成天低头撸代码，就不想偶尔
也抬头仰望一下星空么？

- 程序员老高 -

计算的本质

- 一个野生中年程序员的胡说八道 -

他是谁

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.

- Ludwig Wittgenstein

他是谁

Die Grenzen meiner Sprache bedeuten
die Grenzen meiner Welt.
- Ludwig Wittgenstein

The limits of my language mean
the limits of my world.
- Ludwig Wittgenstein

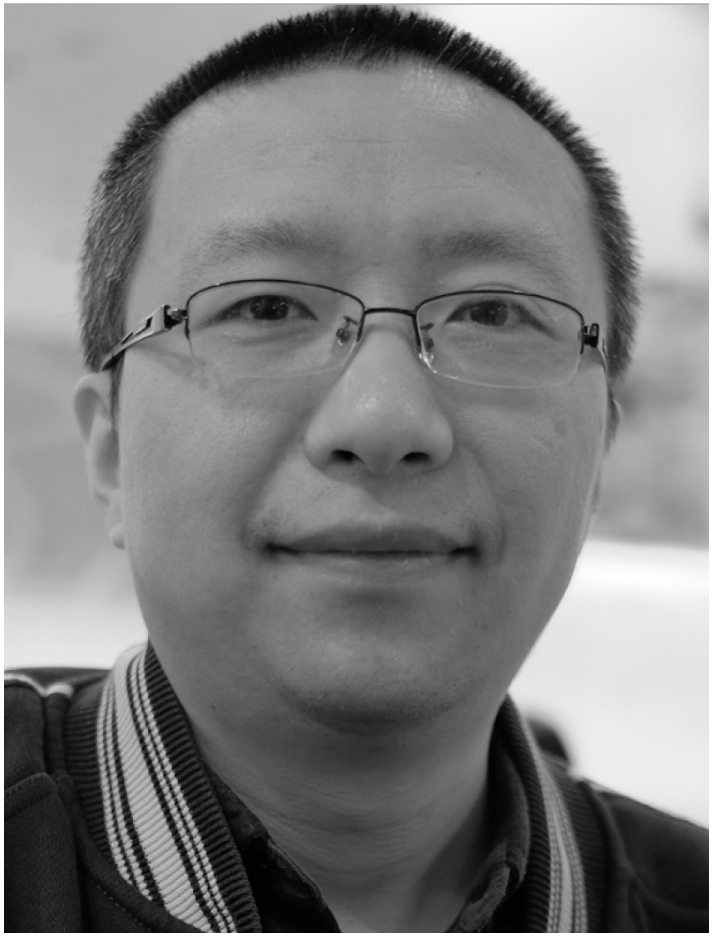
他是谁

Die Grenzen meiner Sprache bedeuten
die Grenzen meiner Welt.
- Ludwig Wittgenstein

The limits of my language mean
the limits of my world.
- Ludwig Wittgenstein

语言的边界就是
世界的边界。
- 路德维希·维特根斯坦

我是谁



@laogao
@程序员老高

70后程序员
Scala
嗜茶
爱听古典音乐

免责声明

本次分享的内容仅为个人观点，不代表主办方及本人所服务公司的立场，不作为投资建议。

今天的主题

从一个多年软件开发从业者的视角，和大家聊一聊，关于范畴论和编程的一些思考。不求严谨，但求有启发。

有什么预期

如果你已经对范畴论有所了解，接下来的20多分钟，可能不会带来什么新的知识点。也许有新的视角，但我不能保证。

如果你之前完全没有听过范畴论，也没关系，只要你听得懂中文，做得来小学应用题，我保证不会有你听不懂的语言或概念。

如果你恰好是软件行业的从业者，但一直理解不了别人说的范畴论和函数式编程这些研究课题、概念、工具的意义在哪里，那么我觉得你来对了地方，希望你从今往后，再看世界，会有不一样的洞察。

好我们开始

→

立论：什么是计算

计算是
通过有意识的抽象，用可被执行的算法，解决现实世界的问题。

展开来说

现实世界的问题，可以通过一系列合乎逻辑的变换和建模，投射到另一个（抽象的）范畴，并用这个新范畴内的语言和方法（比如某个分支的数学，或计算机程序）来描述、推演、求解，得到答案。

这样，我们就能用我们熟悉的（或者说更为方便的）语言和方法，来“计算”现实，大大提升了人们应对和处理现实问题的效率。

什么是抽象

抽象是
去掉不重要、不相关的细枝末节的过程。

展开来说

抽象是相对于具象来说的。在这个现实世界和自然语言的关系中，现实世界是具象的，自然语言是抽象的，自然语言是人类对现实世界的抽象。

但光有自然语言是远远不够的。人是非理性的动物，认知和思考能力有限，而语言又是灵活多变的，存在各种二义性。在这样的大前提下，为了有效地解决现实世界的问题，除了暴力，我们还有什么别的更优雅的方法？

