人才培养

浅谈国内高校编程语言教育

李 樾
南京大学

关键词：程序设计语言 编程语言 高等教育

前言

2018 年，我在丹麦做博士后研究期间撰写了这篇文章，当时的写作契机是我感受到虽然我国的计算机类高等教育行之多年，但其软件核心——程序设计语言的教育却较国外有不小的差距。导致差距的原因是什么？国内的程序设计语言的教育是否真的需要进一步提高？它真的有那么重要吗？如果要改进的话，可以从哪里开始着手？针对这些问题，我在本文中给出了我的一些浅见，并做出了一些呼吁，希望同道中人看到后，记在心里，默默努力，能为祖国的程序设计语言教育蓄一份力。

2019 年末，我加入了南京大学计算机科学与技术系。回国后我发现，实际上我国已经开始在程序设计语言的教育与科研上发力了，包括北京大学引进程序设计语言国际知名学者胡振江教授；教育部高等学校计算机类教指委委任上海交通大学傅玉熙教授组织筹建“计算机专业教育丛书——程序设计语言系列”编委会，由程序设计语言国际知名学者、南京大学冯新宇教授（华为编程语言实验室主任）担任主编，他们当时正在筹划建设我国急需的程序设计语言本科教材。此外我还知道有一个“程序设计语言组（筹）”微信群，群里有很多学者讨论程序设计语言的教育，并献计献策。因此，现在我对我国未来程序设计语言的高等教育很有信心，我也将和这些优秀的前辈和同仁们一道，为国内高校程序设计语言的教育与科研贡献一份力量。

此文虽言不至肺腑，但拳拳之心在目，寥寥心得，一家之谈，权作抛砖引玉。

开篇

在互联网上活跃着很大一批“潜水的”程序员，有的出于对编程的热爱，不断磨练、提升自己的编程阶位；有的迫于自身的能力和工作的需要，不得不强迫自己加班补课，生怕掉队；有的一直努力着试图在高薪与兴趣之间牵上一条能够说服自己的线。他们大多低调勤奋，每天与代码死磕，拿着相对称心工资的同时，也为加班的压力烦心无奈，但更多的幸福也许来自每每调试成功或完成任务那一刻的暗自欣喜，悄悄地告诉自己“我还不赖”，默默地激励自己“我可以更好”。这篇文章就是写给那些简单的人——如今和未来的“程序员”们。

动机

此文的写作动机主要来源于两股力量的碰撞：惋惜和鼓舞。

惋惜

我发现很多科班出身的计算机或软件专业的大学生对一些编程语言[1]问题如“什么是解释器”“什么是编程语言的语义”“什么是函数式编程”“为什么语言会有静态和动态类型”等基础知识并不清楚，

更不要提诸如“编程语言的核是什么”“它们如何被一些基本的语言特性组成搭建最终形成自己的语言生态”等更进一步的知识。更令人不安的是，很多人认为“我不了解它们，但这并不影响我编程，所以不知道也无所谓”。

想到这里，我的脑海中立刻浮现出我人生中的第一门编程语言课“C语言”。当时全国通用的教材是清华大学谭浩强老师编著的绿色封面的《C程序设计》。虽然授课教师不断强调“变量使用前应该声明”“循环数组不能越界”“printf函数的参数有哪些”……之后又学了一些编程语言（Java面向对象编程设计、C++等），但一路下来我对于某个编程语言的理解依然停留在语法和开发环境层面，知道如何利用帮助文档和一些相关框架或函数库完成实际编程任务，然后心虚地在个人简历上写上熟悉该编程语言。

如今想来，很多学生不喜欢编程，感觉它难且太复杂，枯燥无味却又不得不死记硬啃，写了那么多代码和作业却依旧感觉与理解编程语言相距甚远。这很大程度上不是学生的问题。如果教师仅一味地讲解这些语言的语法细节以及如何使用它们实现数据结构和算法，就很难让学生真正理解编程语言并提起兴趣。这些学生或程序员本可以更喜欢编程，更热衷于编程工作。想到这里，不由地为他们（也是当年的自己）感到些许惋惜。

鼓舞

同时我在互联网上又发现另一拨人。他们是纯粹的编程语言爱好者，经常利用互联网分享和交流很多有趣且有用的编程语言知识。我觉得他们对编程语言的理解很深且很羡慕他们，因为他们讨论的很多有用的知识我都不懂，甚至之前都没听说过，同时也感受到他们对自己喜欢的东西的那种难得的专注和纯粹。

