面向自动驾驶的C++实战
第7章 智能驾驶C++项目实战

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. BAIDU APOLLO介绍
2. APOLLO CYBER RT简析
3. 环境搭建和工具使用
4. 代码编写和调试
5. 神经网络实现分析
6. 项目实战分析

1 BAIDU APOLLO介绍

1. Baidu Apollo介绍

1.1 基本介绍

介绍

百度公司研发的Apollo（阿波罗）是一个开放的、完整的、安全的自动驾驶平台，可以帮助汽车行业及自动驾驶领域的合作伙伴结合车辆和硬件系统，快速搭建一套属于自己的自动驾驶系统。

Apollo自动驾驶开放平台为开发者提供了丰富的车辆、硬件选择，强大的环境感知、高精定位、路径规划、车辆控制等自动驾驶软件能力，以及高精地图、仿真、数据流水线等自动驾驶云服务，帮助开发者从0到1快速搭建一套自动驾驶系统 [1] 。

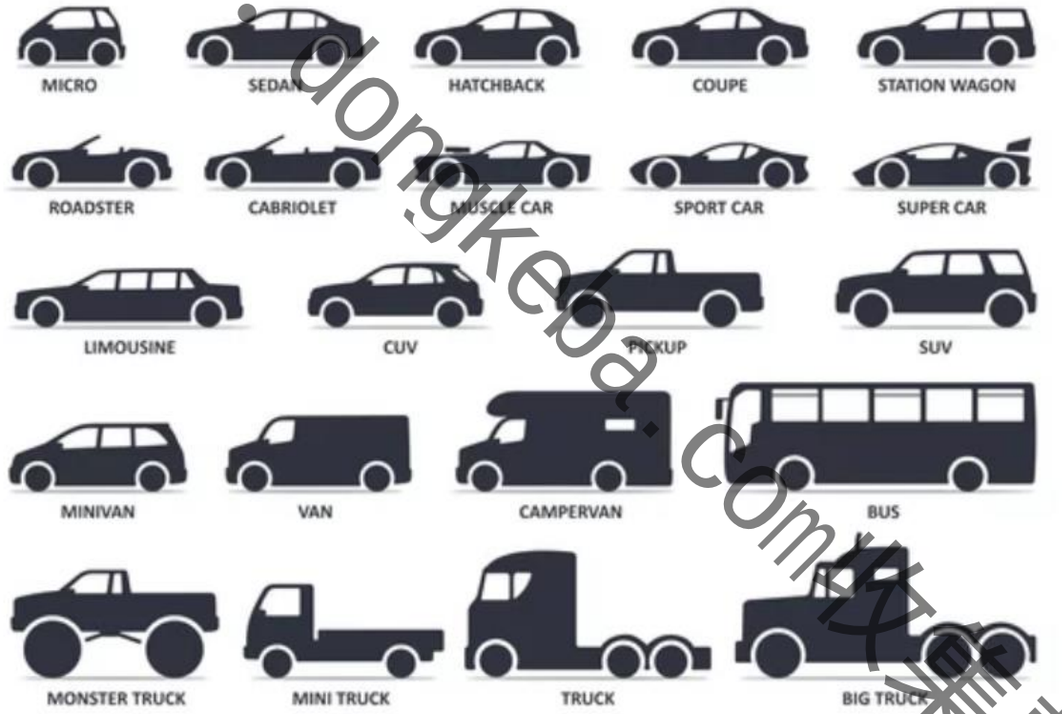
Apollo开放平台9.0全新架构图, link

1. Baidu Apollo介绍

1.1 基本介绍

开放生态

百度Apollo的远景：让Apollo自动驾驶时代汽车行业的Andriod生态系统[2]。



各种各样的自动驾驶车辆，都基于同一套系统

百度的商业设想，带来了开放的Apollo，也带给了我们在自动驾驶领域很好的学习机会。

1. Baidu Apollo介绍

1.2 和竞争对手的比较

对比

- 国外
  - **奔驰**、**宝马**、**丰田**、**本田**、**福特**等国际汽车巨头，智能驾驶的研发能力较差；
  - Tesla FSD致力于将L2、L3智驾搭载到乘用车，有Tesla AI Day分享，但没有开源计划；
  - **Waymo**、**Cruise** 、**Aurora**等公司聚焦于L4场景，没有开源代码。
- 国内
  - 智能驾驶的Tier1或Tier2供应商，如**华为**、**大疆**；
  - 造车新势力，如**理想**、**蔚来**、**小鹏**；
  - 老牌国企或民企OEM主机厂，如**上汽**、**一汽**、**广汽**、**比亚迪**、**吉利**、**长安**、**长城**。
  - 这些知名公司都没有开源智能驾驶代码，往往只有新闻宣传或者技术概要分享。

百度是 信息科技研发实力较强 + 开源代码 + 主动分享推广 的唯一一家头部公司。

1. Baidu Apollo介绍

1.2 和竞争对手的比较

和Autoware的对比

项目	Baidu Apollo	Autoware
Github Page	Apollo Auto	The Autoware Foundation
投入资金	数十亿元至百亿级别	估计为数千万至亿美元级别
投入时间	自2013年至今	自2015年至今
背靠组织	百度	Autoware基金会 背后是Tier IV、AutoCore.ai等公司
主要所在国家	中国	日本
系统复杂度	高，包含感知、决策、规划、控制等全栈自动驾驶技术	高，主要专注于自动驾驶系统的中间件和算法
系统架构	基于自研Cyber RT	基于ROS
社区规模	大	中
商业应用案例	多	中
文档完整性	高	中
总结	• 全栈解决方案、强大的中文支持、大公司资金人力支持； • 系统复杂度高，对学习者的技术要求更高。	• 国际化程度高、文档相对较少； • 对于熟悉ROS生态的学习者更友好。

1. Baidu Apollo介绍

1.2 和竞争对手的比较

Autoware

	Autoware	Waymo	Aurora	Cruise
Description	Open-source software for autonomous driving technology	Developer of self-driving solutions for autonomous vehicles	Provider of AI and SaaS-based self-driving solutions for autonomous vehicles	SaaS-enabled solutions for autonomous car
Founded Year	2017	2009	2017	2013
Location	Bunkyo (Japan)	Mountain View (United States)	Mountain View (United States)	San Francisco (United States)
Company Stage	Unfunded	Series E	Acquired	Acquired
Unicorn Rating	-	Decacorn	-	-
Total Funding	-	$5.5B	$693M	$7.72B
Funding Rounds	-	3	6	10
Latest Round	-	Series E, $2.5B, Jun 16, 2021	Post IPO, $600M, Jul 26, 2023	Series E, $1.35B, Feb 01, 2022
Investor Count	-	12	17	39
Top Investors		CPP Investments, Silver Lake & 10 others	CPP Investments, T. Rowe Price & 15 others	T. Rowe Price, Qualcomm Ventures & 37 others
Tracxn Score What is this?	12/100	84/100	81/100	79/100
Overall Rank	91st	1st	2nd	3rd

1. Baidu Apollo介绍

1.2 和竞争对手的比较

相关新闻

- 据百度财报，自2015年正式成立L4事业部以来，百度每年的研发投入都不低于100亿。2021年更是跃升到200亿元以上，2022年为223亿元左右。近十年，累计研发投入接近1500亿元。

- 2021年11月01日，AutoCore.ai宣布完成由富士康、高瓴创投继续领投，IDG资本联合领投，松禾资本，民银国际，广汽资本和赛博朋克奇点科技公司跟投的 数亿元A轮融资。AutoCore作为Autoware基金会的创始顶级会员，是基金会多个项目的主要贡献者之一[link]。

- 2024年6月17日，自动驾驶系统开源软件供应商提雅智行（TIER IV）宣布其B轮融资再获5400万美元（85亿日元），这使得其B轮融资总额达到1.32亿美元（207亿日元），公司总融资额达到2.43亿美元（381亿日元）。在A轮融资期间，提雅智行的业务范围从开发开源软件Autoware扩展到商业化综合软件平台，这些平台涵盖自动驾驶系统的通信、保险和风险管理功能[link]。

可以基本得出结论：Autoware的资金投入远不及Apollo，所以大概率无法达到Apollo的水准。

另外，源码中还有很多其他信息：ApolloAuto/apollo-Contributors

1. Baidu Apollo介绍

1.2 和竞争对手的比较

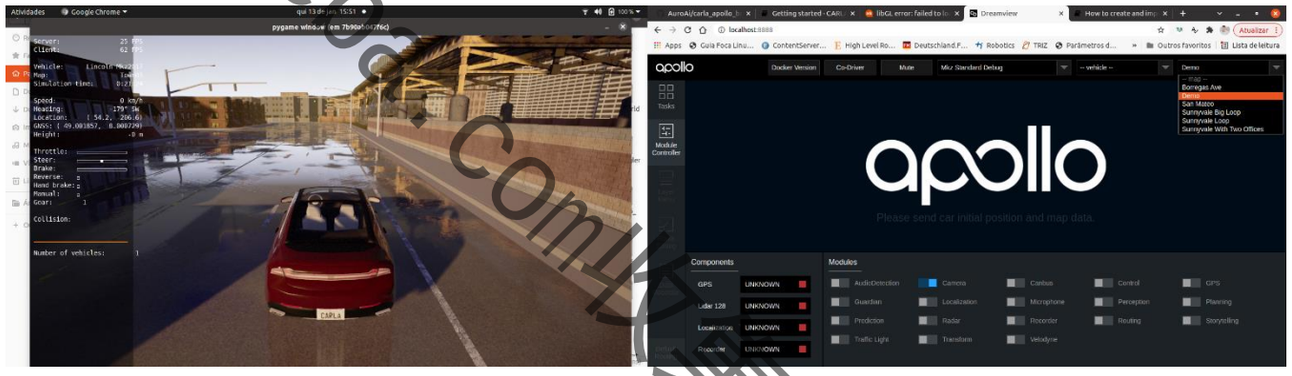
其他选择

CARLA

- **优点**：CARLA基于虚幻引擎构建，可以生成高保真的3D环境，提供视觉上的真实性，这对于测试车辆感知系统尤其重要；CARLA能够实现实时的仿真速度，这对于快速迭代和验证算法至关重要。
- **缺点**：CARLA主要是一个模拟器、仿真器，用于算法研究、场景模拟、教育培训，不关注实际运行于自动驾驶汽车上的模块管理、消息通信、运行效率和调度，不是真实可部署的自动驾驶平台。

所以，CARLA主要应用在自动驾驶的模拟仿真，如ROS+CARLA联合仿真、Apollo+CARLA联合仿真。

Virtual Traffic Flow in CARLA, link



Apollo和CARLA联合仿真，link

所以，本章选择基于Apollo开展。

1. Baidu Apollo介绍

1.3 学习资料

官方文档

- Apollo自动驾驶开放平台文档：https://apollo.baidu.com/community/Apollo-Homepage-Document
- Cyber RT Documents：https://cyber-rt.readthedocs.io/en/latest/CyberRT_Quick_Start.html