展开来说

答案是：逻辑和有意识的抽象。

逻辑让我们从无序中找到有序，有意识的抽象有助于去掉那些跟问题本质无关的细节，而不是随便什么细节。

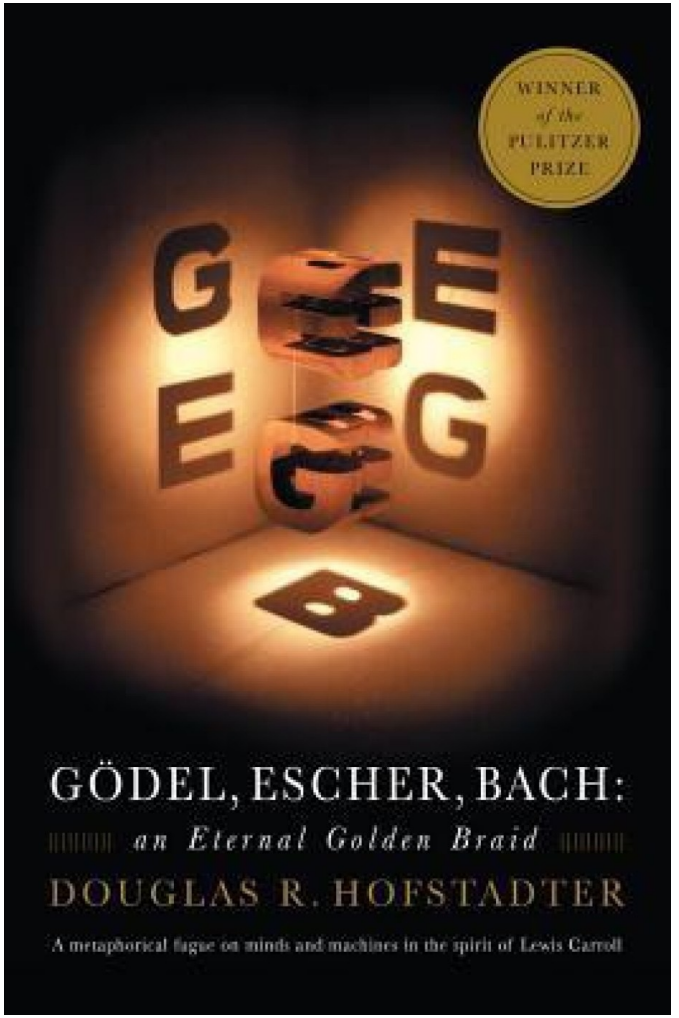
自然语言相对于现实世界的抽象，这个过程中丢失的细节通常是碰巧，是因为我们不理解很多事情，而不是因为我们确切地知道哪些细节是解决问题的核心和根本，哪些不是。