这些对兴趣的专注和纯粹本应是大学生在所学专业上该有的年轻的、百舸争流的样貌，如今却被年薪、房价和社会的物化价值观所左右。如果连我国顶尖大学的教育都摆脱不了物欲横流的浮躁，又如何完成国家民族集体人格的进一步塑造？在这种背景下，这些编程语言爱好者们的所为有如夏日之凉风，让我意识到我国始终有一群爱学习、爱思考、忠于兴趣的人，他们的存在激励着我说出想说的话，希望让更少的学生被惋惜，让更多的学生受鼓舞。

为什么编程语言教育在我国重视力度不够？

计算机行业日新月异，随着计算能力的不断提升和信息智能化在各行业的大量普及，“人工智能”“大数据”等领域受到了前所未有的瞩目。加之我国正处于产业升级之时，想要利用国内巨大市场和国人对于新鲜事物特有的包容心态，在所谓“工业革命4.0”的历史进程中弯道超车，我国把“人工智能”“大数据”“物联网”等计算机应用列为了国家战略规划项目。

这些行业的从业人员没有让大家失望，很多智能应用确实在很大程度上改善了人们的生活体验。然而如此一来，计算机应用方向强大的优势更容易让人们忽视对计算机基础方向的关注和发展（基础方向因其回报不能被立即“看到”已经受到了一定程度的忽视）。最重要的计算机基础方向之一就是编程语言，这也是在我国一直不被重视的研究方向。

如果说硬件的核心是芯片，那么软件的核心便是编程语言。中兴芯片事件闹得沸沸扬扬，当时我看到报道后既气愤又无奈。其实我国对芯片的研究是有投入的，然而由于芯片行业本身研究成本极高，还要和国际上更重视更有钱的机构竞争，创新成本太大。可是编程语言作为软件的重要核心，研究成本相较于硬件几乎为零，在我国却一直不被重视，因为很多人对编程语言抱有一种误解的态度：“通用的编程语言全世界就那么几个，该有的都有了，我也会用，有什么可研究的？总不能让我再去设计一个新的编程

[1]文中出现的所有“编程语言”都指“程序设计语言”。

语言吧？那有什么用？而且也没人用啊！”

如此一来，编程语言的研究很容易被轻视，进而导致其研究人才的缺失。人才缺失意味着没有足够专业的人来为其重要性发声，又因为其作为基础方向，发展效果不像应用方向那样“立竿见影”，国家高等教育决策层很难主动意识到发展它的必要性，因此决策层便不会发展这一方向并重视、引进相关人才。

此外，编程语言是在国外发展了近 60 年如今仍旧活跃的一门结构性很强的重要计算机研究方向，并不是只要会编程、有博士学位和相关计算机背景的人就能教好这门课。我国缺乏编程语言方向的人才，势必造成今天国内高校编程语言教育相对落后的局面。

重视编程语言教育真的很重要吗?

“在 X 年内，进入世界知名大学行列”“在 X+Y 年内，冲入世界一流大学行列”是近几年国内很多高校的通用口号模板。然而再来看看真正的世界“知名”“一流”大学（例如麻省理工学院、斯坦福大学、加州大学伯克利分校、卡耐基梅隆大学、康奈尔大学、宾夕法尼亚大学、华盛顿大学、哈佛大学、苏黎世联邦理工学院、牛津大学、剑桥大学等名校），会发现它们之中但凡有点规模的计算机院系，几乎都有编程语言方向的教授或研究人员，而且往往排名越靠前的大学越重视编程语言，相关科研人员也越多，编程语言课程都是由这些专业人才讲授的。

反观我国绝大多数 985、211 高校，无论是计算机学院还是软件学院，在编程语言方向上居然很少有全职教授。此外，中国计算机学会（CCF）有那么多计算机子方向的专委会，但就是没有编程语言专委会（形式化方法专委会和软件工程专委会很难直接取代编程语言专委会，反之亦然）。这一对比应该就能说明问题了：不是我们有问题，就是那些世界一流大学有问题。然而我认为，我国编程语言教育的落后和人才的缺失是个无从责备的问题；换言之，我相信如果我国高校或教育决策层的领导意识到了这个问题的重要性，是有动力和能力解决的。

