

# 面向自动驾驶的C++实战

## 第一章 课程介绍

主 讲：姜朝峰

公众号：自动驾驶之心

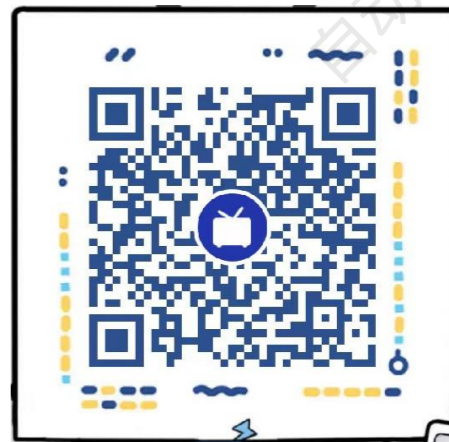
# 自动驾驶之心

## 全栈知识矩阵

自动驾驶之心，专注自动驾驶与AI



公众号  
干货每日放送



B站  
不定期直播+最新视频



知识星球  
海量图文教程和Paper分享



视频号  
技术视频一站式分享

# 主要内容

- ① 主讲人介绍
- ② 为什么自动驾驶选择了C++？ 不同编程语言的比较
- ③ 如何有效学习C++？ 编程语言和自然语言比较
- ④ 面向群体和学后收获
- ⑤ 讲解思路与课程大纲

本资料由：dongqueba.com收集整理



## 主讲人介绍

1. 本硕就读于浙江大学，硕士毕业于浙江大学机器人实验室
2. 曾多次获取机器人世界杯RoboCup小仿人组冠/亚军
3. 曾在华为智能汽车解决方案BU、安途智行AutoX工作
4. 八年C++经验，熟悉自动驾驶中间件、自动驾驶仿真等



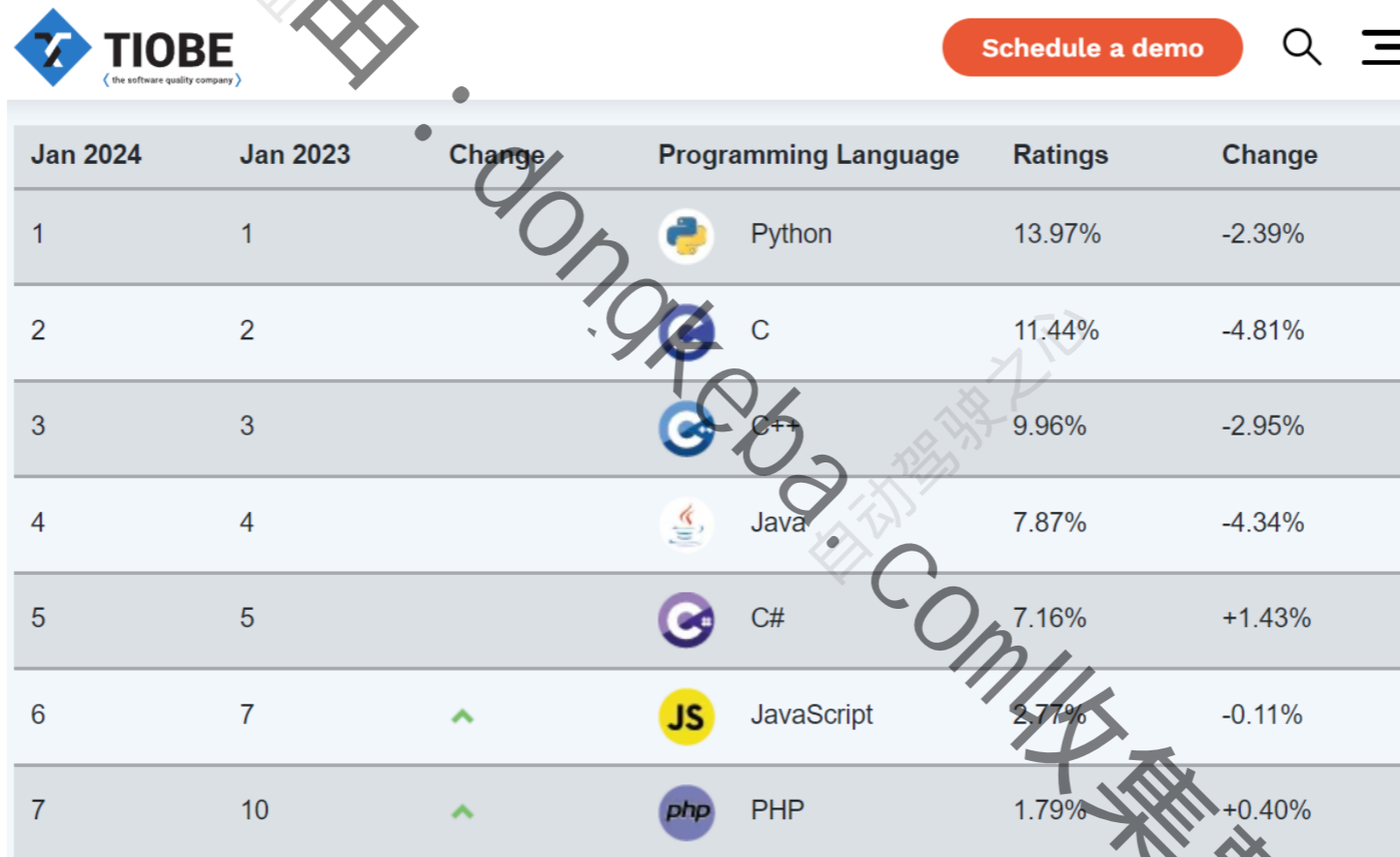


# 为什么自动驾驶选择了C++? 不同编程语言的比较








# 1. 为什么自动驾驶选择了C++？不同编程语言的比较

## 1.1 主要编程语言

热门语言排行：编程语言排行榜网站TIOBE会定期更新编程语言排行榜。



The screenshot shows the TIOBE Programming Language Index website. The header includes the TIOBE logo, a 'Schedule a demo' button, and search and menu icons. The main content is a table with 7 columns: Rank (Jan 2024), Rank (Jan 2023), Change, Programming Language (with icon), Ratings, and Change. The table lists the following languages: Python (Rank 1, 13.97%, -2.39%), C (Rank 2, 11.44%, -4.81%), C++ (Rank 3, 9.96%, -2.95%), Java (Rank 4, 7.87%, -4.34%), C# (Rank 5, 7.16%, +1.43%), JavaScript (Rank 6, 2.77%, -0.11%), and PHP (Rank 7, 1.79%, +0.40%).