什么是逻辑

逻辑是
对有效推论的哲学研究，也就是形而上的研究。

什么是有意识的抽象

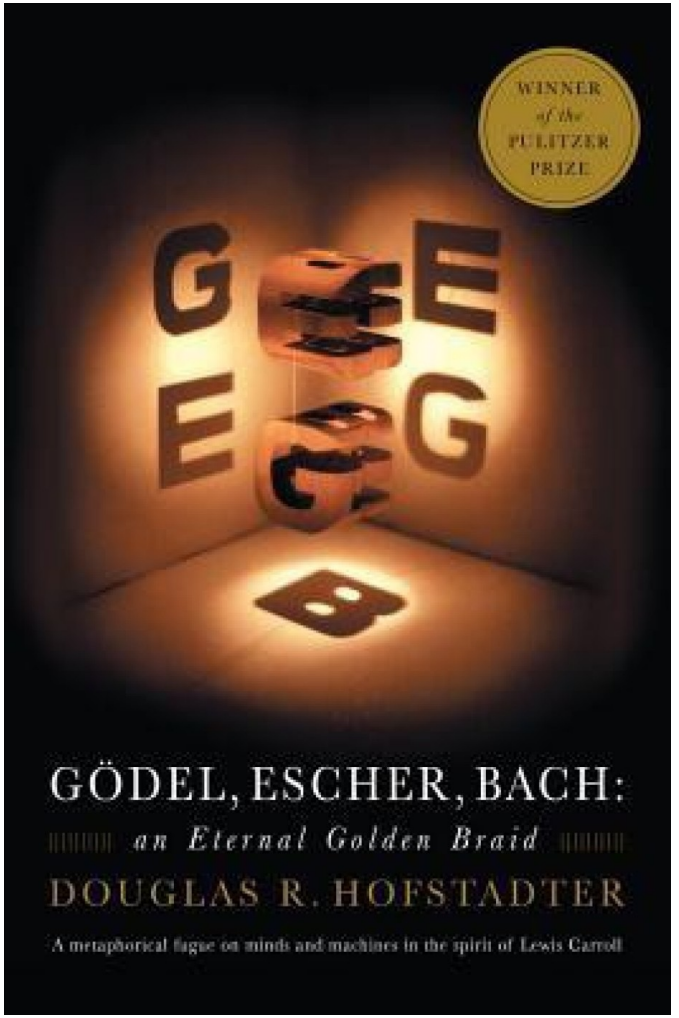
看问题的角度很重要：
要解决什么问题决定了不同抽象的合理性排序。



什么是有意识的抽象

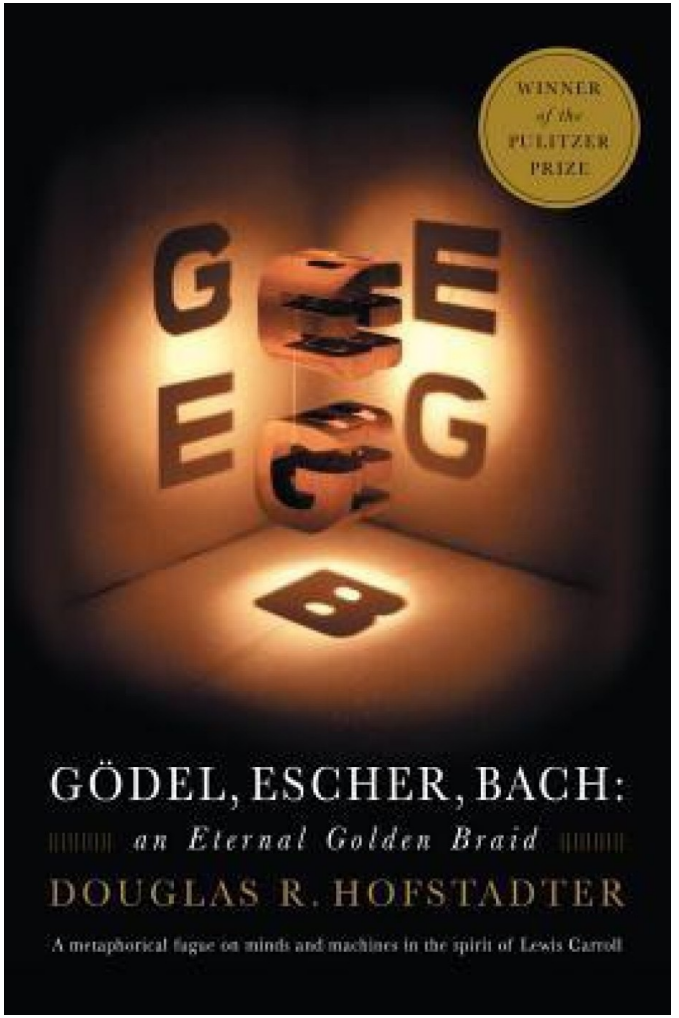
抽象到什么程度是合理的：
比如探照灯，离地面越近，照到的细节越多，但容易丢失全貌，离地面越远，则照得更广，但更细节的部分因为光的强度不够，就会变得不再明晰可辨。