接下来，我想从个人和社会两个层面主观地谈谈提高编程语言教育质量的好处。

个人层面

编程语言虽然复杂，但其设计遵循着一些基本规则。不同语言的语法语义虽然有所不同，但一些基本的核是相似的（例如基于表达式扩展的语法和对基本数据类型的抽象），编程语言可以顺着这些核一点点展开，根据语言设计的初衷选择合适的编程范式（例如，命令式、函数式或两者的混合式），进而对核进行相应的调整（例如函数本身是不是一个基本数据类型），一点点添加需要的规则（例如什么样的作用域或什么样的类型系统）和特性（例如是否支持多线程，是否支持异步），这些规则和特性与编程语言之核的组合，造就了今天琳琅满目的编程语言。

然而，如今国内的大多数计算机专业教育不解释这些基本原理以及它们与编程语言的本质关系，导致学生对编程语言缺乏一种抓得住的“全局观”和举一反三的能力，看到的都是被“糖化”后的语法和复杂的语用环境，所以会感到语言太复杂、枯燥。如此一来，以下情形便容易解释了。例如：

1. 经常对编译器和解释器的报错毫无头绪；

2. 无法理解一些编程框架或库的设计与用法，因此只能模仿模板例子来编程，对于背后被封装和使用的原理一无所知，进而导致一旦有问题自己找不出原因，只能求助于网络或同事；

3. 对于大的任务该用什么样的编程语言或如何利用其特性，无法做出独立的思考，理所当然地认为这是团队负责人该管的事；

还有太多心有余而力不足的情况……

因此，对于个人，受到更好的编程语言教育意味着可以更好地理解从语言本身到构建出的复杂软件系统，对于问题演变的“势”和通过编程手段解决问题的“器”也都会有更好的把握。编程语言的教育好比是为了构建个人学习编程知识树的主干，每次遇到新问题都可以利用已有的编程语言知识解

决，进而每一步编程经历的积累都是给自己的知识树添枝增叶，日积月累，写程序的速度不知不觉地更快，写出的程序更简洁易懂，更少出错，更安全，也更容易学习理解新的编程语言和软件框架。

更重要的是好的编程语言教育可以帮助个人对编程语言产生更浓厚的兴趣，这对于程序员健康心态的调整很重要。作为一名程序员，如果没有兴趣，就会迫于生活压力收罗各种经验编织履历向别人证明自己已走了多远；有了兴趣，就有可能一路打怪升级，并时刻提醒自己还差多远，这一切却又怡然自得，事业幸福感油然而生，生活幸福感便交相辉映。

社会层面

随着社会对于信息化的不断依赖，软件势必走向复杂化，进而对软件可靠性、高效性、安全性等都提出了新的挑战。这意味着对程序员个人编程能力和素质都会有更高的要求。因此，我国高校在培养软件人才方面应当应势所趋，重视编程语言的教育进而提升未来程序员的编程素质、基本功和对编程的兴趣，这对于整体提高处于信息化进程中的社会生产力大有裨益。随之而来的是软实力的质变，并一定会同时酝酿新技术的革新，因为基数和兴趣都得到提升，概率便不会让创造力失望。

简而言之，更好的编程语言教育可以切实地提高软件生产力，同时培育创新的沃土。例如，在大数据和智能计算的驱动下，医疗、城市建设、特种装备等很多行业都会利用自己领域内的数据进行一番智能变革，这给软件创新带来新的机遇。程序员应该有能力有信心意识到，他们可以创造或调制适合该行业领域应用的专属编程语言，使行业专家可以很方便地使用语言编程，满足各种特定的可靠性、性能甚至知识产权等需求。这给新兴软件应用行业的创新提供了丰厚的技术资本，这些应用行业本身又会刺激市场，为计算机行业创新带来新的需求和机遇。

综上所述，重视并提高编程语言教育质量对个人和社会的发展都大有裨益。但是如何提高编程语言教育质量？最直接有效的方法是教授好编程语言课程。

如何教授编程语言课程？

首先，这个问题没有标准答案，每个人心中都有自己的看法。其次，很多教授编程语言的教师和有长时间编程经验的程序员都比我更懂编程语言。一番自我觉悟后，我仍想结合国内高校编程语言教育的一些实际情况，阐述一下关于“应该如何教授编程语言课程”的几点**个人看法**。