官方视频

- Apollo精品课程：https://apollo.baidu.com/community/online-course
- Apollo公开课：https://apollo.baidu.com/community/public-course

代码仓库

- Apollo8.0 Github Repo: https://github.com/ApolloAuto/apollo/tree/r8.0.0

社区文章

- Cyber入门与探索：https://apollo.baidu.com/community/column/3
- Apollo入门与探索：https://apollo.baidu.com/community/column/2

资料整理

- Apollo开发者社区课程整理：https://mp.weixin.qq.com/s/alhlb2KXOYbgoNm87rfffg

站外视频

- https://www.bilibili.com/video/BV1bW4y187MQ/?vd_source=d460cd92309c1e8308165e58f41393d8
- https://www.bilibili.com/video/BV1gi4y1C7XC/?vd_source=d460cd92309c1e8308165e58f41393d8

2 APOLLO CYBER RT简析

2. Apollo Cyber RT简析

2.1 Cyber RT框架介绍

概述

Cyber RT (Cyber Run Time) 是专为自动驾驶场景设计的**高性能运行时**框架。它提供了一个统一的基础设施，用于管理和协调Apollo系统中的各个模块。

- **模块化设计**：Cyber RT采用模块化设计，将自动驾驶系统的各个功能模块（如感知、定位、规划、控制等）解耦，使得每个模块可以独立开发、测试和部署。

- **消息传递机制**：利用高效的消息传递机制，各个模块之间可以高效地交换数据，实现低延迟和高吞吐量。
- **文件配置**：Cyber RT 使用多种格式的配置文件管理系统参数、模块设置和消息通道等。通过这些配置文件，开发者可以灵活地调整系统行为，适应不同的应用需求。
- **开发者工具**：提供一系列开发工具，包括代码生成器、调试工具、数据记录与回放、性能分析工具等，帮助开发者高效地进行模块开发和系统集成。并且支持在仿真环境中进行开发和测试，降低实际路测的风险和成本。

2. Apollo Cyber RT简析

2.1 Cyber RT框架介绍

编程模型

Cyber RT的编程模型里包含了多类基本单元，包括：

- Component：基本的功能单元，如感知、规划模块等。
- Node：通信的基本单位。
- Channel：用于数据交换的通道。
- Reader/Writer：用于读写Channel中的数据。
- Timer：用于周期性任务的执行。

优点

- **解耦**：发布-订阅模型使得各个模块之间相互独立，可以独立开发和测试。
- **高效**：通过共享内存和高效的消息队列，实现大数据量的快速传输。
- **实时性**：设计上的低延迟和高吞吐量，满足自动驾驶系统对实时性的要求。
- **灵活性**：支持动态添加和移除发布者和订阅者，提高系统的灵活性。

2. Apollo Cyber RT简析

2.1 Cyber RT框架介绍

目录结构

参考自：https://apollo.baidu.com/community/article/29

```
cyber           消息中间件，替换ros作为消息层
docker          容器相关
docs            文档相关
modules         自动驾驶模块，主要的定位，预测，感知，规划都在这里
    calibration     校准、标定，主要用于传感器坐标的校准，用于感知模块做传感器融合
    canbus          通讯总线，工业领域的标准总线，鉴于工业界的保守，我估计后面会有新的总线来取代
    common          一些公用的代码，如时间、日志等
    contrib
    control         控制模块，根据planning生成的路径对车辆轨迹进行控制
    data            地图等生成好的数据存放在这里
    dreamview       仿真，能够对自动驾驶过程中的数据进行回放、模拟运行，以对算法做测试
    drivers         雷达，lidar，GPS，canbus，camera等驱动
    guardian        守护程序，作为control到底盘之间的守护者，防止异常
    localization    定位，获取汽车的当前位姿
    map             地图模块
    monitor         监控模块，主要是监控汽车状态，并且记录，用于故障定位，健康检查等
    perception      感知，获取汽车当前的环境，行人，车辆，红绿灯等，给planning模块规划线路
    planning        规划，针对感知到的情况，对路径做规划，短期规划，生成好的路径给control模块
    prediction      预测，属于感知模块，对运动物体的轨迹做预测
    routing         导航，就是地图上查询2点之间的全局线路
    third_party_perception 第三方感知模块
    tools           工具，这里面的工具倒是很多，后面再详细介绍下
    transform       坐标、位姿变换等操作的工具
    v2x             vehicle-to-everything，实现车辆与一切可能影响车辆的实体实现信息交互的服务
scripts         脚本
third_party     第三方库
tools           额外的工具目录
```

2. Apollo Cyber RT简析

2.2 Component组件机制

本小节内容参考自：**Apollo官方文档-框架设计-软件核心-CyberRT-CyberRT组件机制[link]**。

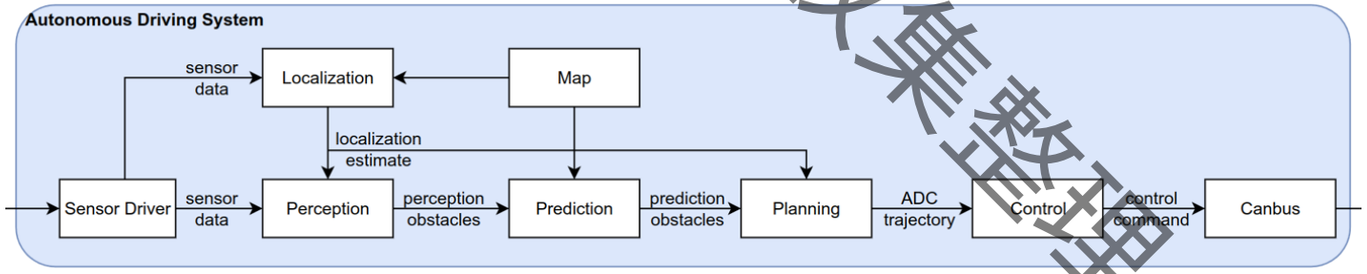
主要问题

- 我要开发的组件，主要实现什么功能？
- 组件的上游输入是什么？输出是什么？分别多久一次（周期/频率）？
- 组件的主要计算处理，多久执行一次？是按照一定是周期定时执行，还是由某个事件触发执行？
- 添加一个新组件，能不能尽量简单，让我不需要关注无关的组件，更多关注于算法本身？
- ……

概念

组件，即**Component**，是Cyber RT提供的用来构建功能模块的基础类，可以理解为Cyber RT对算法功能模块的封装，配合Component对应的DAG文件，Cyber RT可实现对该功能模块的动态加载。

被Cyber RT加载的 Component 构成了一张去中心化的网络。每一个Component是一个算法功能模块，其依赖关系由对应的配置文件定义，而一个系统是由多个component组成，各个component由其相互间的依赖关系连接在一起，构成一张计算图。



2. Apollo Cyber RT简析

2.2 Component组件机制

类型

Component有两种类型，分别为 apollo::cyber::Component 和 apollo::cyber::TimerComponent。

消息触发式

- **Component**：提供消息融合机制，最多可以支持 4 路消息融合，当 从多个 Channel 读取数据的时候，以第一个 Channel 为主 Channel。当主 Channel 有消息到达，Cyber RT会调用Component::Proc函数进行一次数据处理。
- **TimerComponent**：不提供消息融合，与 Component 不同的是，TimerComponent::Proc函数不是基于主 channel 触发执行，而是由系统定时调用，开发者可以在配置文件中确定调用的时间间隔。

定时式

创建与工作

1. 包含相关的头文件；
2. 定义一个类，并继承Component或者TimerComponent。
3. 重写Init()和Proc()函数；Init()函数在 Component 被加载的时候执行，用来对Component进行初始化，如Node创建，Node Reader创建，Node Writer创建等等；Proc()函数是实现该Component功能的核心函数，其中实现了该Component的核心逻辑功能。
4. 在Cyber RT中注册该Component，只有在Cyber RT中注册了该Component，Cyber RT才能对其进行动态的加载，否则，cyber RT动态加载时报错。

Code sample: link1, link2, planning_component.h

最终，所有的Comopnent都会被编译成独立的.so动态库文件，Cyber RT会根据开发者提供的配置文件，按需加载对应的Component。

2. Apollo Cyber RT简析

2.3 Message消息传递机制

本小节内容参考自：**Cyber RT 通信机制简介**[link]和**Cyber RT API Tutorial**[link]。

概述

Cyber RT 的消息传递机制主要采用发布-订阅（Publish-Subscribe）模型，这种模型使得各个模块可以解耦，模块之间通过消息队列进行通信，而不是直接调用彼此的函数。

