计算框架Cyber RT

目录

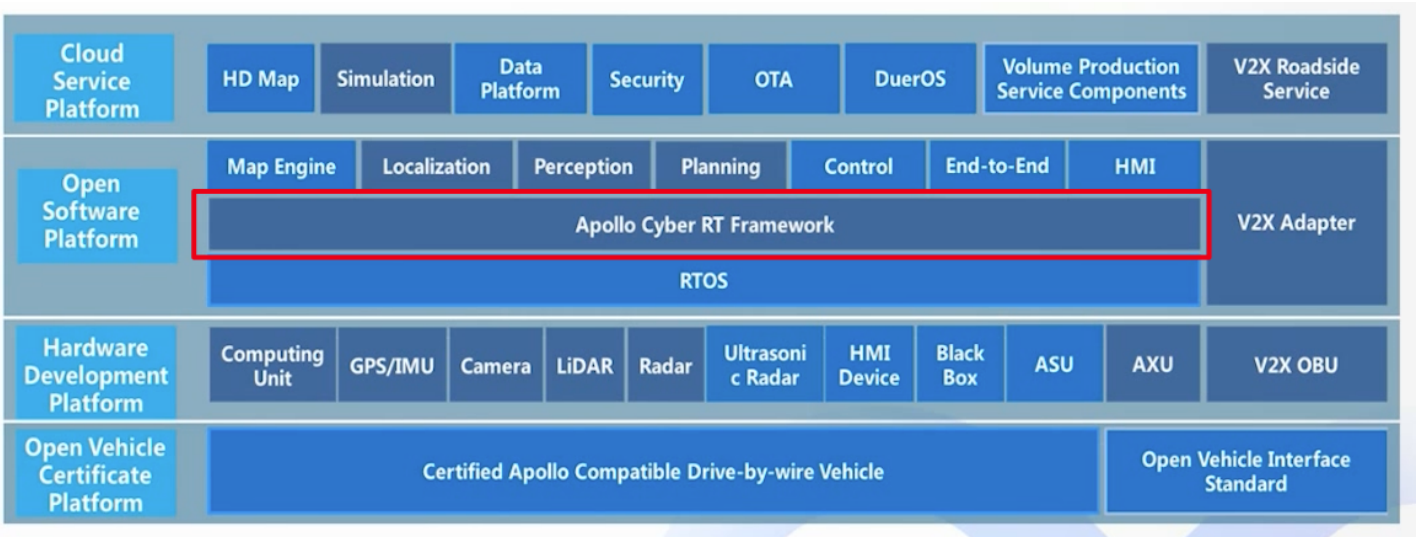
> 参考资料
>
> 1. Apollo 3.5 计算框架（Cyber RT）设计分享
> 2. Cybertron Tutorials
> 3. 自动驾驶 汽车平台技术基础（第4、5章）
> 4. apollo介绍之Cyber框架(十一)
> 5. 【精选】自动驾驶Apollo源码分析系统,CyberRT篇(一):简述CyberRT框架基础概念-CSDN博客
> 6. Cyber RT学习笔记 --- 1.Cyber RT框架介绍-CSDN博客
>
> 源码地址
>
> 1. 源码地址（github）
> 2. 源码地址（icode）

一. 基础功能

1.1 概述

Cyber RT位于apollo架构的软件平台层，是apollo自动驾驶平台的计算框架，于apollo3.5版本正式引入（apollo3.5版本以前使用的ros）。

软件平台层：RTOS为实时操作系统，是确保在规定时间内完成指定功能的操作系统。上层的地图、定位、感知、pnc等算法模块，是基于计算框架开发的功能组件。



Apollo 3.5 Architecture

Cyber RT是一个**分布式收发消息**（通信 第4节 介绍）和**调度**（调度 第5节 介绍）的框架，同时提供一系列**工具**和**接口**辅助开发和定位问题。Cyber RT框架核心理念是基于组件的概念构建、加载各功能模块。组件有预先设定的输入输出。每个组件就代表一个专用的算法模块。框架可以根据所有预定义的组件生成有向无环图（DAG）（拓扑结构 第2节 介绍）。