脉络和整体的把握

一门比较好的编程语言课程的授课脉络可以由简入深地“成长”出一门编程语言。这个“成长”不是基于具体编程语言的语法特征，而是基于语言的基本概念。换言之，教授一门真正意义的编程语言课（尽管这门课指定了一门需要掌握的编程语言），主要是为了让学生学习隐藏在这门具体语言背后的、支撑它的理论知识，而这门语言的具体的语法语义特征是为了理解这些编程语言知识。

例如，编程语言可以从数据抽象开始，通过表达式来表达基本的“逻辑”和“运算”，引入条件和循环对其“控制”，引入函数对其“封装”，函数的封装需要作用域，函数的调用需要上下文等概念来支撑；再抽象一点，命令式编程语言抽象数据，用变量绑定，用状态存储；函数式编程语言既抽象数据又抽象行为，因此可以通过表达式传递“计算”，而无需状态存储；更抽象一点，面向对象编程语言有了状态和行为的统一封装，引入了类和继承等基本概念，而后者又需要方法覆写、属性隐藏等概念支撑等。如此一来，在一点点讲解这些编程语言知识的同时，用特定的语法、语义来具体解释这些概念和知识。如此，学生不但能更深刻地掌握一门具体的编程语言，也能了解编程语言设计的整体和核心，这是举一反三的基础，更是兴趣和自信的源泉。

当然，有的教师可能愿意把整个编程语言安排成初级和高级两门课程。在初级课程中讲授一门具体语言，在高级课程中专门讲解编程语言的理论知识，这也是一个合理的设计。但我个人更偏好将二者放在一起讲，因为理论和实践双管齐下，能刺激

学生更强的求知欲。原因很简单，理论需要实践来解释“如何”，实践需要理论来支撑“为何”。一个学生如果能同时知道“如何”和“为何”，没有理由不愿意了解更多，虽然最后对编程语言有没有兴趣是学生自己的事，但是是否引导了学生的兴趣是教育的事。

教授被忽略的重要编程语言知识

国内的编程语言课程更侧重对某一门具体语言的讲解，而缺少对编程语言整体设计的理解和判断。这好比学人体素描不识人体骨骼构成，学中医不讲阴阳，学音乐不知乐理。如此一来，知识的理解受限且无法被融会贯通，不可深究亦不能长远。接下来我用国内编程语言课中经常被忽略的两个重要的编程语言知识点“函数式编程”和“语言的静态动态类型”作为例子简单说明一下。

函数式编程

国内编程语言课程接触到的一般都是命令式语言（imperative），例如 C、Java、C++ 等，其中面向对象（Object-Oriented，OO）的理论也有所普及。很多人听说过函数式语言（functional programming languages），但是并不真正了解。因为现在毕业生找工作时用人单位并没有强制要求求职者了解函数式编程，所以学生觉得学不学无所谓。

实际上了解函数式编程的思维更有助于理解用代码“抽象”和“计算”的概念。例如，如果普通数据值（value）可以抽象，那为什么行为（behaviour/action）不能抽象？我们可以把行为（用函数 function 来表达）看成像数据一样的值（一等公民 first-class citizens），这样一来，传入函数的参数既可以是普通数据值，也可以是行为（函数）本身，进而引出高阶函数（higher-order function）的概念。这使得函数可以通过其“函数参数”来“多态”行为，而一次计算任务就可以看成一套“行为”的组合（函数的传递和调用），这样在实现一些计算任务时，代码会更简洁，逻辑会更清晰。

如今“函数式编程”被应用在越来越多的主流面向对象编程语言中。例如编程一霸 JavaScript 广泛支持了函数式编程机制；企业级代码新秀 Scala 一直以面向对象与函数式的“完美”结合标榜自己；就连面向对象编程语言的“代言人”Java 在 Java 8 后也开始通过“糖化”语法等操作来支持函数式编程（如 λ 表达式）。对于一些纯函数式语言，如 Haskell，已经成为很多特殊行业（如金融）的宠儿。因此，教授函数式编程（或思维）对编程语言的理解和实践都是有益的。

语言的静态动态类型

有时编程语言可以通过“静态语言”和“动态语言”进行分类，如 Scala、Haskell 属于前者，JavaScript、Python 属于后者。因此，有必要讲讲为什么会有静动之分，为什么不能各自取代对方，这对于理解编程语言的设计初衷和面对不同编程任务如何选择语言，都是有帮助的。