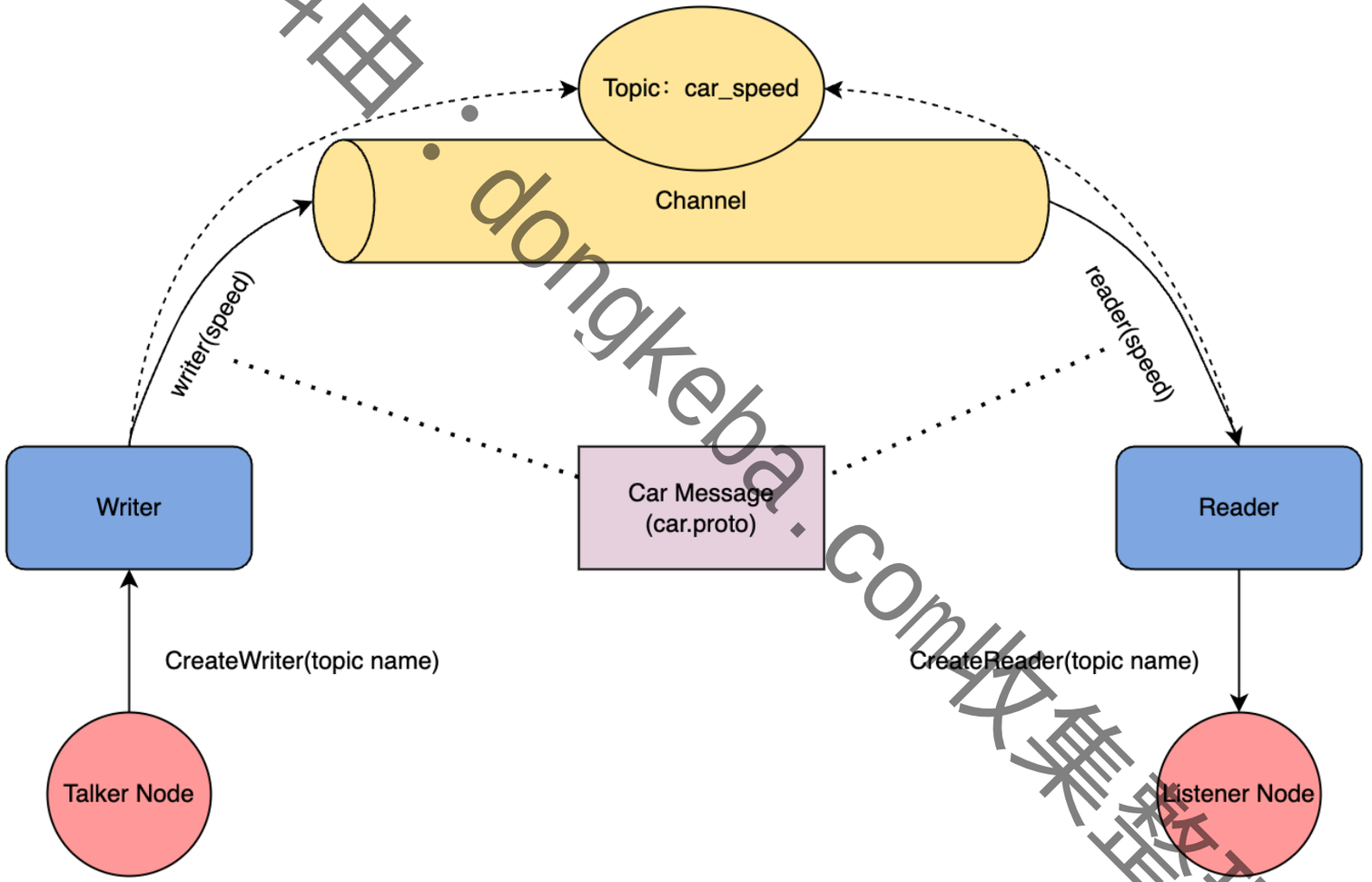
核心概念

- **消息（Message）**：消息是发布者通过通道传递的数据单元，消息包含具体的信息，如传感器数据、控制指令等。
- **消息类型（Message Type）**：消息类型是对消息数据结构的定义，描述了消息中包含的数据字段和数据类型。消息类型确保发布者和订阅者之间的数据格式一致，避免数据解析错误。
- **主题（Topic）**：标识消息的种类和目的，发布者发布消息和订阅者订阅消息时使用。
- **Publisher（发布者）**：负责发布消息的参与者。发布者将消息发送到一个特定的话题（Topic）。
- **Subscriber（订阅者）**：负责接收消息的参与者。订阅者从特定的话题接收消息。
- **Channel（通道）**：消息传递的媒介，一个通道对应一个话题。通道用于连接发布者和订阅者，发布者将消息**发送**到通道，订阅者从通道**接收**消息。通道是个抽象出来的概念，具体的传输过程在Cyber RT内部实现。

在Apollo中，使用通常使用**Protocol Buffers（Protobuf）**定义消息类型，并传输消息。

2. Apollo Cyber RT简析

2.3 Message消息传递机制



reader-writer通信, link

2. Apollo Cyber RT简析

2.3 Message消息传递机制

实际示例

以自动驾驶系统中的激光雷达感知模块为例：

1.定义消息类型

```
syntax = "proto3";
message LidarData {
  uint32 timestamp = 1;
  repeated float points = 2;
}
```

```
auto lidar_data = std::make_shared<LidarData>();
lidar_data->set_timestamp(current_time);
// Do something to fill point cloud
auto writer = node->CreateWriter<LidarData>("/sensor/lidar");
writer->Write(lidar_data);
```

2.发布者发布消息。主题为/sensor/lidar，**通道**为/sensor/lidar通道，发送**消息**为一个包含具体时间戳和点云数据的 LidarData 消息。

3.订阅者接收消息。主题为/sensor/lidar，**通道**为/sensor/lidar通道，订阅者从这个通道接收 LidarData类型的消息，并立即触发回调函数开始处理。

```
auto reader = node->CreateReader<LidarData>("/sensor/lidar",
    [&](const std::shared_ptr<LidarData>& msg) {
        // 处理接收到的激光雷达数据
    }
);
```

2. Apollo Cyber RT简析

2.3 Message消息传递机制

服务通信机制

Cyber RT还提供了另一种通信机制：服务（Service）通信。参考link。

特性	消息机制（Message）	服务机制（Service）
模型	发布-订阅（Publish-Subscribe）	请求-响应（Request-Response）
通信方式	异步：发布者发送消息后立即往下运行，不需要等待响应结果。	同步：请求者发送请求并等待响应，服务提供者处理请求并返回结果。
主要组件	Publisher、Subscriber、Channel等	Client、Service、Service Name
适用场景	持续数据流、广播	断断续续的命令、点对点通信
优点	松耦合、高扩展性	紧耦合、确定性、适应性命令控制
类比	UDP	TCP 互联网C/S架构
典型应用	传感器数据发布和订阅	偶尔查询某个状态数据
代码示例	talker, listener	server and client
Apollo实例	planning_component.h	

2. Apollo Cyber RT简析

2.3 Message消息传递机制

背后的通信机制

通信需求的复杂性包括：

- **通信的节点和节点之间关系不确定**。可以在
  a) 同一个进程内
  b) 同一个计算机内不同进程间 ← 我们需要的
  c) 不同计算机之间进行
- **传输消息的数据量和频率各不相同**。可能数据量很大、频率也很高；可能数据量小但频率高；可能数据量大但偶尔才有；
- **传输消息的优先级和重要性不同**。有的消息至关重要，必须尽快送达且不能丢失；有的消息不重要，偶尔丢失几次也没关系。
- ……

操作系统基本的通信机制有：**管道（pipe）、命名管道（named pipe）、信号（signal）、消息队列（message queue）、共享内存（shared memory）、信号量（semaphore）、网络套接字（socket）**（详细了解请看《操作系统》）。 ← 我们拥有的

不论是Apollo还是ROS，都是基于上述最基本的、操作系统提供的通信机制，进行组合、优化、简化得到一套通信框架/通信中间件方案。

把供给和需求进行匹配

2. Apollo Cyber RT简析

2.3 Message消息传递机制

背后的通信机制

Apollo中主要使用的是：

- **指针传递**：对于**同一个进程内**的通信，Cyber RT主要使用传递指针的方式，也是最高效的通信方式；
- **共享内存**：对于**大数据量**的消息（如图像、点云数据），Cyber RT主要通过共享内存实现高效的数据传输，减少内存拷贝，提高传输速度。
- **网络套接字**：不管是进程间还是多机之间，都可以敏捷灵活地支持，但是通信效率较低。

和ROS类似，Apollo底层使用**DDS（Data Distribution Service）**进行真正的数据传输，可以通过DDS的QoS（Quality of Service）策略服务更精准高效地控制、管理消息的传输。

2. Apollo Cyber RT简析

2.4 文件配置和代码浅析

component内目录结构

```
common/
    xxxx_gflags.h              # Google Flag配置
    xxxx_gflags.cc
    xxxx_algorithm.h          # component内通用的算法或函数
    xxxx_algorithm.cc
conf/                          # 运行中需要读取的配置文件和参数
    highway_config.pb.txt
    city_config.pb.txt
dag/                          # component的dag配置，说明组件运行的依赖关系
    xxxx.dag
impl/                          # 内部实现
launch/                        # 单独运行本component的配置文件
    xxxx.launch
proto/                        # 消息通信用到的Protobuf配置文件
    message1.proto
    message2.proto
BUILD                          # 本component的构建配置文件
README.md                      # 关于本component的使用描述
xxxx_component.cc              # component代码入口源文件
xxxx_component.h              # component代码入口头文件
```

2. Apollo Cyber RT简析

2.4 文件配置和代码浅析

简单分析

dag文件

- 用于配置一个component的启动依赖；
- 实际上是以 cyber\proto\dag_conf.proto 中的DagConfig 定义得到的*.pb.txt配置文件。

launch文件

- 用于脚本单独启动一个component；
- 是 scripts\cyber_launch.sh 的参数文件。

mainboard浅析

- cyber\mainboard\mainboard.cc

component浅析

- cyber\component\component.h
- cyber\component\timer_component.h
- cyber\component\component_base.h

Writer/Reader浅析

- cyber\node\node.h
- cyber\node\reader.h
- cyber\node\writer.h

2. Apollo Cyber RT简析

2.5 开发者工具

Cyber RT还提供了一系列开发者工具，帮助开发者更高效地进行模块开发、调试和性能优化。

参考文档： https://cyber-rt.readthedocs.io/en/latest/CyberRT_Developer_Tools.html

Cyber Recorder

一个用于**录制和回放**Cyber RT消息的数据记录工具。这对于自动驾驶系统的开发和调试非常有用，因为可以捕捉实际驾驶过程中产生的数据，然后在开发环境中回放这些数据进行分析和改进。

- **录制数据：**捕捉和存储从各种传感器和模块生成的实时数据。
- **回放数据：**重现录制的场景，以便在开发和测试时使用相同的数据集。
- **支持多种数据格式：**可以处理各种传感器数据和系统消息。

```
apollo@practical-instance-12316:/apollo_workspace$ cyber_recorder play -f demo_3.5.record -l
file: demo_3.5.record, chunk_number: 3, begin_time: 1546888377338834894 (2019-01-07-19:12:57), end_time: 1546888422886740928 (20
19-01-07-19:13:42), message_number: 61615
earliest_begin_time: 1546888377338834894, latest_end_time: 1546888422886740928, total_msg_num: 61615

Please wait 3 second(s) for loading...
Hit Ctrl+C to stop, Space to pause, or 's' to step.

[RUNNING] Record Time: 1546888382.442    Progress: 5.103 / 45.548
```

2. Apollo Cyber RT简析

2.5 开发者工具

Cyber Monitor

一个**实时监控**工具，用于监视Apollo系统中各个模块的运行状态和消息传递情况。它帮助开发者及时了解系统的运行状况，诊断问题并进行调试。

- **实时监控：**显示系统中各模块的状态和消息流动情况。
- **模块健康检查：**查看每个模块的健康状态，包括CPU、内存使用情况等。
- **日志和警告：**提供实时日志记录和警告信息，帮助快速定位和解决问题。