基于Cyber RT，各算法模块可以独立开发自己的Compoent，通过Node与其他Compoent（功能模块）建立通信。得益于这种分布式消息收发机制，各模块相互解耦，上层功能模块开发者 既不必关心本模块pub的消息最终到哪里，也不必关心自身sub的消息来自于哪个模块。只需要根据需要通过Node的reader和writer发布和订阅消息即可，至于消息如何流转框架已经为我们做好了。

1.2 基础组件

1. 节点（Node）

在Cyber RT框架中，节点是最基础的单元，每个节点都有各自独立的算法程序。它能够基于信道、服务等功能与其他节点进行通信，各个节点之间进行通信即可形成拓扑关系，并完成指定任务。通过使用节点，可将代码和功能解耦，提高了系统的容错能力和可维护性，使系统简化。同时，节点允许了Cyber RT能够布置在任意多个机器上并同时进行。

2. Compoent

Component是cyber中封装好的数据处理流程，对用户来说，对应自动驾驶中的Planning Component, Perception Component等，目的是帮助我们更方便的订阅和处理消息。实际上**Component模块在加载之后会执行"Initialize()"函数**，这是个隐藏的初始化过程，对用户不可见。在"Initialize"中，Component会创建一个Node节点，概念上对应ROS的节点，**每个Component模块只能有一个Node节点**，也就是说每个Component模块有且只能有一个节点，在Node节点中进行消息订阅和发布。

3. module

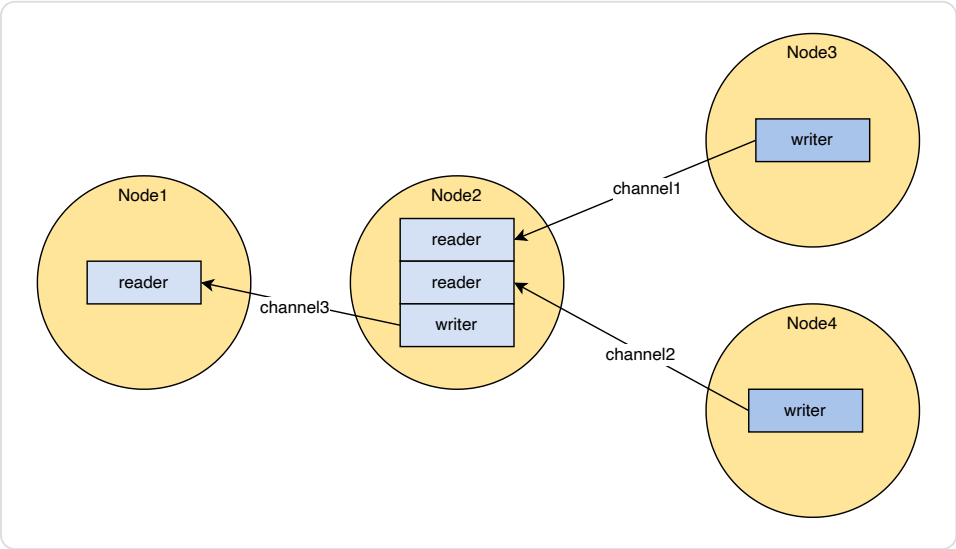
若子系统过于庞大，则可将子系统内部的算法重新拆解成多个Component来构建子系统。原则上每个功能模块对应一个module，一个module可以包含很多个compoent。由这些compoent共同构成这个功能模块。

4. 信道（Channel）

若需要完成节点之间的通信，则需要建立一条信息传输通道，在Cyber中称为信道，节点可以将消息发送进入某一指定的信道之中。若有其他节点定义接口接收此信道的消息，则可完成消息的收发过程。若没有，则消息也依然存在于信道之中。

5. Reader/Writer

Reader是消息的接收方，Writer为消息的发送方。在Node节点中可以创建Reader订阅消息，也可以创建Writer发布消息，每个Node节点中可以创建多个Reader和Writer。