静态语言定义变量需要声明类型，调用函数需要匹配类型，而动态语言一般没有类型声明和匹配的约束。前者的程序在运行前一般需要编译器静态检查这些约束，而后者的程序可以直接在解释器上动态“裸奔”。各自的好处容易理解，静态语言虽然需要程序员费时费力照规矩办事，但面对复杂系统时能大量减少运行时错误，写出的程序也容易理解和便于分析；动态语言解放编程者天性，没有太多约束，可直接运行，轻便省力，但更容易导致运行时错误，且代码一般只有程序员自己当天能读懂。这也解释了为何复杂的企业级软件更青睐于静态语言，而脚本或轻量级程序一般选择可以迅速上手的动态语言。

理解了这些，也就更容易理解编程语言为何有时“动中向静”（JavaScript 和 TypeScript，动态语言静态化），有时“静中有动”（Java 和 Reflection，静态语言通过 metaobject 打开动态的缺口），有时又“静动兼修”（Gradual typing）。

拓展一些常见的编程语言知识

有时拓展讲解一些常见的编程语言知识会更有趣且有启发性。我以 Java 语言为例举两个简单的例子：继承和异常处理。

继承

面向对象编程语言课程都有针对继承的讲解（毕竟面向对象的三大核心是封装、继承、多态），比如 Java 中类和父类的继承关系以及继承带来的好处。但是我们往往有一种倾向，就是觉得存在即合理，因此觉得继承一定就是好的，进而一些教师通常并不讲解 Java 中基于类的继承的弊端。实际上 Java 基于类的继承方式被很多编程语言学者吐槽过，因为它在很多情况下会造成不必要的冗余和开销（打个比方说，你和你父亲的基因很多是一样，但是不代表你俩穿的衣服都是一样的）。实际上很多继承关系是可以被更轻量级的组合（composition）代替的，比如 Mixin 技术（虽然不像 Scala、Ruby 等语言在语言层面原生态支持 Mixin，但 Java 在 Java 8 后通过接口默认方法（interface default methods）也可以间接实现 Mixin）。了解了这些知识，就可以更容易理解其他继承设计的初衷和意义了，比如 JavaScript 这种基于 prototype 的更轻便的继承方式。

异常处理

说到异常处理，最直观的感受是编程语言提供的一种处理错误的机制。很多程序任务都不可避免地要处理各种可预判的错误，例如访问的文件不存在、取值的对象为空等等。在异常处理机制下，错误可以标准化也可以定制化，不同的错误可以根据其类型通过 try/catch 捕捉处理。在此基础上教师如果能再拓展讲一点，学生也许能从更抽象的层面了解编程语言背后的设计哲学。这一点就是“异常处理”背后的“逻辑分流”设计。程序本身有以期待的输入进行任务计算完成输出的“主体逻辑”，然而这一过程不可避免地要处理“错误逻辑”。如何让程序以及编程者的主体逻辑思维不受错误逻辑干扰是异常处理机制设计的一个重要初衷，代码也会随之简洁易懂。有了对“逻辑分流”的理解，学生看到诸如 JavaScript 的 Promise 这样的特性（resolve 和 reject 分流），就会更容易理解该特性设计的初衷，并愿意在合适的编程场合使用该特性。

综上所述，我觉得教师对一些基础知识（如“继承”“异常处理”等）进行拓展讲解，能让学生更容易跳出某个语言本身，更容易站在编程语言设计层面去思考，进而将知识融会贯通形成自己的知识树，甚至有可能进一步思考各种新的有趣的编程语言的雏形，这种兴趣的代入和思考的启发应该是教育的重心。我也相信这种“由内而外”的“漫天想象”日后会带来很多具有生命力的创新。

对于语义的重视

编程语言的语义非常重要，学习好语义可以从本质上理解编程语言，因为它教你了解运行程序的解释器是如何工作的。很多通用编程语言用很长的规范来具体描述语言的语法、语义和特性，以及它们应该被如何使用，市面上的很多编程语言书籍也是参考这些规范文档撰写的。然而无论是官方的规范手册还是书籍都用自然语言描写，这会产生很多歧义。这也意味着即便通过看书学习了一个个语法特性尝试编程后，也很难对语言有一个整体的了解，对于语言局部的细节也很难有准确的把握。