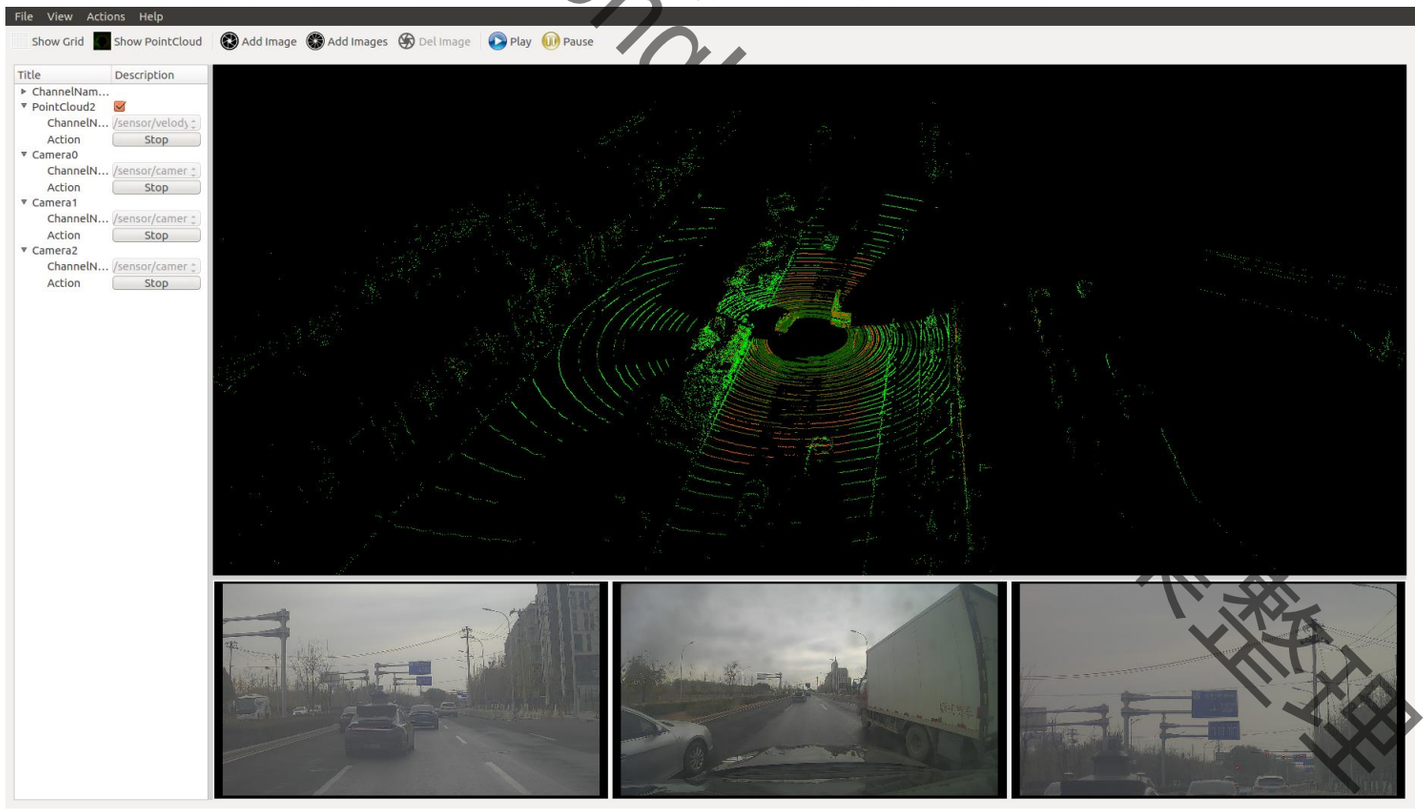
2. Apollo Cyber RT简析

2.5 开发者工具

Cyber Visualizer

一个数据可视化工具，用于展示和分析来自Apollo系统的各种传感器数据和消息。它帮助开发者更直观地理解数据，并进行详细分析。

- **3D数据可视化：**可以在3D空间中展示传感器数据，如激光雷达点云、相机图像等。
- **数据分析：**提供工具用于详细分析传感器数据和系统消息。
- **交互式界面：**用户可以通过交互界面选择和过滤数据，进行更深入的分析。

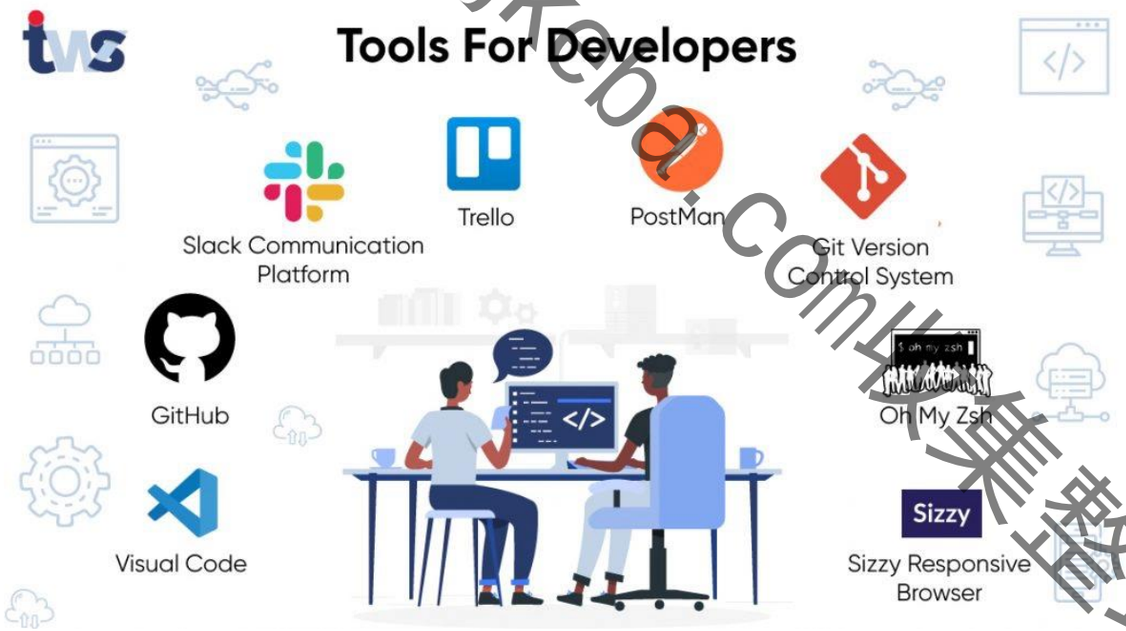


2. Apollo Cyber RT简析

2.5 开发者工具

心得分享

- 开发者工具的重要性，并不比最终产品低；
- 这些工具都不是凭空诞生、一开始就有的，而是根据研发者的需求而陆续增加、完善的；
- 开发者在开发中感到低效、重复、混乱，这些实实在在的痛点，才迫使开发者先开发出开发者工具，再继续开发产品；
- 我们也可以开发自己需要的开发者工具，不断提高开发效率；
- 开发者工具的开发，也占据着程序员一定的工作时间比例。



3 环境搭建和工具使用

3. 环境搭建与工具使用

3.1 Docker简介

Docker出现以前的问题

- **依赖管理问题**：目标项目依赖大量的库、包、软件，而且依赖库本身也有很多依赖，这些都要逐个手动下载、安装；
- **版本冲突问题**：库A依赖库C 1.5版本，库B依赖库C2.6版本，版本冲突、版本兼容也需要花费大量精力处理。
- **环境一致性问题**：在这台机器可以好好地运行，怎么到另一台机器就编译失败或者运行失败了？
- **快速部署问题**：在本地开发完成，想要在目标环境上运行起来，又要花大量时间重新安装一遍各种依赖。
- **资源利用问题**：如果使用虚拟机，相当于在OS上再启动一个OS，资源占用多，启动慢。
- **应用隔离问题**：想在主机上运行两个环境，一个随便修改做测试，另一个提交正式代码运行看结果。
- Docker的出现，解决或者大大优化了上述问题。

Docker的概念

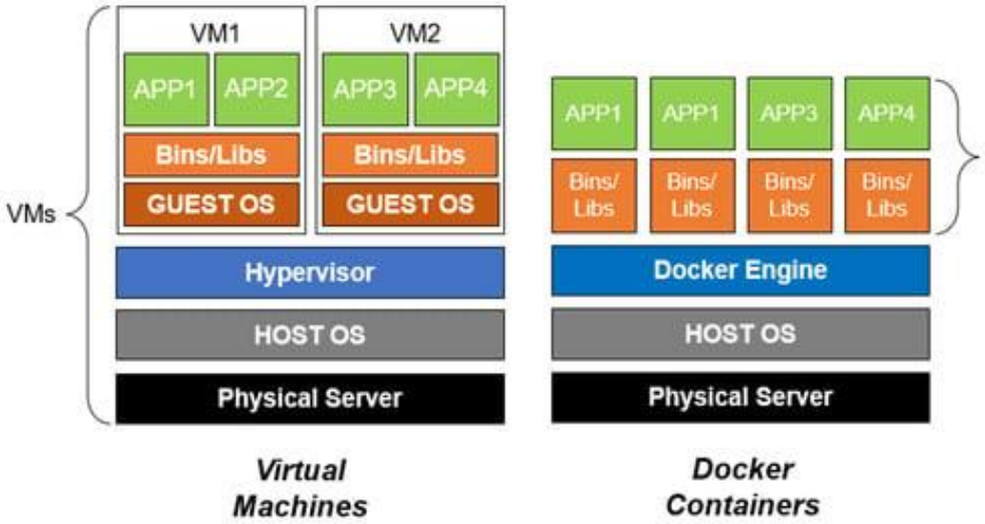
Docker是一个开源的容器化平台，用于开发、运输和运行应用程序，它使用容器来封装应用及其依赖，确保应用可以在任何环境中一致运行。

Docker基于Linux内核的cgroup、namespace以及UnionFS等技术，对进程进行封装隔离，属于操作系统层面的**虚拟化技术**。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。