Jan 2024	Jan 2023	Change	Programming Language	Ratings	Change
1	1		 Python	13.97%	-2.39%
2	2		 C	11.44%	-4.81%
3	3		 C++	9.96%	-2.95%
4	4		 Java	7.87%	-4.34%
5	5		 C#	7.16%	+1.43%
6	7	↑	 JavaScript	2.77%	-0.11%
7	10	↑	 PHP	1.79%	+0.40%

TIOBE index

<https://www.tiobe.com/tiobe-index/>



# 1. 为什么自动驾驶选择了C++？不同编程语言的比较

## 1.1 主要编程语言

### 编程语言分类

编译语言、虚拟机语言和解释型语言是编程语言的三种不同类型，它们在代码执行的方式上有所不同。

#### 1. 编译语言（Compiled Language）

- 例子：C、C++等。
- 编译过程：编译语言的源代码在运行之前需要通过编译器转换为机器码或字节码，编译器会一次性将整个源代码转换为目标代码。
- 执行方式：编译生成的目标代码可以直接在计算机上运行，而无需再次进行翻译。这通常意味着编译语言的执行速度相对较快。

#### 2. 虚拟机语言（Bytecode Language）

- 例子：Java、C#（通过Java虚拟机和.NET虚拟机执行）。
- 编译过程：虚拟机语言的源代码会先被编译成中间代码（字节码），而不是直接编译为机器码。
- 执行方式：中间代码在运行时由虚拟机解释执行，或者被即时编译为目标机器码执行。这种方式结合了编译语言和解释型语言的特点。



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.1 主要编程语言

### 编程语言分类

#### 3. 解释型语言 (Interpreted Language)

- 例子: Python、JavaScript等。
- 执行方式: 解释型语言的源代码是逐行或逐语句被解释器执行的, 而不需要通过编译步骤生成目标代码。解释器在运行时直接对源代码进行解释执行。
- 优点: 灵活性较高, 代码可以在不同平台上直接运行, 无需事先编译。
- 缺点: 执行速度通常较慢, 因为每次运行都需要解释代码。