人脑的工作原理类似。

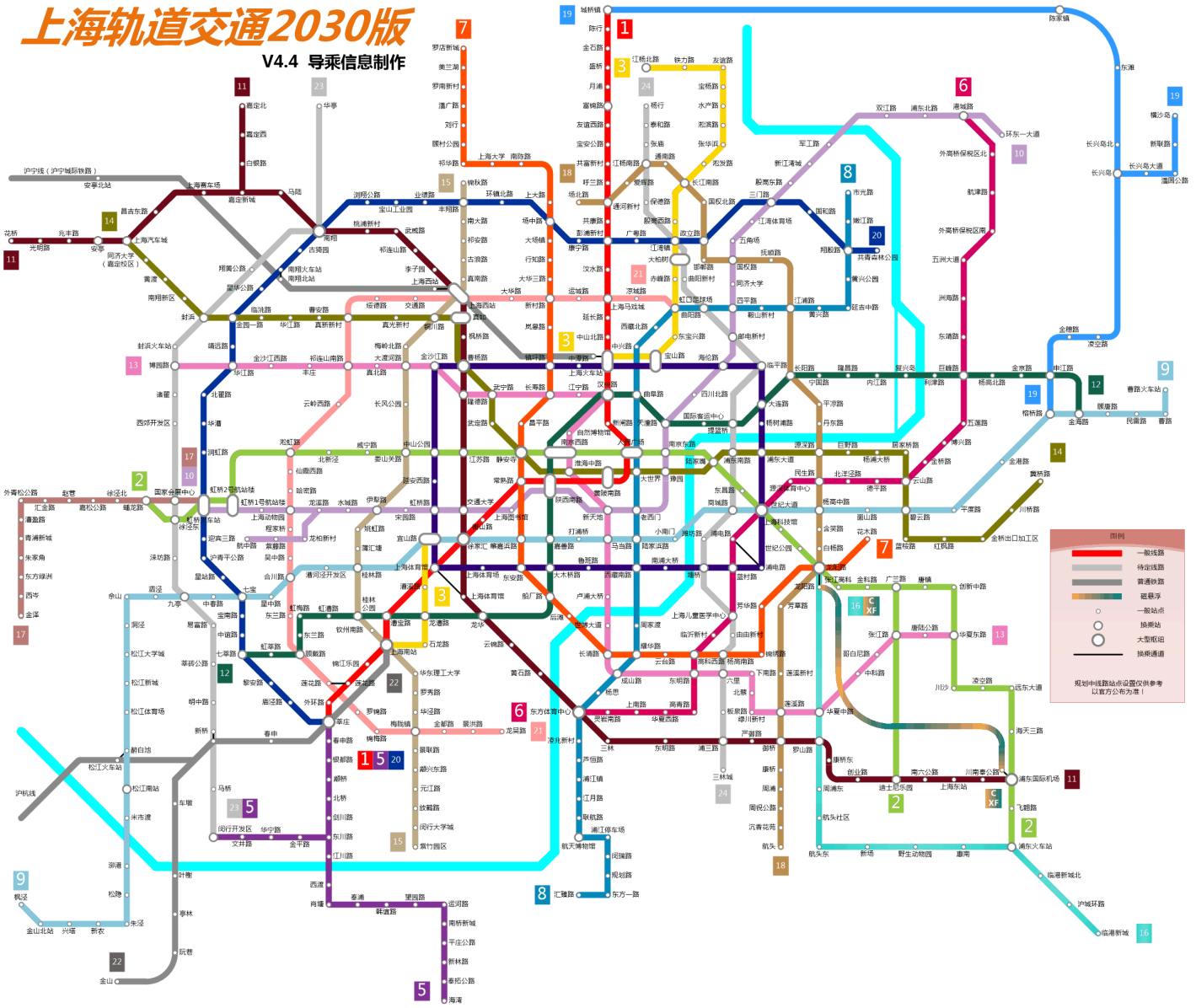


什么是有意识的抽象

抽象过程中被忽略的细节不是不存在，也不是对最终结果完全没有影响，只是这些细节在思考和推演的特定路径上不重要（有干扰），我们并不能因此忘记它们。



大家一定都见过类似这样的图



什么是算法

算法是
用于解决某一类特定问题的具体操作步骤的规约。

回顾一下

计算是
通过有意识的抽象，用可被执行的算法，解决现实世界的问题。

回顾一下

Q. 要解决现实世界的问题，我们追求的是什么？
A. 是效率。

回顾一下

Q. 提升效率的关键是什么？
A. 是对问题正确的认知和抽象。

回顾一下

Q. 对问题正确的认知和抽象，我们有什么特别好用的思维工具？
A. 有，最好的工具是数学，以及数学的数学。

什么是数学

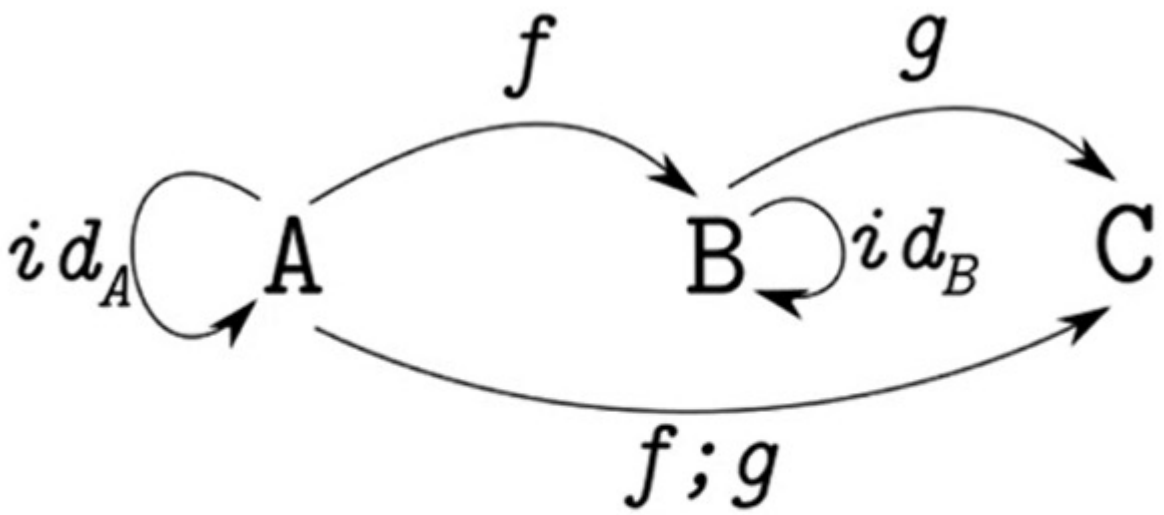
数学是
对合乎逻辑的事物如何运作的合乎逻辑的研究。

什么是数学的数学

数学的数学就是范畴论。
是对合乎逻辑的事物如何运作的合乎逻辑的研究的合乎逻辑的研究。

数学的数学

范畴论讲的是，在特定上下文中，对象和对象之间的关系。



$$id_A ; f = f ; id_B = f$$

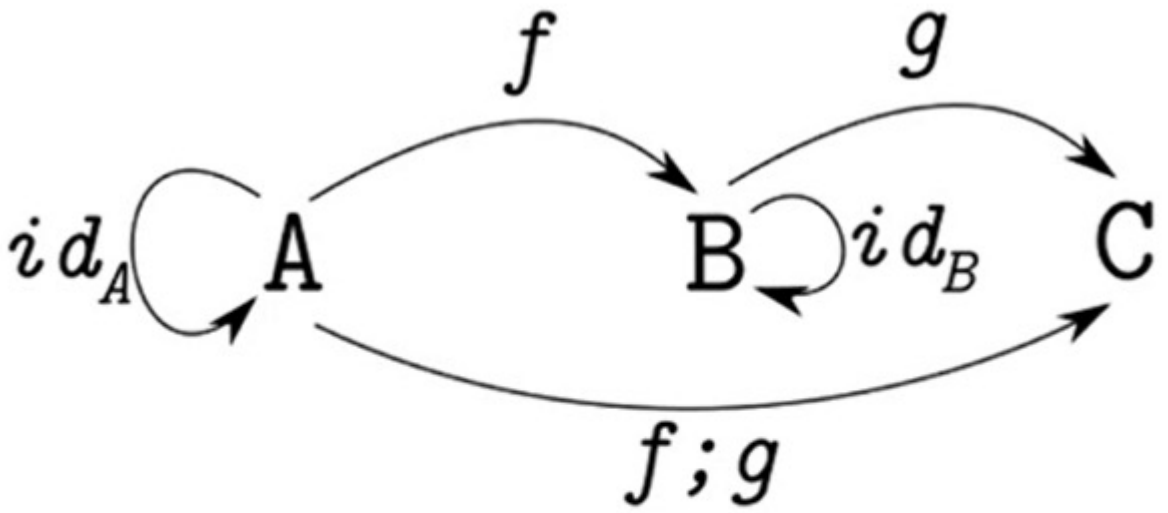
$$(f;g);h = f;(g;h)$$

范畴论和编程的关系

范畴论中的对象，相当于编程世界的类型；
范畴论中的关系，相当于编程世界的函数；
范畴论中的组合，相当于编程本身（函数和函数组合成算法）；
范畴论中的追图，相当于编程世界的程序执行；

范畴论和编程的关系

类型、函数和组合



$$id_A ; f = f ; id_B = f$$

$$(f;g);h = f;(g;h)$$

为什么要用范畴论理解编程

→

什么是编程

Q. 编程的核心是什么？
A. 是算法。

什么是算法

算法是
用于解决某一类特定问题的具体操作步骤的规约。

什么是算法

算法是