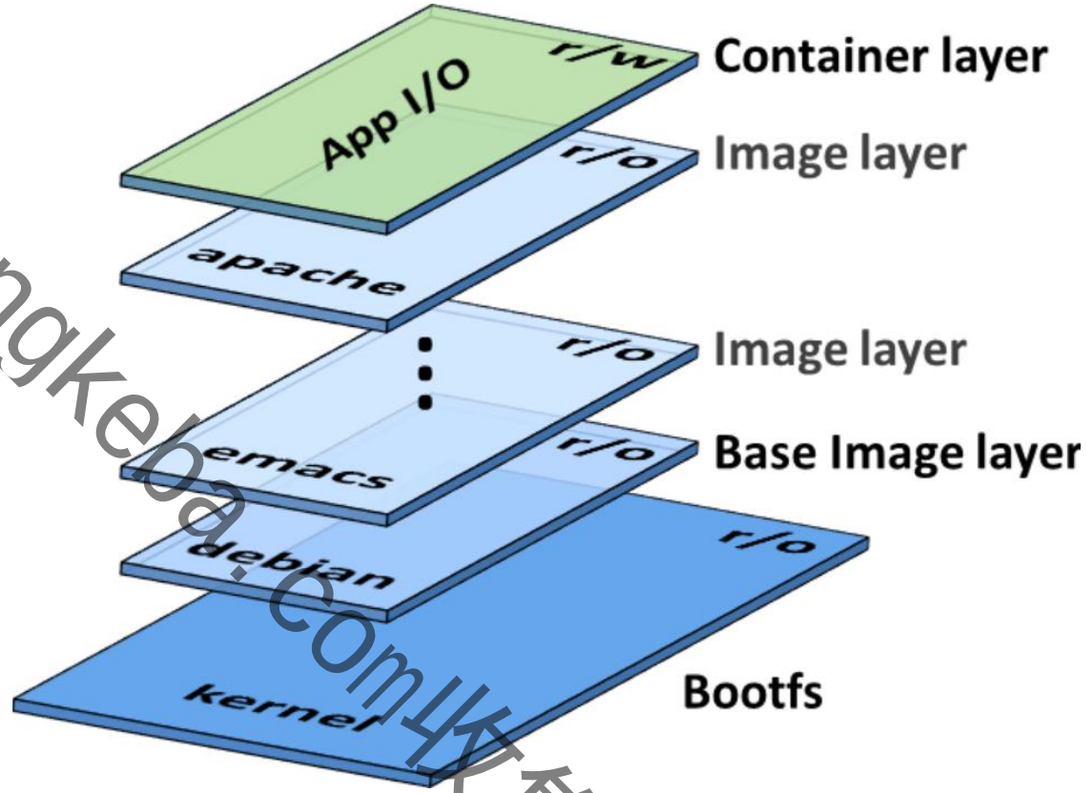
Docker在容器的基础上，进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极大的简化了容器的创建和维护；Docker通过简化应用的打包、分发和运行过程，极大地改变了软件开发和部署方式。

3. 环境搭建与工具使用

3.1 Docker简介



Virtual Machines VS. Docker Containers, link



Docker Image Layers, link

3. 环境搭建与工具使用

3.1 Docker简介

概念	说明
镜像(Images)	Docker镜像是用于创建Docker容器的模板，是Docker生命周期中的构建或者打包阶段。 镜像是一种基于联合文件系统的层式结构，由一系列指令一步一步构建出来，一个镜像可以分为多个镜像层。
容器(Container)	容器是独立运行的一个或一组应用，是镜像运行时的实体，是启动或者执行阶段。 容器基于镜像启动，一旦容器启动完成后，我们就可以登录到容器中安装自己需要的软件或者服务。
仓库(Registry)	Docker仓库用来保存镜像，可以理解为代码控制中的代码仓库。 Docker公司运营的公共的Registry-DockerHub提供了庞大的镜像集合供使用；企业也可以自己构建私有的Registry。 一个Docker Registry中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本。
客户端(Client)	Docker客户端通过命令行或者其他工具使用Docker SDK与Docker的守护进程通信。
主机(Host)	一个物理或者虚拟的机器，用于执行 Docker 守护进程和容器。
Dockerfile	Dockerfile是一个文本文件，包含了一系列命令来组建一个Docker镜像。 开发者可以使用Dockerfile来定义镜像的构建过程。

可以参考Git的设计概念。

3. 环境搭建与工具使用

3.1 Docker简介

举例

假设我们要构建和运行Apollo 9.0（只是举例说明，不代表实际的依赖），需要正确配置好以下环境（即各种库、软件、配置文件且为特定版本）：

1. **操作系统**：Ubuntu 18.04 LTS
2. **编程语言**：C++17、Python 3.6.9
3. **常见工具**：curl、make、g++、unzip、vim、git
4. **依赖库**：OpenCV 3.4.1、PCL (Point Cloud Library) 1.8.1、Protobuf 3.3.0
5. **AI中间件和工具**：CUDA 11.2、cuDNN 7.5.1、TensorFlow 1.15.0
6. **环境变量**：如PYTHONPATH、LD_LIBRARY_PATH等
7. ……

对应Dockerfile：

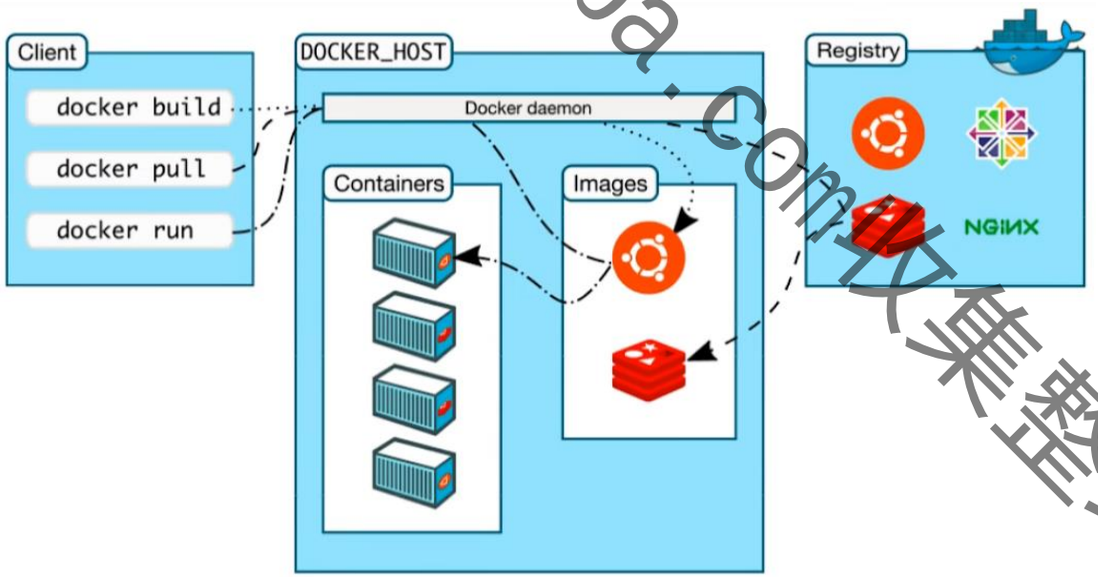
```
FROM ubuntu:18.04 # 基础镜像

# 环境变量设置
ENV DEBIAN_FRONTEND=noninteractive
ENV LANG=C.UTF-8

# 更新APT包索引并安装常用工具
RUN apt-get update && apt-get install -y sudo lsb-release gnupg2 wget curl vim git build-essential cmake &&
apt-get clean

# 安装Python 3.6.9
RUN apt-get update && apt-get install -y python3.6 python3-pip python3.6-dev && apt-get clean

# 安装C++编译器
RUN apt-get update && apt-get install -y g++ && apt-get clean

# Much More to RUN ......
```

3. 环境搭建与工具使用

3.1 Docker简介

使用过程

1. **研发者制作镜像：**Apollo的研发工程师，根据Apollo运行所需要的所有依赖，编写了Dockerfile，并得到最终的Docker镜像，里面包含了多个镜像层，每个镜像层对应一个“RUN”操作包含的修改；
2. **发布者上传镜像：**Apollo的研发工程师把这份（或多份）标准镜像上传到目标Registry（在百度智能云BSC下）；
3. **使用者拉取镜像：**Apollo的使用者，不需要再一个个下载，只需要从Registry拉取整个镜像，就把所有的环境都获取到了。
4. **用镜像创建容器：**使用命令，通过image创建本地的container，进入container，就得到一个万事俱备的Apollo环境。



3. 环境搭建与工具使用

3.1 Docker简介

- 围绕Docker的镜像、容器、仓库等，又有很多知识、概念和操作命令，感兴趣的同学可以再拓展学习；
- 本门课程只要知道基本概念、按照操作步骤执行即可。

其他参考资料

- Docker for beginners[link]
- docker.docs[link]
- 入门Docker教程[link]

3. 环境搭建与工具使用

3.2 搭建Apollo环境

本节可以参考官方文档：Apollo-安装指南-源码安装方式[link]

1. 准备操作系统

在一台新的主机，或者虚拟机内，安装**Ubuntu18.04**或者**Ubuntu20.04**（实测两个版本都可以）。
参见官方安装指南；或者参考其他安装教程进行安装。

2.下载Apollo项目代码

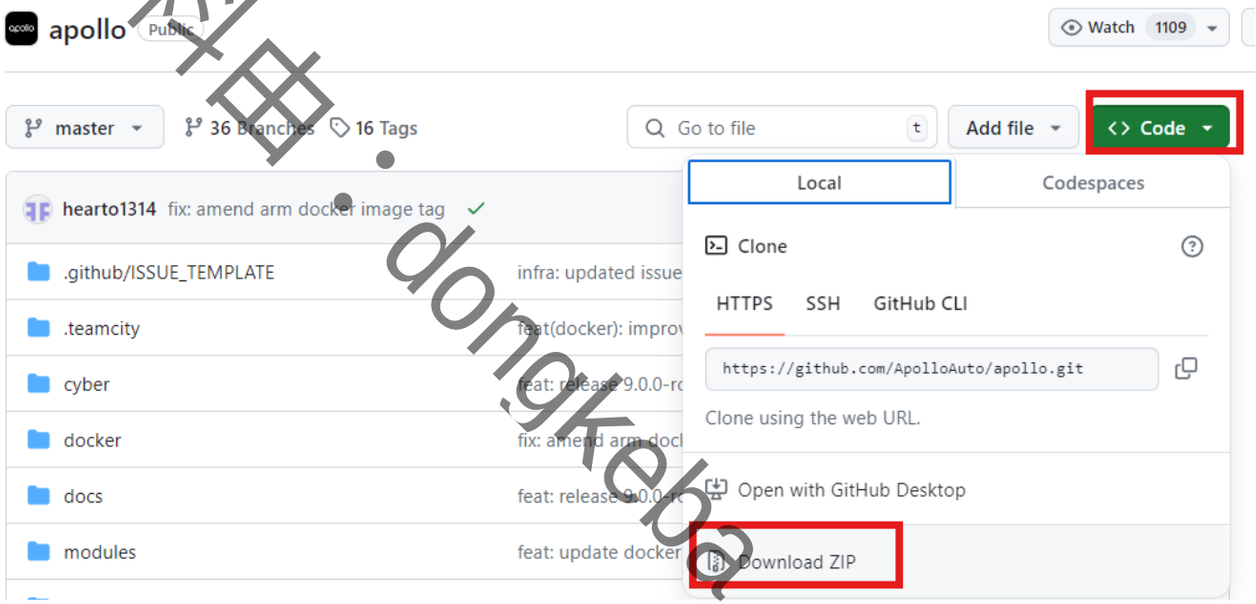
官方Github Repo: https://github.com/ApolloAuto/apollo
Apollo 8.0或者Apollo9.0均可，我们不会用到8.0->9.0的新特性。
可以通过git命令下载：

```
# 以下两种方式，二选一即可
git clone git@github.com:ApolloAuto/apollo.git    # SSH 方式
git clone https://github.com/ApolloAuto/apollo.git # HTTPS 方式
```