总体而言, 编译语言、虚拟机语言和解释型语言这三种类型的语言在执行方式、擅长领域上存在差异, 选择使用哪种类型的语言通常取决于项目需求、性能要求和开发者生态。



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.1 主要编程语言

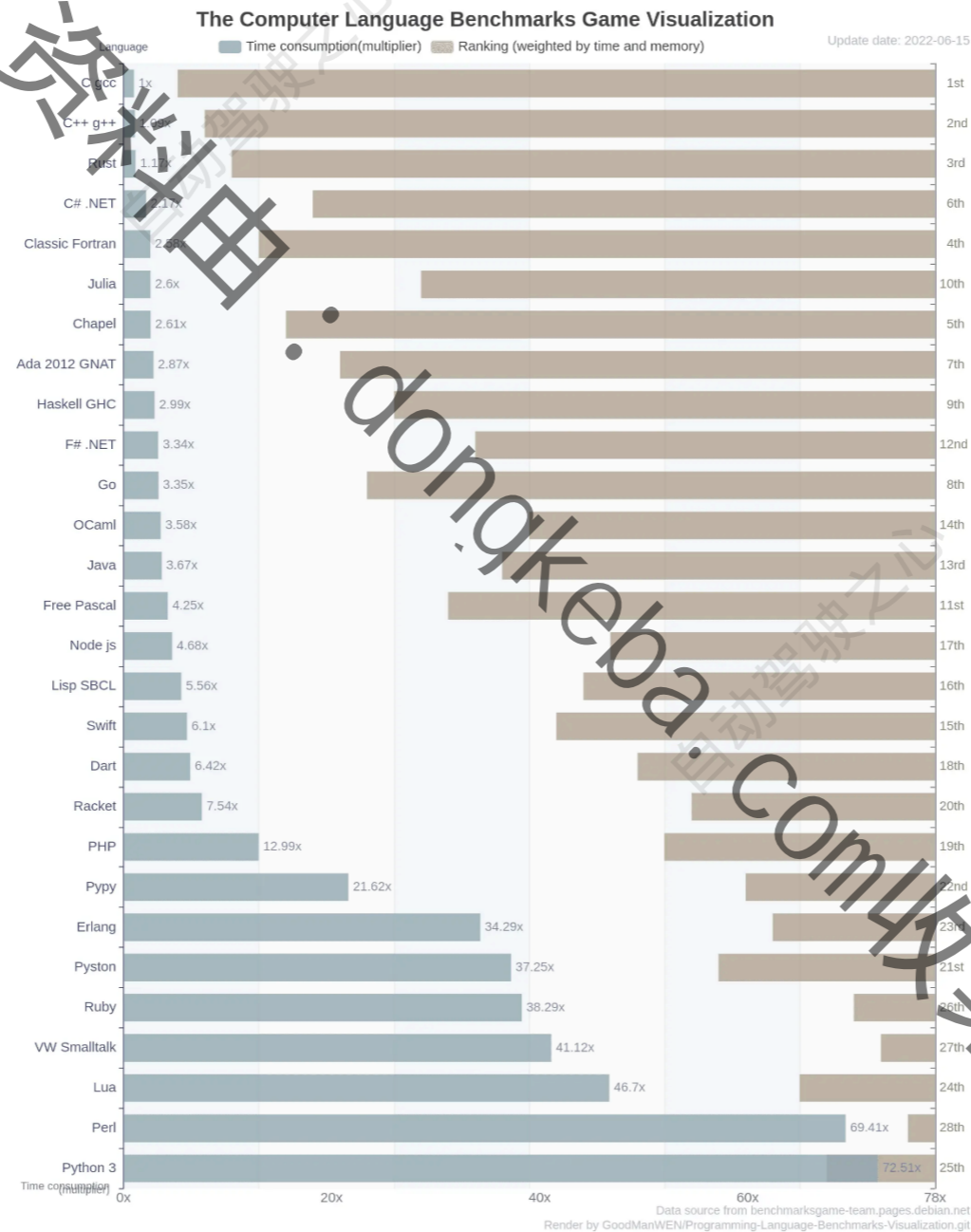
### 语言特点分析

1. Python是一种解释型语言，这意味着Python代码不需要预先编译，而是可以逐行解释执行。这种解释性质使得Python更灵活，但也导致了性能上的较大牺牲。
2. C是一种过程式语言，主要关注过程和函数。通常用于嵌入式系统、操作系统、系统级编程等对性能要求较高的领域。
3. Java是一种先被编译成字节码、然后在Java虚拟机上运行的编程语言，并可以自动进行GC（垃圾回收）。这使得Java具有跨平台的特性，但也引入了一些性能开销。Java是一种跨平台的面向对象语言，适用于大型企业级应用。
4. C#是一种面向对象的语言，适合构建Windows应用和Web应用。C#最初是为Windows平台设计的，因此在其他平台上的支持相对较弱，生态较弱。
5. JavaScript和PHP通常用于Web开发、互联网应用，在Web领域有很强大的生态，除此之外却用处不多。都是弱类型语言，可能导致一些类型相关的错误，难以调试和理解。
6. Matlab: MATLAB脚本语言在数学建模、科学计算和工程应用方面具有强大的功能，但这都基于MATLAB软件生态。在一些方面，如开源性、通用性和性能方面，则存在很大限制。



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.2 性能对比



Programming-Language-Benchmarks-Visualization



www.zdjszx.com

# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.2 性能对比

**Table 4.** Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Normalized global results for Energy, Time, and Memory



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.2 性能对比

平均而言，编译语言花费 5103 毫秒，虚拟机语言花费 20623 毫秒，解释型语言花费 87614 毫秒。

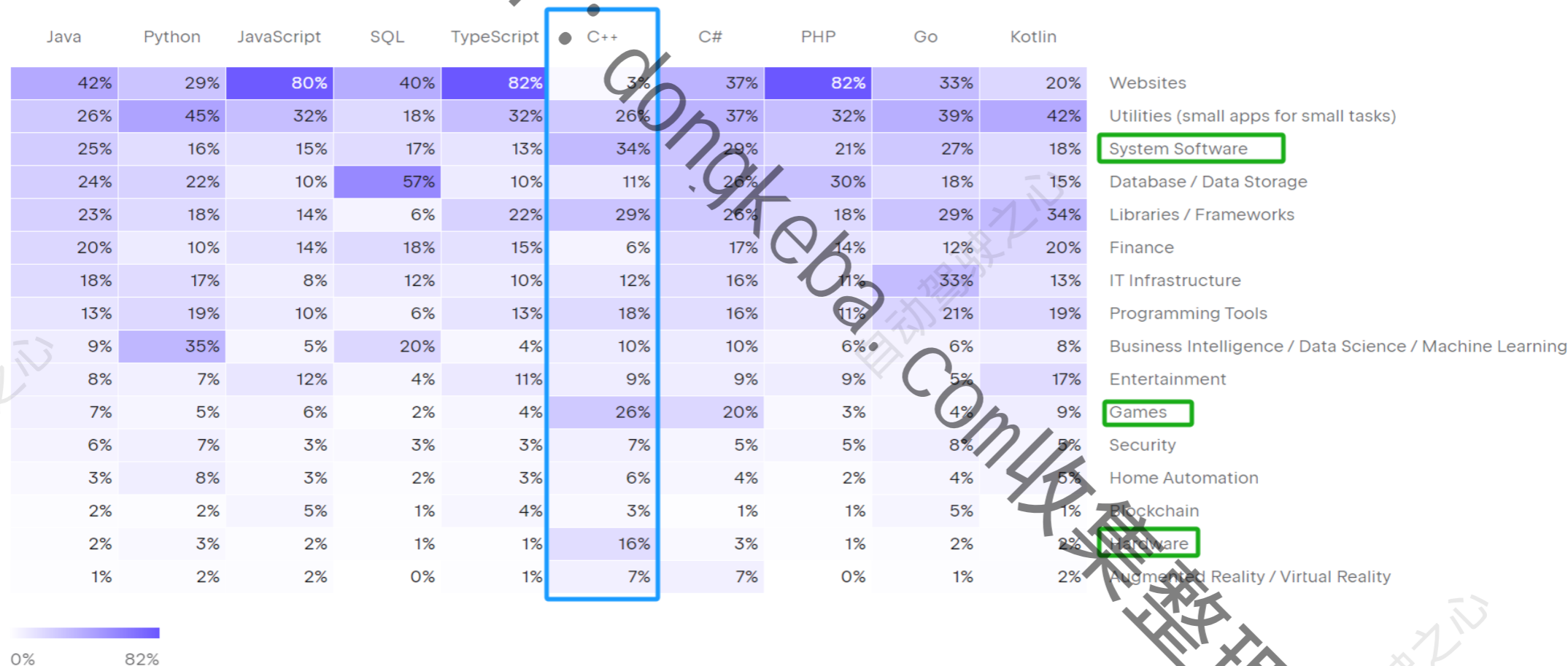
平均而言，编译语言需要 125Mb，虚拟机语言需要 285Mb，而解释语言则需要 426Mb。



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.3 擅长领域

What types of software do you develop with your main languages?