用于解决某一类特定问题的具体操作步骤的规约。

而操作步骤之间存在顺序和依赖关系，排在后面的步骤可能会依赖排在前面步骤的输出。

什么是算法

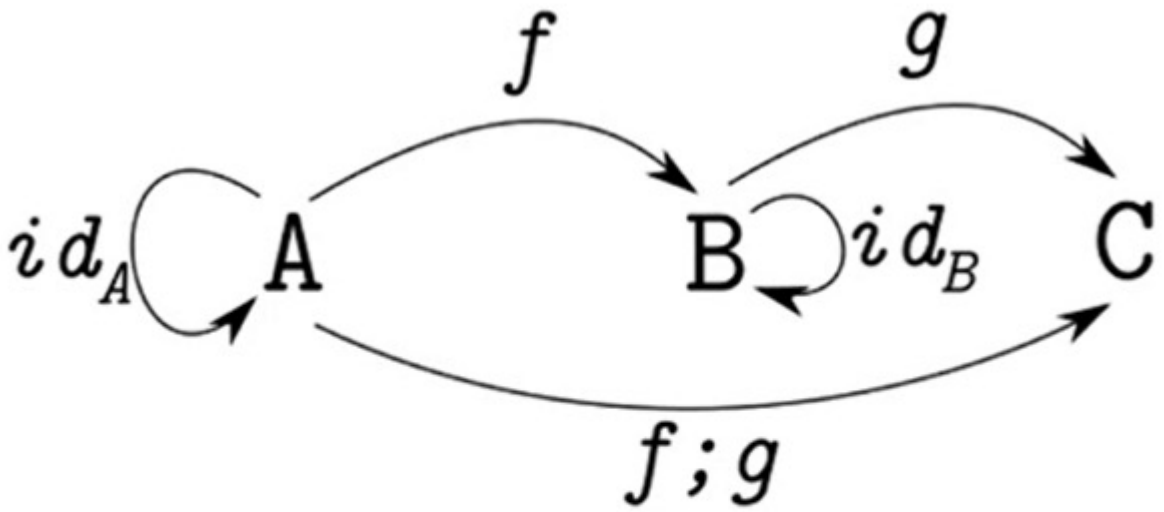
算法是
用于解决某一类特定问题的具体操作步骤的规约。

而操作步骤之间存在顺序和依赖关系，排在后面的步骤可能会依赖排在前面步骤的输出。

这种顺序和依赖，对应到范畴论里，就是那些箭头，以及箭头之间的关系。

用范畴论描述的编程

对编程而言，范畴论这张图核心意思就一个：组合
函数f和g可以组合，因为f的返回值类型与g的参数类型相同



$$id_A ; f = f ; id_B = f$$

$$(f;g);h = f;(g;h)$$

举栗子之前

我们来回顾一下范畴论有哪些核心内容：

对象 + 对象之间的关系
点 + 箭头
范畴论描述的是关系如何compose

举栗子之前

形式逻辑也可以用范畴论的语言来表达：
人固有一死，苏格拉底是人，所以苏格拉底会死。

举栗子之前

回顾一下什么是算法？

算法是
用于解决某一类特定问题的具体操作步骤的规约。

算法描述的是工序，我们天然理解的，日常经常要通过编程来处理的业务逻辑，也都是工序。而工序的核心，正是如何**compose**，即如何组装我们的代码逻辑。有前后次序，且排在后面的依赖排在前面的步骤。

经常出现的一种情况是若前序未完成，则后序无法进行。但这并不影响我们对将要发生的事情进行描述。

准备好了吗

举个栗子

为了方便讨论，我们举一个超级复杂的栗子。假定：

类型**A**表示商品**ID**（比如**Long**）
类型**B**为表示商品（比如**Item**）
类型**C**为表示商品名称（比如**String**）

函数**f**表示通过商品**ID**查询商品
函数**g**表示通过商品获取商品名称

问题：
已知商品**ID**，求商品名称

举个栗子

答案：

如果给我一个从商品ID到商品的函数，再给我一个从商品获取商品名称的函数，我就能通过商品ID得到商品名称。

举个栗子

Surprise!