在这种背景下，编程语言的形式化语义显得尤其重要，它通过一套基于数学的形式化描述，利用语法定义其相应的语义。需要注意的是，因为实际的通用编程语言很复杂，很难形式化准确描述其所有的特性语义，如此一来整个语言及其使用的环境并不能够得到严谨准确的描述和证明，可以说如今的编程语言包括编译器和运行环境实际上都有 bug，只是很多都还没有暴露而已。因此，形式化语义一般只描述一个编程语言的核心部分，但这些核心部分已经可以为编程人员构建出该编程语言的一个整体设计面貌。编程语言领域语义的研究有较长的历史，很多国外顶级大学的编程语言课程都有对形式化语义的专门讲解，描述语义的方法也有很多（如指称语义、操作语义等）。

尽管编程语言语义很重要，但我国很多高校的编程语言教育忽略了这一块，只有为数不多的高校在编程语言课上会单独讲授包括语义在内的语言理论知识，例如北京大学的王捍贫、熊英飞和南京大学的梁红瑾、冯新宇等授课教师。

实际上，如果不是单独的编程语言理论课程，

就不用全面细致地讲解它们。例如，如果编程课程教授的语言是 Java，可以考虑从整个 Java 核心部分的语法和语义中挑选一种，比如操作语义中的小步状态机方法来描述，并讲解每个核心语法和其对应的语义，这样学生在掌握描述语法、语义方法的同时，也从整体和核心细节上理解了 Java，这非常有益于日后更好地理解编程语言和编写程序。

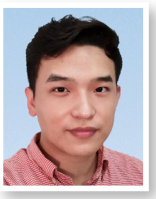
当然，我前面说的这么多内容很多时候不可能被一门单独的课程全部覆盖到，不同高校可以结合自身的实际情况，选择性地为学生提供不同的编程语言课程。

总结

互联网是一个挺好的地方，虽然让人感受到很多浮躁，但仍然能在互联网上看到人们对于知识的渴望和对进步的渴求，尤其是那些身处更新速度极快的计算机领域中的学生和程序员们。鉴于自己在国内受过的编程语言教育和在互联网上看到的形形色色的编程语言问题，我切实感受到了我国编程语言教育相对落后，因此写下这篇文章，希望让更少的学生被惋惜，让更多的学生受鼓舞。我虽资质平平，但仍想借此文发声 ：我国高校主管计算机和软件教育的领导们，各计算机学院软件学院的院长们，人工智能、物联网、大数据等计算机应用固然重要，但请不要忽视计算机基础、软件核心“ 编程语言 ”的教育和研究，毕竟所有应用软件都是用编程语言写的，毕竟如今的计算机行业对程序员的“ 质 ”的要求越来越高，毕竟我们有这么多对编程感兴趣、渴望编程知识的准程序员们。期待我国编程语言教育和科研的进步。

后记

对于“ 国内编程语言教育 ”这个话题，即使是“ 浅谈 ”，我也觉得这个题目很大。我自知对编程语言的理解还很欠火候，也真心觉得自己的编程能力与有很多工作经验的程序员相差甚远，授课能力与经验相比于很多教授编程语言的教师也相差很多，因此每每下笔，都怕是妄谈，对文中每个观点都尽量客观地呈现自己所见所想。尽管人微言轻，但我还是希望这篇文章能引发软件行业的学生们和程序员的一点思考，也希望能让那些关心学生和计算机教育的相关领导看到，真正对改变我国编程语言教育的现状有所帮助。 ■



李 樾

CCF 专业会员。南京大学计算机科学与技术系副教授，博士生导师。主要研究方向为程序设计语言与程序分析。

yueli@nju.edu.cn

（本文责任编委：翟季冬 许 嘉）

2022年度“CCF-华为胡杨林基金-形式化专项”评审结果

“ CCF– 华为胡杨林基金 – 形式化专项 ”分为开放课题和产业课题。中国计算机学会（CCF）形式化方法专业委员会与华为于 2021 年 12 月发布了 5 个产业课题方向，开放课题方向不限定具体研究内容，主要资助具有前瞻性、前沿性、能为产业全面升级储备能力的课题。该基金自发布以来，受到了国内科研学者的广泛关注，共收到有效项目申请 25 份，其中产业课题方向 15 份，开放课题方向 10 份。经专家组评审，共有 5 个产业课题项目和 5 个开放课题项目获得资助。

扫码查看评审结果