# 1. 为什么自动驾驶选择了C++？不同编程语言的比较

## 1.4 C++的优势

- 性能和效率：** C++在对计算资源要求严格的自动驾驶系统中表现出色，能够提供卓越的性能和响应速度。
  - 实时系统支持：** C++是一种高性能的编程语言，它直接编译成机器码，可以直接运行。这使得C++对实时系统提供了良好的支持，能够满足自动驾驶系统对快速响应的需求。
  - 系统级编程：** C++是一种系统级编程语言，能够直接访问硬件和进行底层操作。这对于处理自动驾驶中的传感器数据、执行控制算法和与硬件交互非常重要。可以精确控制代码的内存占用、CPU等资源的分配，确保自动驾驶系统资源合理分配。
  - 并发性和多线程支持：** 自动驾驶系统通常需要处理大量的并发任务，例如同时处理感知、决策和控制。C++提供了强大的并发性和多线程支持，使得开发人员能够有效地处理这些复杂的并发任务。
- 强类型检查：** 强类型检查意味着在编译时对数据类型进行更严格的检查，可以在编译时捕获到许多潜在的类型错误，避免了在运行时发生类型不匹配的问题。这有助于提高代码的稳定性和可靠性，代码可读性也更高。
- 庞大的生态系统：** C++拥有庞大的生态系统和丰富的库支持，这使得开发人员能够更轻松地构建和维护复杂的自动驾驶系统。许多自动驾驶相关的开源库和工具都是用C++编写的。
- 成熟的工程实践：** C++是一种经过多年发展和验证的编程语言，拥有丰富的工程实践和设计模式。这有助于构建可靠、健壮且易于维护的自动驾驶系统。有众多成熟的编译器和IDE，能提高自动驾驶软件的开发效率。



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.5 最终决策

从研发部门的角度: 研发效率、生态 -> 否决C、Javascript

从车企OEM的角度: 便宜 -> 否决C#、Java

从用户的角度: 快速流畅、稳定性 -> 否决Python

最终答案: C++

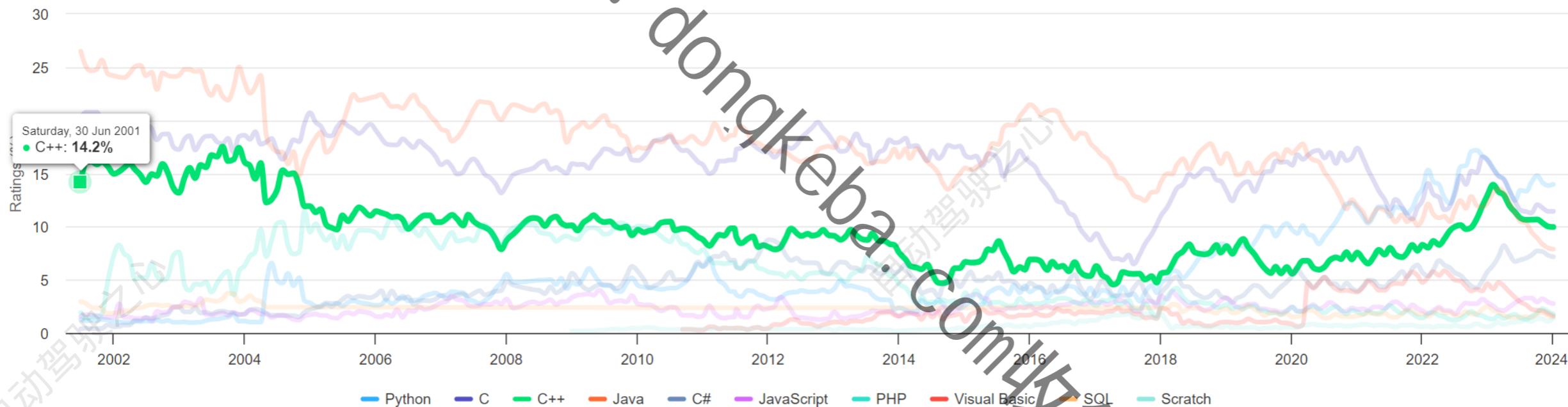


# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.6 发展趋势

TIOBE Programming Community Index

Source: www.tiobe.com



<https://www.tiobe.com/tiobe-index/>



# 1. 为什么自动驾驶选择了C++? 不同编程语言的比较

## 1.6 发展趋势

《反脆弱》书里介绍了一个林迪效应:

- 对于会自然消亡的事物, 生命每增加一天, 预期剩余寿命就会缩短一些。比如, 一般年轻人比长者的剩余寿命要长;
- 对于不会自然消亡的事物, 生命每增加一天, 意味着更长的预期剩余寿命。也就是说, 如果老技术的寿命已经有 80 年, 新技术的寿命有 10 年, 那么老技术的预期剩余寿命将是新技术的 8 倍。

这其实是进化论和马太效应的一种体现。



# 1. 为什么自动驾驶选择了C++？不同编程语言的比较

## 1.6 发展趋势

### Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2024	2019	2014	2009	2004	1999	1994	1989
Python	1	4	8	6	11	22	22	-
C	2	2	1	2	2	1	1	1
C++	3	3	4	3	3	2	2	3
Java	4	1	2	1	1	16	-	-
C#	5	6	5	8	9	32	-	-
JavaScript	6	8	9	9	8	21	-	-
Visual Basic	7	19	-	-	-	-	-	-
PHP	8	7	6	5	6	-	-	-
SQL	9	9	-	-	7	-	-	-
Assembly language	10	13	-	-	-	-	-	-
Objective-C	27	11	3	42	48	-	-	-
Lisp	30	28	14	17	15	10	7	2
(Visual) Basic	-	-	7	4	5	3	3	7