举个栗子（第二季）

如果
- 我们有多个商品ID
- 每个商品也有一个或多个名称

……

在这样的世界里，我们先前的知识还有用吗？

举个栗子（第三季）

如果：
- 我们不一定能拿到商品ID（可能调用方没有明说）
- 通过商品ID不一定能获取到商品（可能商品下架了）
- 每个商品的商品名也可能取不到（可能某个商品没有维护名称）

……

在这样的世界里，我们先前的知识还有用吗？

举个栗子（第四季）

如果：
- 我们暂时还没有拿到商品ID（过一阵子会）
- 通过商品ID不一定能当场获取到商品（可能需要网络请求）
- 每个商品的商品名也可能取不到（可能又是另一次远程调用）

……

在这样的世界里，我们先前的知识还有用吗？

统一回答

在上述三个新的世界中，我们过去的知识依然有效！

统一回答

三个「世界」
对应三个「范畴」
翻译成代码：

List[T]
Option[T]
Future[T]

统一回答

每个新的范畴，都是在原有核心算法的基础上，
额外的处理了特异性的部分，即：
单值/多值
可选值
将来值
……
原有的函数组合关系依然成立。

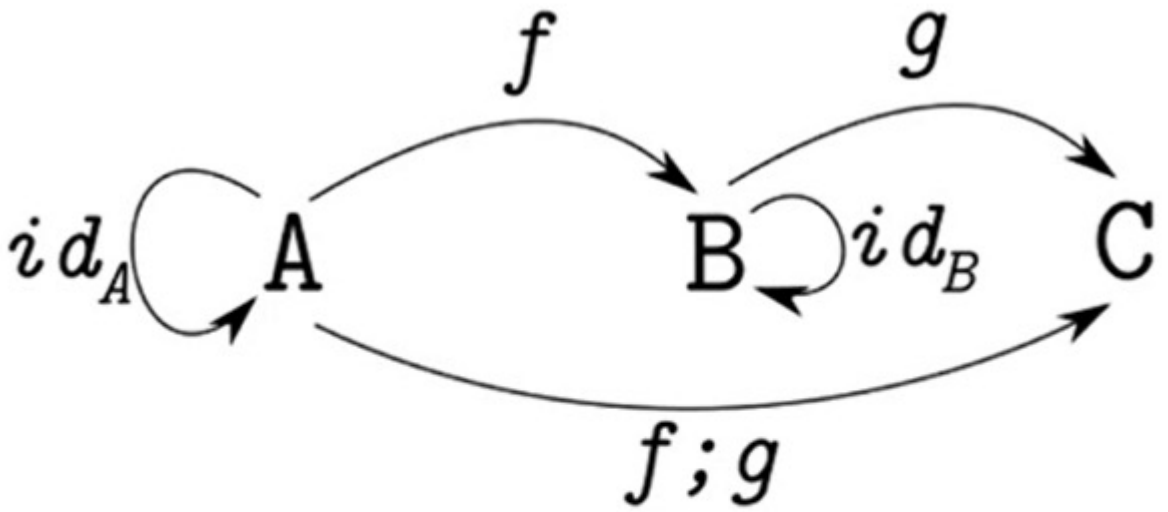
统一回答

除了原有的函数组合关系依然成立之外
还有一个显著特征：

在我们「追图」的过程中
对象形状保持不变
List[T]
Option[T]
Future[T]
也就意味着它们都将继续待在同一个范畴中
否则无法**compose**

