好学习了。
2019-11-12 22:43


daniel李
当看到老师说uml意义不大的时候我就懵了，还好原来是指不需要按严格标准死磕uml。

我平时在功能开发初期和后期都是用uml把我的想法可视化然后让师兄审核，减少pr被reject机率。而且也容易让别的工程师接手做功能拓展。

不过确实互联网公司如果不是大厂，确实很少人能看懂uml。
2019-11-11 07:32

作者回复
实际上，大厂也未必都在用。比如类图中几种类关系，同学们有几个能准确的用不同的图线画出来呢?
2019-11-11 09:31


NeverMore
对于UML，我觉得同样不要过于“学院派”，过度求其严谨，而忘记使用它的目的是什么，此谓舍本逐末。毕竟它终究只是一个工具，最终能够服务于我们的表达，方便我们的交流即可。是否要简化，当然也要看场景，至少对于学习这门课程而言，并不需要让其过于复杂而提高我们的学习成本。
另外，我特别欣赏老师这种删繁就简、力求简约和高效的风格，或许这也是一种极客精神吧。
2019-11-11 14:46


方向
UML在毕设时候是必须的，什么用例图，时序图，活动图，非得写上去才显得高大上，但一直不得要领，当时也是网上搜相关的模仿着填充进去。始终认为这种图的目的也是为了传达明确的设计意图，遵循最基本的规范能够达到看懂、意图明确的效果就行了。
2019-11-11 08:52


香蕉派2号
1.同意老师观点，UML是提供一种规范和准则，如果严格的按照规范来做可能过犹不及，在时间成本和规范之间必尽量要做到平衡。
2.除了以上的概念，还想到了低耦合高内聚，模块化，可维护性，可扩展性，可复用性，对象的唯一性，对象的分类（是is-a还是has的关系）等。
2019-11-11 05:33


BBQCAPTURE
专栏重点是设计模式，只要便于理解，什么样的图都没关系的，凡事都要抓住重点，我觉得软件开发最大的忌讳就是追求完美，死扣细节。
2019-11-11 10:30


唐龙
即便我们使用面向对象编程语言，写出来的代码也不一定是面向对象编程风格的，也有可能是面向过程编程风格的。嗯~刚学C++的时候干过这事。
2019-11-11 00:36


Hash
对于UML（统一建模语言），我个人觉得它的作用还是很大的，因为它可以帮助开发人员更好的去分析一个软件的设计过程，

通过它的哪些表示的方法吗，会让人的思路更加的清晰，如果是一个软件的负责人，那么使用UML来分析问题，我觉得再好不过。
软件开发是一个工程问题，就好比盖房子，只有每一步都规划好，分析好，设计好，盖出来的房子才好，总之，我个人觉得值得花时间去学UML！

2019-11-12 10:53

编程界的小学生

1.我觉得首先uml这东西很牛逼，很有必要去画，但是也需要分场景，比如crud还强行画一个出来那就是浪费时间，比如超级复杂的东西要画，那我觉得就可以简化，多配上文字注释。比如需求一般，不是很复杂也不是很简单的那种也可以好好画一下，必要的地方配上文案描述。uml能帮助我们瞬间理解这个东西到底要做什么，流程是怎样的，画出来不光是现在看还是以后复习看，他都很香！
2.我觉得缺少了一个“组合”，首先要以类和对象作为代码的基石，还要能灵活的支持组合特性才算不严谨的面向对象语言。组合算封装特性的一部分吗？还是说只要以类和对象为基石的开发语言都支持组合？

2019-11-11 09:39

作者回复

组合跟封装应该没啥关系呢。

2019-11-15 11:57

忆水寒

UML设计的合理，新开发者也能去快速开发。关键还是看自己开发还是别人开发。作为学习来说，简单的UML能表达意思即可。

2019-11-11 06:36

Qfxl

看过很多本厂的设计文档 很少会画类的设计图 基本是系统整体架构和流程图 而且代码变化快 文档容易脱钩 首席架构甚至说过不想看类是怎么设计的 最重要还是大方向功能性正确 所以uml大概知道就好了 学有余力了解下亦可

2019-11-27 20:19

阿健

关于uml类图，我要发表一下个人观点。目前我也是在一个互联网公司，我们有比较严格的设计方案模板，其中类图，流程图，用例图，状态图是必备的，时序图暂时还没有那么要求严格，目前团队内统一用platuml来画图，基本进入团队先学这个，如果忘记语法，就去官网查一下。磨刀不误砍柴工，互联网公司开发项目，还是需要uml类图的。

2019-11-19 17:09

村口叶师傅

UML类图在功能开发完成后的交付文档中一般都会提供，方便其他同事了解代码。我不能保证看的人都懂UML类图，但是自己要尽量保证其正确性，最起码让懂的人能看明白

2019-11-13 09:02