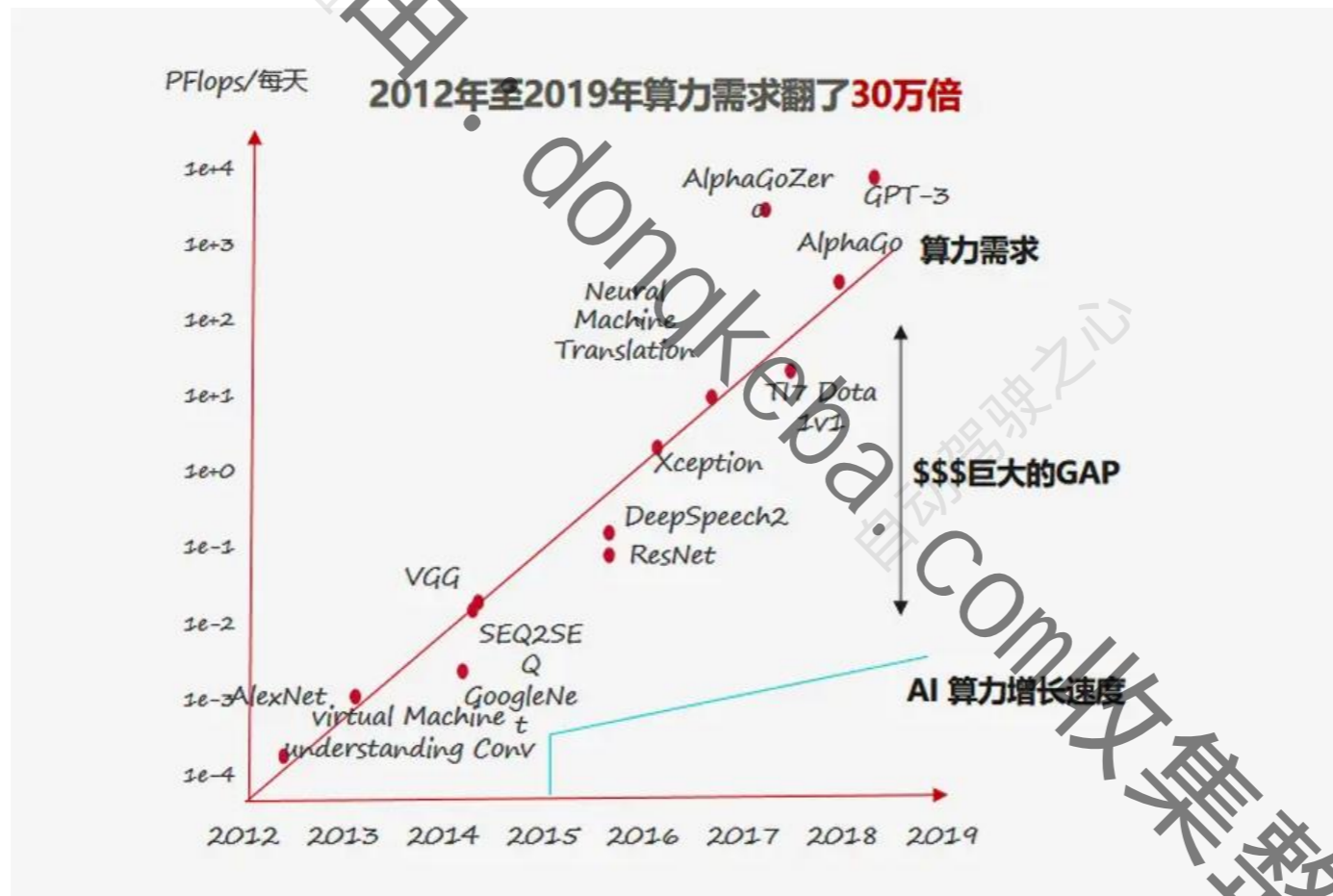
编程语言里的常青树——C/C++



# 1. 为什么自动驾驶选择了C++？不同编程语言的比较

## 1.6 发展趋势

在大模型热火朝天的今日，C++会伴随着世界对算力、高性能的需求而继续焕发绵长的生命力。



AI算力需求爆炸式增长



2

## 如何有效学习C++? 编程语言和自然语言比较

## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.1 语言的历史演变

#### 英语的历史

- 公元450年左右, 当时盎格鲁-撒克逊人入侵/移民到不列颠群岛, 在那里形成了早期的英语。
- 随着北方的维京人的到来, 英语逐渐受到了北欧语言的影响, 形成了中古英语。
- 在1066年, 法国的诺曼底公爵夺取英格兰王位后, 法语和拉丁语词汇被引入英语, 对英语的语法和词汇产生了深远的影响。
- 到了近代的百年英法战争和文艺复兴时期, 英语逐渐规范化, 形成了现代英语的雏形。
- 工业革命后, 随着大英帝国的扩张, 英语逐渐传播到世界各地。
- 20世纪之后, 美国人又给英语进行了很多简化、修改, 并且和流行音乐、美国大片一起, 让英语为全球最广泛使用的语言。
- 在现代, 英语的发展仍然不断, 新词汇、外来语和网络语言的兴起都为英语的发展注入了新的活力。



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.1 语言的历史演变

#### C++的历史

- 1972年, Bell实验室的Dennis Ritchie在B语言的基础上, 共同发明了C语言, 并和Ken Thompson使用C重写Unix。
- 1979年, Bjarne Stroustrup到了Bell实验室, 开始从事将C改良为带类的C (C with Classes) 的工作, 1983年正式命名其为C++(CPlusPlus), 意图表明C++是C的增强版。
- 从1985年到1998年, C++从最初的C with Classes新增了很多其他的特性, 比如异常处理、模板、标准模板库 (STL)、运行时异常处理 (RTTI) 与名字空间 (Namespace) 等。
- 1998年, C++标准委员会统筹C++的所有特性, 发布了第一个C++国际标准C++98。
- 2003年, C++标准委员会总结最新技术并发布了C++03标准。
- 从2003年到2011年, C++长期停滞不前。到了C++11, 终于引入了移动语义、右值引用、lambda表达式 (函数式编程)、编译时类型识别 (auto) 等现代编程语言常具备的能力, C++11让C++与时俱进, 开发效率得到了很大的提升。因此C++11以及之后的版本也被称为Modern C++。
- 近年来, C++标准的更新周期缩短并保持稳定, 由C++11到C++14、C++17、C++20、C++23都保持着3年一更新的节奏, 也保持了C++的活力。