再来回顾一下这张图

不同的范畴对**A**、**B**、**C**赋予了更多的关于上下文的「信息」



$$id_A ; f = f ; id_B = f$$

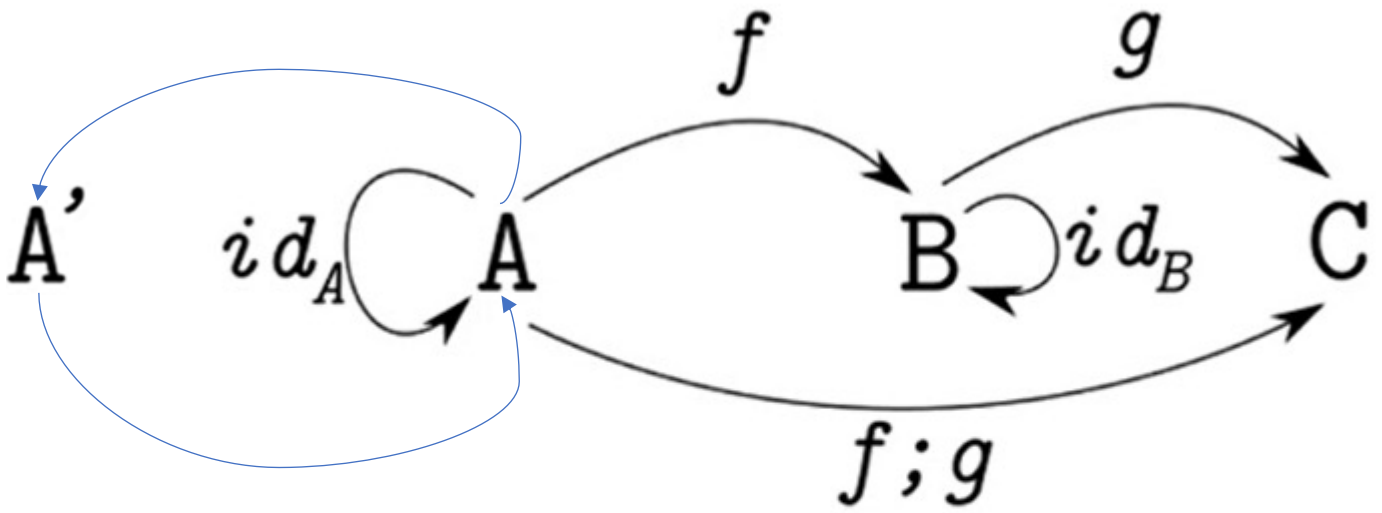
$$(f;g);h = f;(g;h)$$

再稍微展开一丢丢

什么是isomorphism
什么是identity
为什么要有identity

用范畴论描述的编程

想象A的左侧有个A’
而从A到A’ 和从A’到A各有箭头确保在A和A’之间互转不丢信息

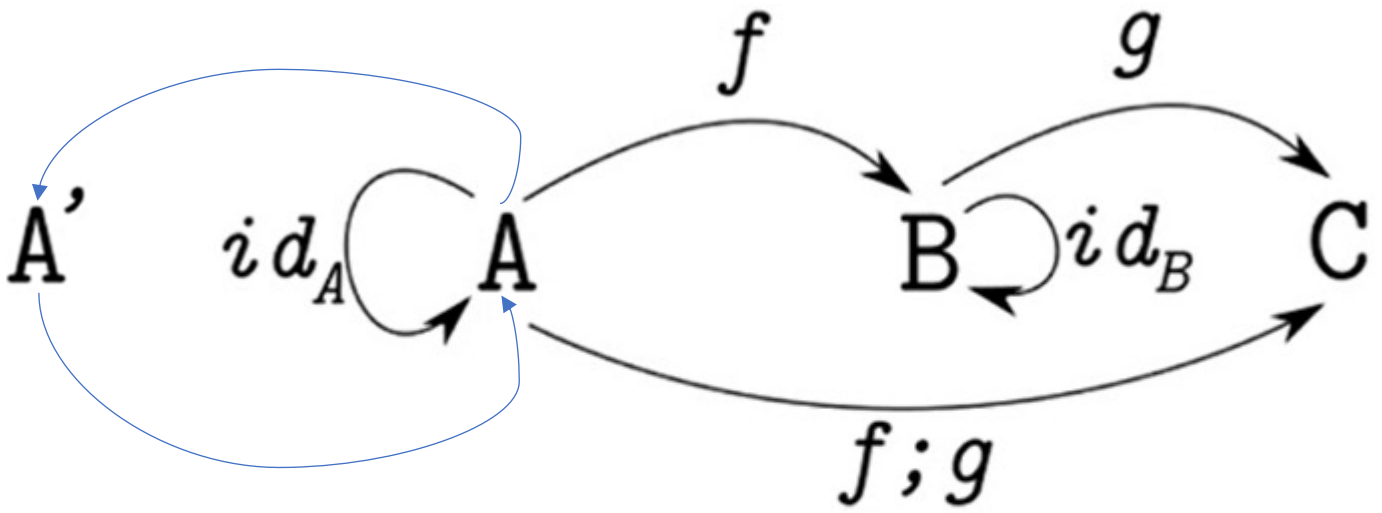


$$id_A ; f = f ; id_B = f$$

$$(f;g);h = f;(g;h)$$

用范畴论描述的编程

在这样一个范畴里，A和A'是等价的，我们称之为isomorphic
而连接A和A'的两个箭头，组合在一起，就是identity

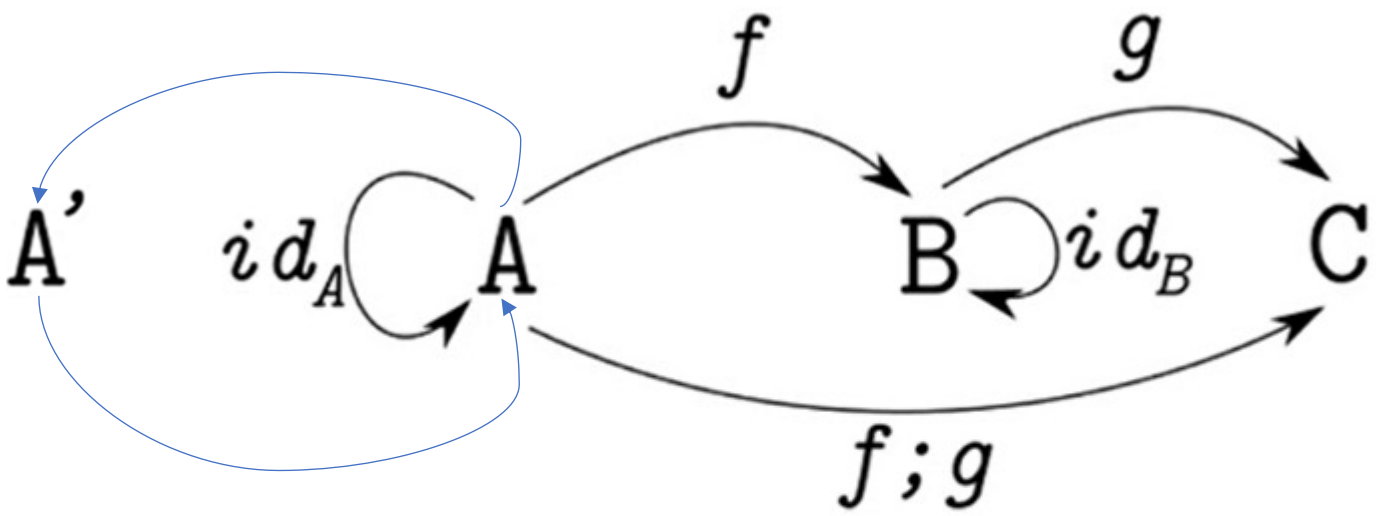


$$id_A ; f = f ; id_B = f$$

$$(f;g);h = f;(g;h)$$

用范畴论描述的编程

identity就是我们在List[T]、Option[T]、Future[T]看到的构造函数



$$id_A ; f = f ; id_B = f$$

$$(f;g);h = f;(g;h)$$

发散思考

过程中抛异常怎么办？

发散思考

需要额外的依赖怎么办？

发散思考

我们真的很想复用我们已经掌握的通过商品ID获取商品名称这个逻辑啊！

发散思考

ZIO[-R, +E, +A]

再回顾一下

计算是
通过有意识的抽象，用可被执行的算法，解决现实世界的问题。

再回顾一下

Q. 要解决现实世界的问题，我们追求的是什么？
A. 是效率。

再回顾一下

Q. 提升效率的关键是什么？
A. 是对问题正确的认知和抽象。

什么是抽象

抽象是
去掉不重要、不相关的细枝末节的过程。

我们关心的，对问题域有帮助的，留下；
我们不关心的，对问题域有干扰的，去掉。

对编程而言

重要且相关的细节：

我们有哪些数据类型 → 对象
我们有什么样的基于这些数据类型的函数 → 关系
这些函数如何有效地组装成满足业务需求的算法 → 组合
这些函数在执行时有没有特别的上下文需要保持 → 投射

对编程而言

重要且相关的细节：

我们有哪些数据类型 → 对象
我们有什么样的基于这些数据类型的函数 → 关系
这些函数如何有效地组装成满足业务需求的算法 → 组合
这些函数在执行时有没有特别的上下文需要保持 → 投射

范畴论帮助我们理解和消化从真实世界到程序之间的过程。

进入尾声

→

什么是计算的本质

最后说说什么是计算的本质