【注】 Apollo项目的体积比较大，接近1G，下载花费时间可能超过1小时。
【注】如果下载总是失败（github.com到cn的网络连接不稳定），可以通过浏览器直接下载整个项目的ZIP压缩包，下载好之后在本地解压使用。如下页图。

3. 环境搭建与工具使用

3.2 搭建Apollo环境



通过浏览器直接下载整个项目包

3. 下载安装Docker

```
# 下载Docker的安装脚本
wget http://apollo-pkg-beta.bj.bcebos.com/docker_install.sh

# 执行该脚本，下载Docker并安装
bash docker_install.sh
```

3. 环境搭建与工具使用

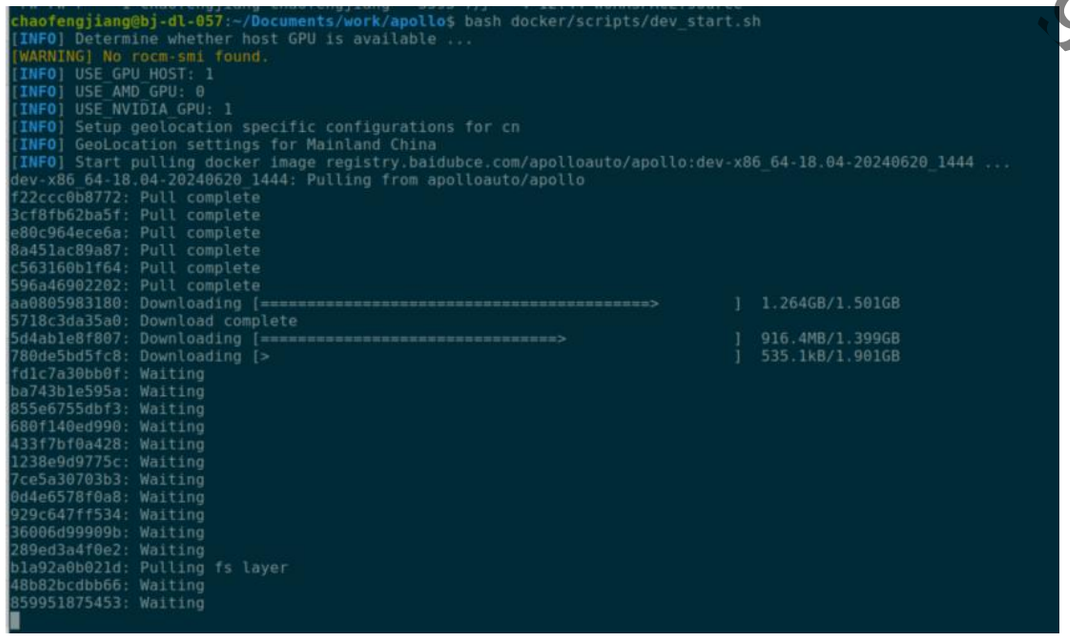
3.2 搭建Apollo环境

4.准备Docker容器

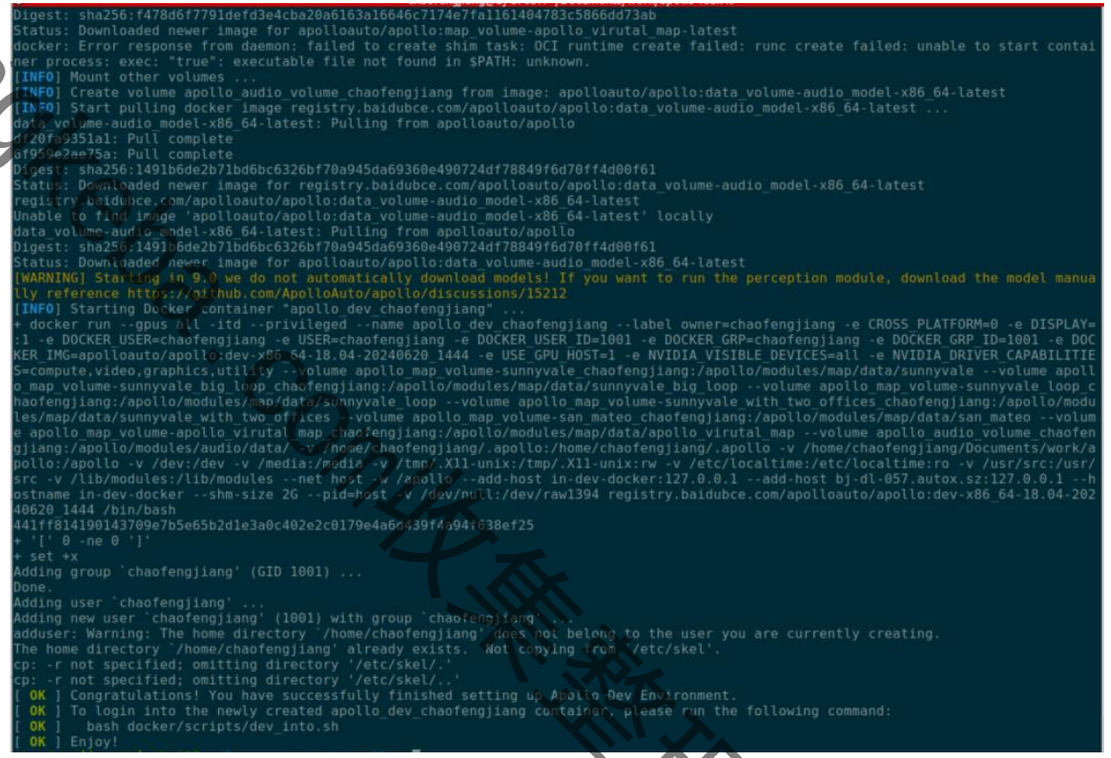
执行Apollo中的脚本，拉取Apollo项目所依赖的环境image：

```
bash /apollo/docker/scripts/dev_start.sh
```

- 如果是第一次执行，这个脚本会自动从远端仓库拉取镜像到本地。所以这个过程包含了大量的下载，所以可能会花费时间3~12h不等；
- 如果镜像已经下载好，会以**镜像**为模版启动**容器**。



执行dev_start.sh



执行dev_start.sh成功

3. 环境搭建与工具使用

3.2 搭建Apollo环境

5.进入Docker容器

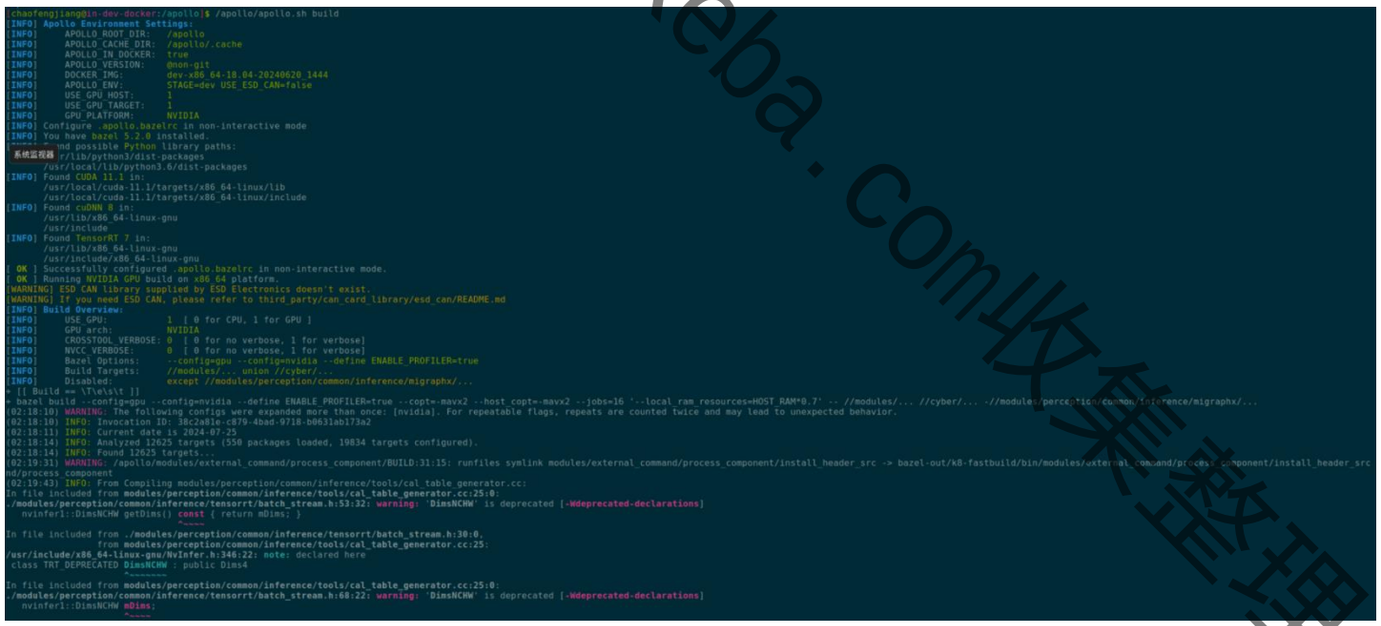
在 apollo 目录下输入以下命令进入容器：

```
bash /apollo/docker/scripts/dev_into.sh
```

6.编译与运行

进入容器后，执行以下命令，编译整个工程的代码：

```
/apollo/apollo.sh build
```



进入容器后，执行编译命令，并开始编译

3. 环境搭建与工具使用

3.2 搭建Apollo环境

打印“Done building apollo”，说明整个Apollo的代码构建完成

整个编译过程大概花费时间0.5-2h，具体耗时取决于计算机的性能。

其他

关于GPU的配置，请参考原始官方文档[link]。

3. 环境搭建与工具使用

3.2 搭建Apollo环境

步骤小结

序号	过程步骤		命令	耗时/h
1	准备操作系统		无	0-5
2	下载Apollo项目代码		git clone git@github.com:ApolloAuto/apollo.git # SSH 方式 git clone https://github.com/ApolloAuto/apollo.git # HTTPS 方式	0.5-2
3	配置运行环境	下载安装Docker	wget http://apollo-pkg-beta.bj.bcebos.com/docker_install.sh bash docker_install.sh	0.1-1
4		准备Docker容器	bash /apollo/docker/scripts/dev_start.sh	3-12
5		进入Docker容器	bash /apollo/docker/scripts/dev_into.sh	0
6	代码构建、编译		/apollo/apollo.sh build	0.5-2