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.2 如何更好地学习

#### a. 确立目标：语言的本质是为了更好地沟通和表达

- 自然语言的学习，是为了人与人之间的沟通；
- 编程语言的学习，是为了让人控制计算机完成具体的任务。

在学习过程中，不能脱离使用场景和目的。每个语言特性，都应该对应具体场景需求。

#### b. 语言的入门其实很简单

- 汉语的语法其实很简单，入门所需要学习的汉字很少。掌握很少文字的小孩子，也可以尽情地表达和沟通；
- 创建一个变量、读取一个文件、打印“hello world”，就可以算作C++入门。

语言由人发明、并面向大多数人，入门一种语言并不难，所以不必畏惧。



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.2 如何更好地学习

#### c. 语言的精通其实挺难

- 历史长、积累多的语言，很难达到精通。比如，想要熟练掌握汉语是很难的，你需要深入掌握俗语、成语、近义词、一词多义、量词、网络流行语等，精通更是要读大量的古今典籍，其中包含了大量的历史、文化等知识背景；
- C++包含了很多特性和范式，非一朝一夕之功。所以学习者不要抱我短期可以将一门语言的细节全部掌握的想法。

#### d. 在实践中学习

- 学习说话，需要和家人反复沟通、纠正错误；
- 学习代码，需要和编译器、操作系统反复沟通，纠正错误；和同学、同事互相code review。



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.2 如何更好地学习

#### e. 复杂特性需要反复练习以形成肌肉记忆

- 背诵名家诗篇，以记住优美的遣词造句。交流和阅读文学作品可以提高语言的表达能力和理解能力。
- 阅读和运行经典代码，以熟悉精妙的高级用法。参与C++社区、阅读、调试他人的代码是学习编程最佳实践和不断提升的有效途径。



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.3 学习资料推荐

#### 入门

我非常不推荐大家一上来就开始读《C++ Primer》、《C++编程思想》等大部头著作，其包含的内容非常详细，上千页的长度很容易让新手产生畏惧感，容易让人想放弃。我的建议不是从这些出名的著作开始读起，反而是应该找入门级别的小册子、网站：

- <https://www.runoob.com/cplusplus/cpp-tutorial.html>
- <https://github.com/changkun/modern-cpp-tutorial>
- <https://github.com/Light-City/CPlusPlusThings>
- <https://github.com/GrindGold/CppGuide/blob/main/file/%E5%85%A5%E9%97%A8%E6%95%99%E7%A8%8B%E9%98%BF%E9%87%8C%E4%BA%91%E5%A4%A7%E5%AD%A6.md#c-%E6%95%99%E7%A8%8B>
- [https://github.com/tomstillcoding/cs-roadmap/blob/main/c++/2022\\_%E6%9C%80%E6%96%B0C++%E5%BC%80%E5%8F%91%E5%AD%A6%E4%B9%A0%E8%B7%AF%E7%BA%BF\\_%E7%A7%91%E7%8F%AD%E7%89%88.md](https://github.com/tomstillcoding/cs-roadmap/blob/main/c++/2022_%E6%9C%80%E6%96%B0C++%E5%BC%80%E5%8F%91%E5%AD%A6%E4%B9%A0%E8%B7%AF%E7%BA%BF_%E7%A7%91%E7%8F%AD%E7%89%88.md)



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.3 学习资料推荐

#### 精进

1. 《C++ Primer》 - 被誉为C++的圣经，非常适合初学者作为C++的工具书和复习教材。
2. 《C++编程思想》 - 侧重介绍C++的编程思想和设计模式，适合有一定编程基础后进一步提高的学习者。
3. 《Effective C++》 - 主要介绍C++编程中的55个经验准则，如何写出简洁高效的C++代码。适合已有一定C++基础的程序员读。
4. 《STL源码剖析》 - 分析探讨C++标准模板库的实现原理，适合想深入理解STL的进阶者。

#### 手册

[www.cplusplus.com](http://www.cplusplus.com) 排版更好，条理分明

[zh.cppreference.com](http://zh.cppreference.com) 内容详细、更新及时



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.4 不仅仅是一门语言

语言的深入学习伴随着领域的专业化:

- **自然语言:** 国学大师不懂媒体文案, 法律专家不会写网络小说。到一定阶段, 语言只是从事某行业的工具。
- **编程语言:** 操作系统、流媒体、Linux嵌入式、自动驾驶虽然都用C++, 但基本属于不同的行业。

学习一种编程语言的过程, 不仅仅是在学习语言本身, 还需要学习这种语言擅长的领域:

- 学习Java的同时, 你还需要学习Web应用架构、数据库管理、SpringBoot框架等;
- 学习Python的同时, 你也需要学习数据挖掘、数据可视化、机器学习等领域知识;
- 学习C++的同时, 你也需要学习操作系统和内存管理、计算机网络、硬件体系架构、性能优化、GPU编程等。



## 2. 如何有效学习C++? 编程语言和自然语言比较

### 2.4 不仅仅是一门语言

技术的数据结构类型是**树**，即技术也是由子技术组成。

学懂主要的语言特性、编程范式之后，应当根据你以后想就业或钻研的领域，围绕某个具体方向去学习，不要杂而不精，迷失在技术的森林中。

毕竟，语言最终只是工具，完成某个领域的工作任务，才是我们真正的目标。



3

面向群体和学后收获

## 3. 面向群体和学后收获

### 3.1 面向群体

本课程面向对自动驾驶技术感兴趣、希望未来用C++编程语言进行自动驾驶研发的同学和工程师。

适合学习本课程的人群包括：

- 在校的本科/硕士/博士
- 正在从事自动驾驶、但不熟悉C++的算法工程师
- 熟悉C++、想跨行进入自动驾驶的研发从业人员
- 工作上需要提升技能的、其他方向的工程技术或管理人员
- 对C++编程和自动驾驶技术感兴趣的小伙伴