什么是计算的本质

计算的本质是跨界：
用一个范畴的方法解决另一个范畴的问题。

* 每个范畴都有自己独特的上下文，或者说，每个范畴都有对什么是有效信息的定义和取舍。

什么是计算的本质

计算的本质是跨界：
用一个范畴的方法解决另一个范畴的问题。

* 每个范畴都有自己独特的上下文，或者说，每个范畴都有对什么是有效信息的定义和取舍。
** 程序的范畴特别有意思，因为它辅助我们利用计算机，利用编程，解决现实世界的问题。

有趣的事情总是发生在边界

大数据 + AI

有趣的事情总是发生在边界

FP + OO

有趣的事情总是发生在边界

统计分析 + Markdown

有趣的事情总是发生在边界

程序员天然是跨界的

我们日常的编程活动
从需求分析到开发落地
都是在跨界

用一个范畴的知识
解决另一个范畴的问题

跨界的工具

就像我们从新手村出发。

跨界的工具

出发前，我们已经完成了核心技能(范畴0)的学习。

跨界的工具

我们要做哪些准备，来迎接未知的世(需)界(求)呢？

跨界的工具

范畴论

跨界的工具

集函数式编程和面向对象于一身的Scala

多说两句Scala

即便在内部
以功能特性这个视角看
Scala也提供了很多让你「跨界」的机会

多说两句Scala

opt-in

多说两句Scala

模式匹配是一种opt-in

多说两句Scala

for推导式是一种opt-in

多说两句Scala

typeclass(类型族)是一种opt-in

为什么这个工具不流行

工具的价值在于解决问题，而不是流行。

为什么这个工具不流行

人，不是机器。

我们学习工具的目的，不是最终被工具和机器替代，我们思维要往上走。所以评价某个工具的好坏，不是它流不流行，而是它能不能帮你解决现在和未来面临的问题。

为什么这个工具不流行

范畴论就是这样的思维工具。

它不一定直接有用，
但绝对能跨界帮你找到有价值的抽象。

人人都是产品经理

产品经理都应该学点范畴论

人人都是程序员

程序员都应该学点Scala

人人都是Scala程序员

Scala程序员都应该体验下ZIO

Broadview
www.broadview.com.cn

基于Scala 3.0的完全指南

Scala编程

第5版

Programming in Scala, Fifth Edition

artima

[德] Martin Odersky [美] Lex Spoon [美] Bill Venners [美] Frank Sommers 著
高宇翔 译

中国工信出版集团 电子工业出版社

Scala编程
第5版
[德] Martin Odersky
[美] Lex Spoon
[美] Bill Venners
[美] Frank Sommers 著
高宇翔 译
电子工业出版社