6. 服务（Service）

服务是Cyber中另一种通信方式，与信道通信相同，基于服务的通信也需要消息的收发节点。但与信道不同的是，服务需要两个节点之间完成请求或应答才可以完成通信。（双向通信，需要关心对方的应答）

7. 调度（Schedule）

每一个Compoent对应一个任务（协程），调度的目的是如何有效分配资源，调度这些任务。

1.3 优势

（1）加速算法（调度算法、协程等）

（2）简化部署

（3）增强自动驾驶能力

二. 拓扑结构

在图论中，如果一个有向图从任意顶点出发无法经过若干条边回到该点，则这个图是一个有向无环图（DAG图）。

基于Cyber RT，各个功能模块（组件Compoent）会生成DAG关系图，各功能模块为DAG图中的一个节点（Node），可独立进行开发。通过dag文件配置动态生成计算流程图并执行。

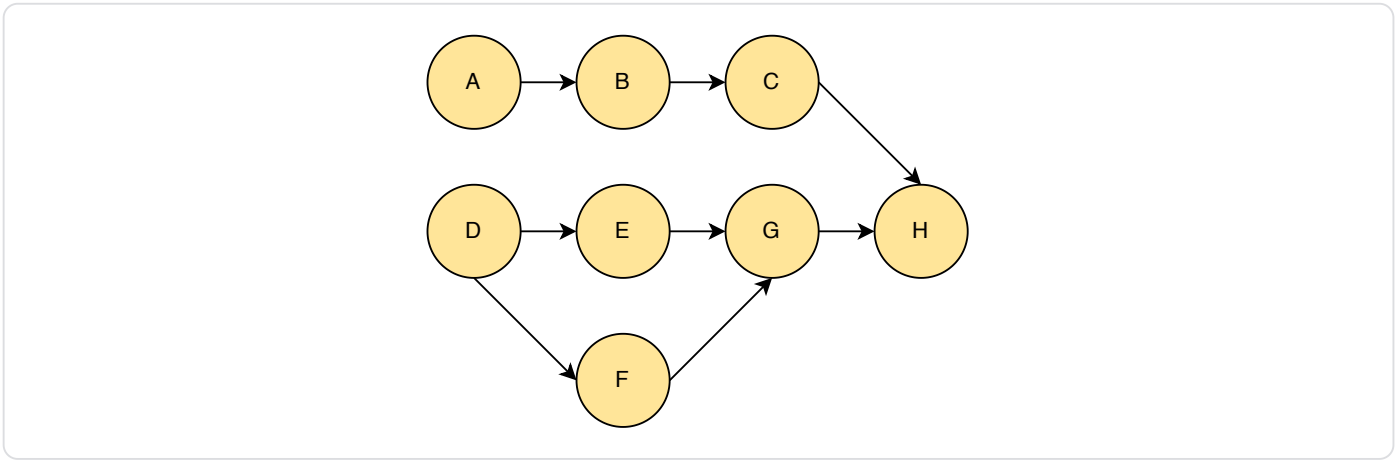
这种拓扑结构第一个显著特点是**去中心化**，在一个分布有众多节点的系统中，每个节点都高度自治，避免了中心故障将影响整个系统的缺陷。

第二个特点是**有多个出度**（如图中D的输出同时发送给E、F，G可以同时接收E、F的输入进行处理），因此可以处理多个出度连接的节点，这样的特征可以加快处理任务的速度，且其拓展性得到提高。利用DAG拓扑