#### 前置要求

- 具有一定的计算机基础，最好掌握至少一门编程语言
- 最好具有理工科背景，一定的高等数学基础；
- 对自动驾驶有一定了解。



## 3. 面向群体和学后收获

### 3.2 学习收获

1. 掌握C++语言的基础知识和高级特性；
2. 掌握代码规范，能够写出较高质量的C++工程代码；
3. 具备用C++解决一定实际问题（量产级）的能力；
4. 具备一名具有1-2年经验的自动驾驶C++工程师；
5. 丰富案例和项目实战，加深理论和实际的融合，在研究工作或者算法工程上得到新的方法设计。
6. 认识多行业从业人员与学习合作伙伴，并在交流中达到更加深入的理解。

通过本课程的学习，可以帮助学生和工程师掌握C++在自动驾驶领域的实际应用，为从事自动驾驶研发工作奠定良好的软件基础。



4

## 讲解思路与课程大纲

## 4. 讲解思路与课程大纲

### 4.1 讲解思路

#### 以问题为出发点

- 问题比结论更重要
- 不以记忆知识点为目标，以分析实际问题为出发点
- 引导大家主动提出问题，思考如何解决实际问题

#### 以代码实战为抓手

- 课程会提供大量的实例，并直接演示编写C++代码
- 每章节后面都设计实战作业，让大家动手实现讲解的特性
- 鼓励多动手实践，将理论和代码应用同步展开讲解

#### 紧密围绕自动驾驶领域

- 选择自动驾驶领域的应用场景和实际问题作为课程案例
- 基于自动驾驶实际工作的工程方法
- 使用自动驾驶仿真平台进行开发调试



# 4. 讲解思路与课程大纲

## 4.2 课程大纲

### 一. 课程介绍

- 1.1 为什么自动驾驶选择了 C++? 不同编程语言的比较
- 1.2 如何有效学习 C++? 编程语言和自然语言比较
- 1.3 面向群体和学后收获
- 1.4 讲解思路与课程大纲

### 二. C++基础

#### 2.1 开始搭建我们的 C++开发环境吧

- 2.1.1 Linux 和 VSCode 环境配置
- 2.1.2 代码如何从文本到可执行程序?
- 2.1.3 编译与链接编译工具
- 2.1.4 CMake 和 Bazel 构建工具介绍
- 2.1.5 构建自己的第一个 C++项目

#### 2.2 自动驾驶任务的类型和变量是怎么命名的?

- 2.2.1 什么是类型和变量
- 2.2.2 通用命名规范: 驼峰法和下划线法
- 2.2.3 文件、类型和变量的命名: 面向工程开发水准

#### 2.3 自动驾驶中不同任务的存储数据类型和结构介绍

- 2.3.1 基础数据类型: 以点云帧为例
- 2.3.2 复合和容器数据类型: 以目标检测为例

#### 2.4 C++基础语法速览回顾

- 2.4.1 输入和输出语法介绍
- 2.4.2 控制流的使用
- 2.4.3 命名空间的作用
- 2.4.4 详解函数
- 2.4.5 指针和引用到底是什么?
- 2.4.6 字符串的使用及扩展

#### 2.5 面向对象编程概述

- 2.5.1 什么是面向对象编程?
- 2.5.2 面向对象的优势有哪些?

#### 2.6 课程作业: 将点云帧关键信息输出显示

### 三. 深入面向对象编程

#### 3.1 类和对象: 经典的 C++概念

- 3.1.1 自动驾驶框架介绍
- 3.1.2 开始实现你的第一个自动驾驶模块吧!
- 3.1.3 导航模块类的定义
- 3.1.4 创建和使用导航模块实例
- 3.1.5 导航模块的构造和析构

#### 3.2 封装为什么如此重要?

- 3.2.1 数据封装的概念
- 3.2.2 为什么我的参数被修改了? 为成员设置访问权限
- 3.2.3 如何控制自动驾驶模块的运行: 成员函数的作用
- 3.2.4 从零开始实现你的全局导航任务吧

#### 3.3 继承: 一次编写, 重复使用

- 3.3.1 继承的基本概念
- 3.3.2 我想开发新算法: 派生类和基类介绍
- 3.3.3 提取基础算法模块: 虚函数和多态性详解
- 3.3.4 进一步实现你的行为预测模块

#### 3.4 多态性: C++中的一把利剑

- 3.4.1 不要重复造轮子: 静态多态性和动态多态性详解
- 3.4.2 设计自动驾驶系统框架: 抽象类和接口
- 3.4.3 自动驾驶中多态的应用场景详解

#### 3.5 STL

- 3.5.1 容器结构和迭代器详解
- 3.5.2 STL 算法详解
- 3.5.3 实战: 自动驾驶点云&图像数据处理

#### 3.6 其他 C++高级特性

- 3.6.1 模板详解
- 3.6.2 泛型编程
- 3.6.3 异常处理

#### 3.7 作业: 实现自己的自动驾驶软件框架



## 4. 讲解思路与课程大纲

### 4.2 课程大纲

#### 四. C++编程中的资源管理

##### 4.1 文件 IO 介绍

- 4.1.1 文件流的概念和基本操作
- 4.1.2 文本文件和二进制文件的读写

##### 4.2 内存管理：自动驾驶落地必备

- 4.2.1 进程内存模型
- 4.2.2 内存泄漏和内存溢出的概念
- 4.2.3 智能指针与 RAII 原则

##### 4.3 线程管理：多线程为什么如此重要？

- 4.3.1 多线程编程的基本概念
- 4.3.2 自动驾驶中为什么需要多线程？
- 4.3.3 线程的创建和销毁
- 4.3.4 线程同步与互斥机制

##### 4.4 GPU 管理与 CUDA 编程

- 4.4.1 GPU 资源管理和 CUDA 编程
- 4.4.2 在 C++中调用 GPU 核心函数
- 4.4.3 GPU 资源的分配和释放
- 4.4.4 异步 GPU 编程