注意事项

- 整体耗时较长，需要耐心操作，耐心等待下载、安装、构建等过程。
- 在下载、编译的过程中可能会出现偶发性失败，可以再尝试1-2次，问题或许会不再出现。
- 确保自己的网络连接、硬盘剩余容量、内存占用率处于合理状态。

3. 环境搭建与工具使用

3.2 搭建Apollo环境

脚本简析

dev_start.sh脚本的主要功能是管理和启动适用于Apollo开发环境的Docker容器。

1. **初始化和设置：**脚本首先定义了相关的路径和变量，如当前目录、缓存目录以及Docker仓库。
2. **版本和镜像定义：**根据架构和不同的需求，定义了不同版本的Docker镜像标签，例如VERSION_X86_64、TESTING_VERSION_X86_64、VERSION_AARCH64等。
3. **命令行选项解析。**
4. **镜像管理：**根据指定的参数或默认设置来确定要使用的开发环境Docker镜像。然后docker_pull()函数从Docker 仓库拉取指定的镜像，如果本地已经下载，就不再重复下载。
5. **设备和卷挂载。**
6. **环境检查和设置：**检查主机操作系统环境、检查目标架构、检查时区和地理位置等。
7. **启动 Docker 容器：** main()函数是脚本的入口点，按顺序执行上述步骤，最终通过Docker命令启动开发环境容器，并设置相关的环境变量和挂载选项。
8. **用户指示和清理：**可以选择stop参数，通过stop_all_apollo_containers()函数停止所有运行中的 Apollo 容器。

dev_into.sh脚本通过调用docker exec 进入已经启动的容器。

4 代码调试运行和分析

4. 代码调试运行和分析

4.1 本地运行

1. 获取record数据包

```
mkdir /apollo/data/record
wget https://apollo-system.cdn.bcebos.com/dataset/6.0_edu/demo_3.5.record -P /apollo/data/record/
```

2. 播放原始record数据

```
cyber_recorder play -f /apollo/data/record/demo_3.5.record -l
```

3. 在SimControl模式下运行

```
/apollo/scripts/bootstrap.sh
```

```
cyber_recorder play -f /apollo/data/record/demo_3.5.record -k /apollo/planning
cyber_launch start modules/planning/planning_component/launch/planning.launch
```

4. 增加调试日志，再运行

参考

- apollo上机实践教程
- apollo规划模块实践
- apollo8.0 本地调试仿真

4. 代码调试运行和分析

4.2 云端仿真

Apollo仿真调试

1. 参考课程《*自动驾驶新人之旅(9.0版)*》-*第三课：仿真自动驾驶任务*，创建或者获取相应的场景。
2. 在Apollo Studio的云实验室中，打开《*借道绕行场景仿真调试（9.0版）*》，实现换道绕行任务的仿真运行。

【注】实验课的云仿真只支持修改配置，不支持修改代码，所以不能做真正的代码调试。

Apollo调试运行遇到的问题

- 自动驾驶开发者需要了解一定的**仿真知识**，包括数据集、场景、logsim、worldsim、评测指标等，才能比较有效地进行算法模块的仿真调试，特别是Prediction、Planning和Control。
- 相比起国外著名项目，Apollo的**使用说明文档**还待完善，存在不少缺失或者矛盾的地方。
- Apollo开源的各个component代码且质量较高，但是开源的调试运行仿真工具比较粗糙：
    - **Apollo本地运行**的功能有限，不够便捷（比如需要手动重启复位）；但是可以修改代码，进行更充分地代码调试；
    - **Apollo在线仿真**支持的功能更复杂和丰富，但是无法实验课提供的线上环境无法修改调试代码；
    - 要想充分地进行代码仿真测试，需要花钱采购**企业解决方案Apollo X**。

4. 代码调试运行和分析

4.3 代码分析

Routing介绍和代码分析

- apollo介绍之Routing模块(六)
- 解析百度Apollo之Routing模块

Planning介绍和代码分析

- 参考资料：apollo介绍之planning模块(四)
- 框架设计：https://apollo.baidu.com/community/course/outline/776?activeId=20615
- 《PnC基础》课程：https://apollo.baidu.com/community/online-course/776
- 《PnC进阶》课程：https://apollo.baidu.com/community/online-course/809
- Planing 2.0综述：https://apollo.baidu.com/community/article/1111

5 C++实现神经网络模型

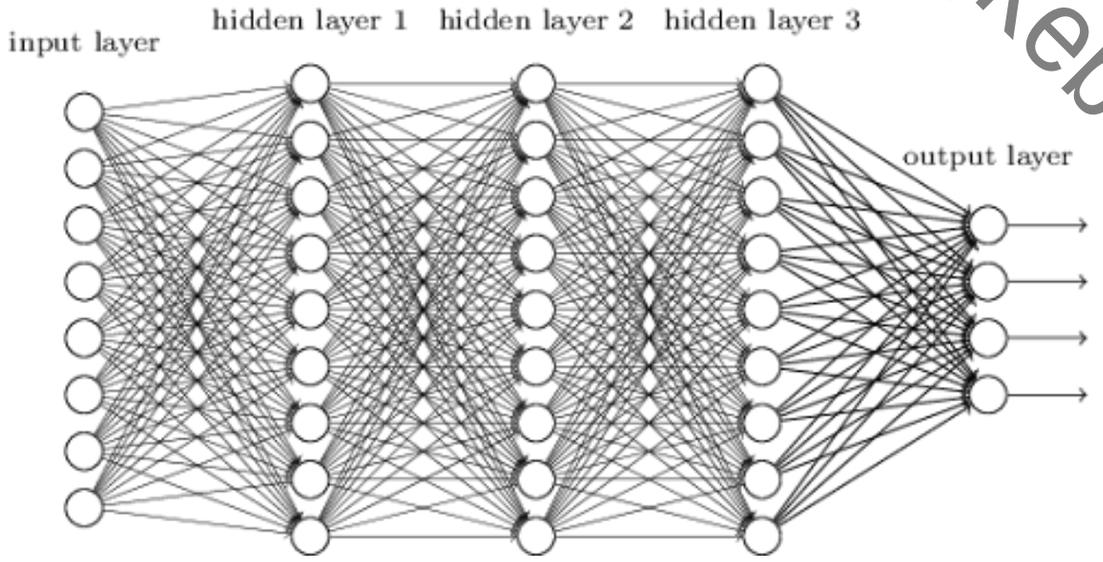
5. C++实现神经网络模型

5.1 设计分析

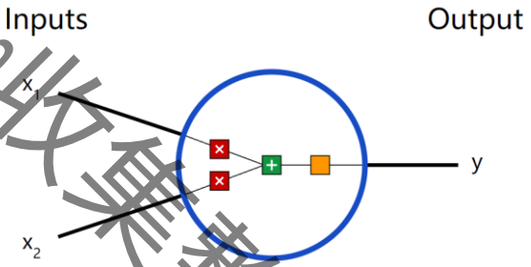
基本数据结构设计

- **神经元**：定义一个类，表示神经元，包含其权重、偏置值以及激活函数。
- **神经层**：定义一个类，表示神经层，用于包含多个神经元。
- **神经网络**：定义一个类，表示神经网络，将各层按顺序连接，形成完整的神经网络模型。

但是，考虑到同一层的神经元neuron的结构是简单且相同的，我们考虑不必为每一个神经元也定义类对象，而是直接使用矩阵来更方便地表达和计算。



Neural Network, link



Neurons

5. C++实现神经网络模型

5.1 设计分析

各种神经网络层

- **输入层Input Layer：** 输入层是网络的第一层，用于接收原始数据。
- **卷积层Convolutional Layer：** 通过卷积核（滤波器）在输入数据上滑动，提取局部特征。
- **池化层Pooling Layer：** 对输入进行下采样，减小尺寸，减少计算量和过拟合。
- **全连接层Fully Connected Layer/Dense Layer：** 每个神经元与前一层的所有神经元相连接。
- **循环层Recurrent Layer：** 用于处理序列数据，具有记忆功能。
- **归一化层Normalization Layer：** 对数据进行归一化处理，标准化输入特征。
- **Transformer Layer：** 基于注意力机制，擅长处理序列数据。
- **激活层Activation Layer：** 应用非线性激活函数。
- **输出层Output Layer：** 输出层是网络的最后一层，用于生成预测结果。
- **More……**

参考

- https://aistudio.baidu.com/projectdetail/5067820
- https://zh.d2l.ai/chapter_multilayer-perceptrons/index.html

5. C++实现神经网络模型

5.1 设计分析

神经网络的操作

1. 模型数据的保存和加载
2. 权重初始化
3. 前向传播
4. 反向传播
5. 并行计算
6. ……

5. C++实现神经网络模型

5.2 代码分析

代码位置

//apollo/modules/prediction/network

- Python对于使用者很方便，但也因为它屏蔽了很多底层实现，让我们对NN的理解不够深刻；
- 阅读代码的C++实现，可以让我们更好地理解原理，理解它的实现。

- 使用Gtest进行测试与调试；
- 尝试进一步补充和完善。

5.3 类似的参考项目

- artificial-neural-network, a basic implementation of an artificial neural network written in C++.
- Nerve, a basic implementation of a neural network for use in C and C++ programs.
- EidNN, about implementation-from-scratch fun and becoming familiar with neural networks and deep learning.