##### 4.5 资源管理的最佳实践

- 4.5.1 资源管理的设计原则
- 4.5.2 资源分配和释放的错误处理
- 4.5.3 C++性能分析和优化常用手段

##### 4.6 作业：4 个 C++小任务，实现四类资源的管理

#### 五. 自动驾驶量产和功能安全

##### 5.1 智能驾驶车规级量产

- 5.1.1 智能驾驶的定义和发展趋势
- 5.1.2 车规级量产的重要性
- 5.1.3 目标：安全性、可靠性和成本控制的平衡

##### 5.2 车规级软件规范：ASPICE 开发流程和 V 模型

- 5.2.1 ASPICE 标准的概述
- 5.2.2 V 模型在汽车软件开发中的应用
- 5.2.3 软件开发的阶段和活动

##### 5.3 风格指南、Misra 规范与车规级代码

- 5.3.1 代码风格指南的作用
- 5.3.2 Misra 规范的目的和要求
- 5.3.3 车规级的代码规范要求

##### 5.4 C++性能优化：量产级水准

- 5.4.1 C++代码性能的重要性
- 5.4.2 代码性能分析工具
- 5.4.3 代码性能分析和优化的主要方式
- 5.4.4 编译器优化和手动优化

##### 5.5 软件功能安全

- 5.5.1 ISO 26262 标准概述
- 5.5.2 安全需求的定义和分析
- 5.5.3 软件安全概念和措施
- 5.5.4 失效处理和恢复机制

##### 5.6 设计模式

- 5.6.1 创建型模式：工厂模式、单例模式
- 5.6.2 结构性：适配器、组合模式
- 5.6.3 行为型：观察者模式

##### 5.7 作业：修改代码中不符合规范的实现，尝试做性能分析和优化

#### 六. 自动驾驶常用算法和工具库

##### 6.1 Eigen 库

- 6.1.1 Eigen 库的概述和应用领域
- 6.1.2 Eigen 的使用与函数介绍
- 6.1.3 Eigen 内部实现简析

##### 6.2 Apollo 数学工具库

- 6.2.1 感知、预测、规划、控制与工具库
- 6.2.2 工具库在智能驾驶中的角色和功能
- 6.2.3 实战系列：源码分析

##### 6.3 用 C++从零开始动手实现自己的 CNN 神经网络

- 6.3.1 神经网络层和模型
- 6.3.2 手动实现一个神经网络推理库
- 6.3.3 神经网络库的基本使用和模型训练

##### 6.4 其它知名库介绍

- 6.4.1 OpenCV
- 6.4.2 PCL 点云库
- 6.4.3 Boost

##### 6.5 作业：扩展所实现的神经网络库，实现 Transformer



## 4. 讲解思路与课程大纲

### 4.2 课程大纲

#### 七. 工作实践指南

##### 7.1 单元测试

- 7.1.1 单元测试的基本原理
- 7.1.2 测试驱动开发简介
- 7.1.3 使用测试框架 GTEST 进行单元测试

##### 7.2 代码管理

- 7.2.1 版本控制的作用
- 7.2.2 Git 的基本使用
- 7.2.3 团队协作与分支管理

##### 7.3 作业：用 GTEST 写自己的测试用例

#### 八. 智能驾驶 C++项目实战

##### 8.1 Apollo 系统介绍

- 8.1.1 Apollo 系统简介
- 8.1.2 Apollo 主要模块划分
- 8.1.3 Apollo 系统运行流程

##### 8.2 Apollo 环境搭建

- 8.2.1 准备 Ubuntu 操作系统和 GPU 驱动
- 8.2.2 Docker 环境搭建
- 8.2.3 下载 Apollo 代码
- 8.2.4 获取 Docker 镜像
- 8.2.5 编译与运行

##### 8.3 Apollo Cyber RT 简析

- 8.3.1 Cyber RT 框架介绍
- 8.3.2 Cyber RT 通信的原理和实践
- 8.3.3 Cyber RT 调度的原理和实践
- 8.3.4 Cyber RT 开发工具

##### 8.4 算法代码编写和调试

- 8.4.1 实现 Routing 模块全局路径规划
- 8.4.2 实现 Planning 换道超车任务

##### 8.5 代码运行和效果展示

- 8.5.1 用 gdb 进行代码调试
- 8.5.2 用 Apollo 仿真将运行结果可视化

#### 九. 工作心得

##### 9.1 保持文档记录

##### 9.2 优化目标，以终为始

##### 9.3 未来漫漫，保持学习



公众号



自动驾驶与计算机视觉  
日常干货分享

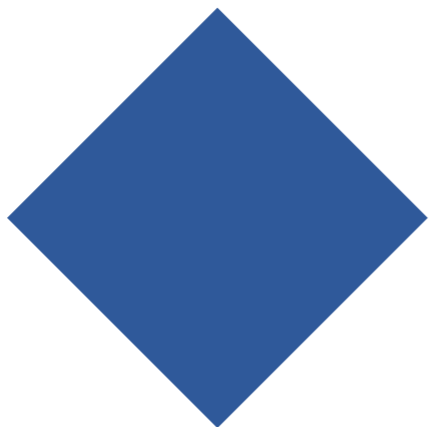
知识星球



加入自动驾驶之心知识星球，  
获取更多硬核干货

自动驾驶与AI全栈技术学习+领域大咖交流+职位内推!

本站资料



# End