6 实战项目分享

6. 实战项目分享

6.1 实际项目的复杂性

原以为的 VS. 实际上的	
一种编程语言	多种编程语言
熟悉C++就可以快速读懂	需要足够的背景知识和对设计方案的了解
一两个依赖的库	大量的依赖库
基本没有工具	多样的工具
单一的流程	复杂的流程
基本不需要文档	需要参考和编写很多文档
自己假想的需求	明确的、真实的需求

大型项目的复杂性

- 一个以C++为主的项目，往往也有大量内容与C++无关。
- 一个大型项目有诸多子功能、管理工具、开发者工具，所以感到茫然、困惑是很正常的事。
- 实现了类似功能的大型项目，往往有大量的设计逻辑和工具是类似的。了解其中一个，再使用其他系统就会驾轻就熟。
- 记住所有的接口API和工具命令是不切实际的，通过整理良好的参考文档快速查找是合理的方式。
- 大胆尝试，和构建工具、计算机在对话中熟悉Apollo等大型项目的使用。

6. 实战项目分享

6.2 没有武林秘籍

- **武林秘籍：**“武林秘籍”通常指的是传说中武侠小说里的神奇秘籍，能让人迅速提升武功。
- **银弹：**“银弹”（Silver Bullet）来源于西方传说，银弹可以一枪杀死狼人等不死生物。

弗雷德·布鲁克斯（Frederick P. Brooks）在他的著作《没有银弹：软件工程的本质性与附属性工作》中提出，没有任何一种技术或方法能够一次性解决软件开发中的所有困难。比喻在编程领域，不存在可以一劳永逸解决所有编程问题的单一方法或工具。

“没有武林秘籍”和“没有银弹”是比喻用语，我在这里想表达的是：

- 不存在一种举世无双的招式可以打败所有对手；
- 不存在一种迅速提升编程能力或解决复杂问题的方法、资源或技巧；
- 也不存在一个项目让我们掌握所有编程知识。

- **没有万能的解决方案。**正如布鲁克斯所说，软件开发中没有银弹，没有一种方法能解决所有问题。C++编程的学习也是如此，掌握C++需要系统的学习和长期的积累。
- **基础和实践是关键。**没有捷径可以代替扎实的基础和大量的实践。学习C++的“秘籍”在于掌握基础之后，通过项目和练习不断提高。

6. 实战项目分享

6.3 实战项目分享-传统项目

计算器

- 功能：实现基本的加、减、乘、除运算。
- 目的：练习基础输入输出、运算符和控制结构。

学生成绩管理系统

- 功能：输入学生成绩，计算平均分、最高分和最低分。
- 目的：练习数组和基本的数据处理。

图书管理系统

- 功能：包括图书的添加、删除、查询和借阅归还功能。
- 目的：练习类和对象、文件操作、数据结构。

简单的聊天程序

- 功能：实现局域网内的简单聊天功能。
- 目的：练习网络编程、套接字编程和多线程编程。

贪吃蛇游戏

- 功能：经典的贪吃蛇游戏，带有图形界面。
- 目的：练习图形编程、事件处理和基本的游戏逻辑。

6. 实战项目分享

6.3 实战项目分享-小型项目

滤波器

- 功能：实现一组滤波器，以实现对数据的平滑滤波。
- 参考代码：//apollo/modules/common/filters/

阻塞队列

- 功能：实现阻塞队列/消息队列，以实现生产者、消费者多线程读写。
- 参考代码： //apollo/cyber/blocker/

线程池

- 功能：预先创建并管理一组线程的机制，用于执行提交给它的任务队列。
- 参考代码： //apollo/cyber/base/thread_pool.h

6. 实战项目分享

6.3 实战项目分享-小型项目

协程

- 功能：实现轻量级线程——协程的主要能力和接口。
- 参考代码： //apollo/cyber/croutine/

Protobuf文件操作

- 功能：将Portobuf的接口进行封装，以方便高效地进行Protobuf文件的读写及其他操作。
- 参考代码： //apollo/cyber/common/file.h

用QP平滑轨迹

- 功能：利用二次规划求解器OSQP对样条曲线进行轨迹平滑。
- 参考代码： //apollo/modules/planning/planning_base/math/smoothing_spline/osqp_spline_1d_solver.cc

A*路由

- 功能：用A*算法寻找地图上的最短路径。
- 参考代码： //apollo/modules/routing/strategy/a_star_strategy.cc

6. 实战项目分享

6.4 实战项目分享-中型项目

神经网络模型

- 功能：实现一种特定的网络模型。
- 参考代码：//apollo/prediction/network

XML文件读取和解析

- 功能：实现简单XML文件的读取和解析。
- 参考代码：**TinyXML2**

muduo

- **描述**：muduo是一个开源的、基于Reactor模式的**C++网络库**，提供了**高性能非阻塞I/O、事件驱动机制、线程池管理和定时器等能力**，可以高效地构建高性能服务器应用。
- 代码地址：https://github.com/chenshuo/muduo/tree/master

6. 实战项目分享

6.4 实战项目分享-中型项目

车辆控制模块

- **任务：**开发一个车辆控制模块，能够跟随规划轨迹的执行。
- **步骤：**
    1. **接收输入消息：**使用Cyber RT接收输入的车辆期望轨迹。
    2. **车辆控制算法：**编写C++代码实现PID或MPC控制器。
    3. **发布输出消息：**发布计算好的油门、刹车、方向盘转角等基本车辆控制指令。
    4. **数据分析：**用Python绘制输入和输出的数据对比曲线，分析算法的效果。

感知障碍物检测模块

任务：开发一个障碍物检测模块，使用传感器数据识别道路上的障碍物。

步骤：

1. **接收输入消息：**使用Cyber RT接收输入的传感器数据（图片或点云）。
2. **数据处理：**处理传感器数据，使用C++实现数据滤波和特征提取。
3. **障碍物识别：**实现基本的障碍物识别算法（如基于形状或颜色的检测）。
4. **发布输出消息：**发布检测得到的障碍物位置、姿态信息。
5. **结果分析：**用Python或JavaScript写一个小工具，将原始传感器数据和检测障碍物结果同时绘制出来，对比分析效果。

6. 实战项目分享

6.4 实战项目分享-中型项目

异常监控与报警模块

任务：开发一个监控与报警模块，用于实时监控自驾系统的运行状态，并在发生异常时触发报警。

步骤：

1. **数据采集：**编写代码采集各模块的状态数据，例如CPU使用率、内存使用率、模块运行状态等。
2. **数据发布：**利用Cyber RT发布这些状态数据，使得其他模块能够订阅和使用这些数据。
3. **实时检测：**编写C++代码，实现对采集数据的实时监控和检测，一旦发现超过阈值，记录异常事件。
4. **报警触发：**设计报警条件，如连续三次监测值超过阈值，则发布异常信息。
5. **报警通知：**利用Cyber RT message将报警信息发布给预设的报警模块，或者支持写入关键日志、发布给前端UI模块等多种方式。

系统运行性能分析工具

任务：开发一个性能分析工具，用于记录和分析系统中各模块的性能表现。

步骤：

1. **数据采集：**调用OS提供的接口或者读取OS的特定文件，定期采集各模块的性能数据（如CPU使用率、内存使用率、网络延迟）。
2. **数据存储：**设计数据存储结构，将采集到的性能数据发布为消息或者直接保存到文件中。
3. **数据处理：**实现数据的统计分析功能，计算平均值、最大值、最小值等性能指标。
4. **UI展示：**利用图形界面展示性能分析结果，例如折线图、柱状图等。
5. **报告生成：**编写代码生成性能分析报告，支持导出为PDF或其他格式。

6. 实战项目分享

6.5 实战项目分享-大型项目

- ROS
- Apollo
- 前面章节提到的各种库，如Eigen、OpenCV、Protobuf
- ……

大型项目往往由多人团队共同开发和维护，个人想开发大型项目是不太可能的，甚至想要运行、调试都很困难；比如自动驾驶的仿真调试，都要专门学习、熟悉背后的原理和使用流程才能开始。

我们可以学习大型项目的整体架构、设计方案，以及学习其中的子模块。

6. 实战项目分享

6.6 建议

找到目标和真实的需求

脱离现实目标和真实需求的编程项目，容易让人放弃，特别是对于C++语言来说；

选择所喜欢的领域（自动驾驶/互联网/游戏/.....），找到好奇的问题，再去找解决方案。

站在巨人的肩膀上

不要从零开始实现项目，而是从学习、理解别人的代码开始：

1. 选择一个成熟项目，熟悉其主要功能和接口；
2. 选择其中一个功能子集或功能簇；
3. 参考源码，重新实现一遍目标功能。

改进也是你的成果

并不是全新的项目才有价值，对一个现有的项目进行补充完善，解决更多问题，也是你的成就。

例如：Google的很多项目并没有多么神奇，在之前就已经存在很多类似的解决方案了；

但Google成功的项目往往在已有项目的基础上，不断地优化、完善，提供**易用的软件接口、各种语言的支持、丰富完整的使用文档、保持兼容、始终开放。**最终Google的项目成为使用最广泛的项目，也就成了事实上的业界标准。

6. 实战项目分享

6.6 建议

Google Repositories in Github, link

7 小结与作业

7. 小结与作业

小结

- 介绍了百度Apollo的背景，说明了选择Apollo的原因，并提供了相关的学习链接；
- 分析了Apollo Cyber RT提供的主要机制及其实现；
- 介绍了Docker，并基于Docker搭建了Apollo运行环境，说明了Apollo的开发者工具；
- 进行了Apollo的本地调试和云上仿真，并简要分析了Routing和Planning的代码；
- 分析了神经网络模型代码的设计和实现；
- 分享了对项目实战的看法，并提供了十多个实际项目的例子供参考。

7. 小结与作业

作业

参考上一小节，选择一个项目进行实现。

实际的C++项目通常包含以下关键要素和组件，最好在自己的项目里也包含这些内容：

1. **源代码文件**
   - .cpp文件：C++ 源文件，包含实现代码。
   - .h或.hpp文件：头文件，包含函数和类的声明。
2. **构建系统文件**
   - CMakeLists.txt：CMake的构建配置文件。
   - BUILD：Bazel的构建配置文件。
   - Visual Studio项目文件：适用于Windows系统，包含.sln和.vcxproj文件。
3. **文档**
   - README.md文件：项目的基本介绍和使用说明。
   - 开发文档：如 API 文档、设计文档等。
4. **测试**
   - 单元测试文件：使用框架如Google Test或Boost.Test编写的测试代码。
   - 测试数据：用于测试的输入输出文件。
5. **资源文件**
   - 配置文件（例如 .ini, .json, .xml）：用于配置应用程序。
   - 图像、声音、文本等资源文件。

7. 小结与作业

作业

另外，对于大型项目来说，还可能有如下文件，根据实际情况来增加：

1. **依赖管理**
   - 包管理器配置文件，例如Conan、vcpkg或Hunter，用于管理第三方库依赖。
2. **脚本**
   - 构建脚本：如build.sh或build.bat，用于自动化构建过程。
   - 测试脚本：如run_tests.sh，用于运行测试套件。
3. **版本控制**
   - .gitignore文件：指定应忽略的文件和目录，通常与 Git 配合使用。
   - 版本控制历史：通过 Git 或其他版本控制系统记录的代码变更历史。
4. **CI/CD配置**
   - CI/CD 配置文件：如GitHub Actions (.github/workflows)、GitLab CI (.gitlab-ci.yml)等，用于自动化构建、测试和部署。
5. **安全和合规**
   - 许可证文件（LICENSE）：说明项目的开源许可证。
   - 安全策略文件：如 SECURITY.md，描述安全报告的流程。

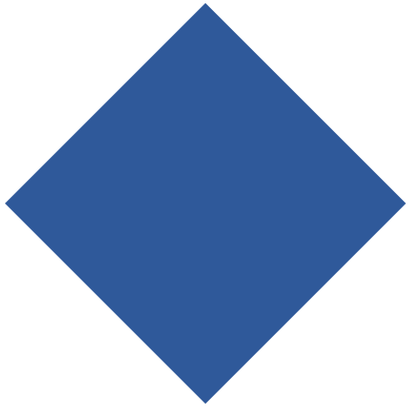
公众号

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！



End