框架可将任务分成各个子任务，每个子任务独立完成各自计算，并**形成输入输出依赖关系（拓扑序）**。

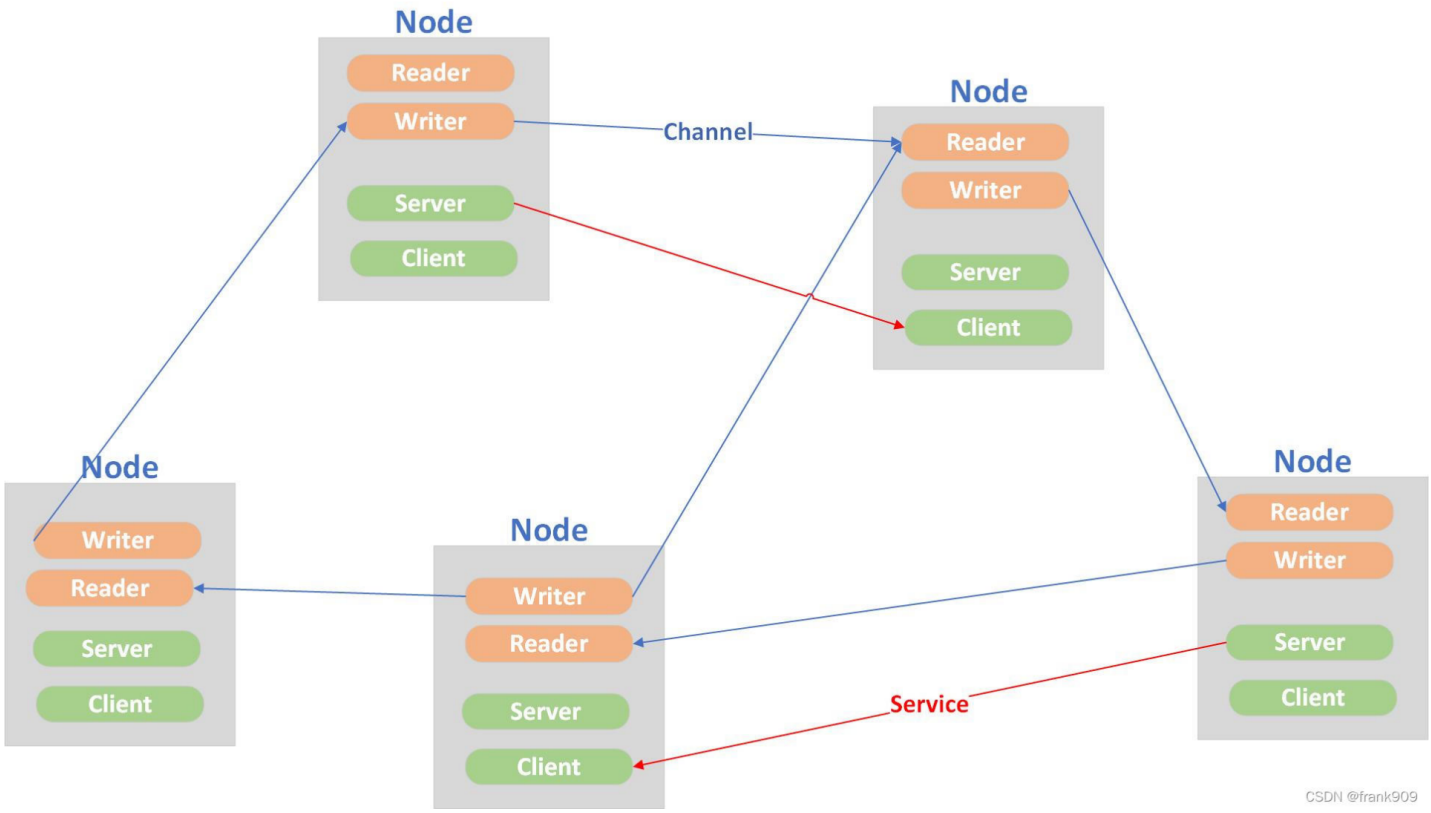
有向无环图的偏序关系可以在调度有着先后顺序限制的系统任务中发挥作用。

第三个特定是**功能解耦，拓展性强**，拓扑结构中的每个节点对应一个功能模块，各功能模块相互解耦，被封装成独立可拆分的计算单元。应用模块可以实时调整某一算法策略而不影响整体算法链路的正常执行。实现**动态可“插拔”**。



DAG图示意

在Cyber RT中，各节点通过reader和writer形成拓扑结构，可以动态的加入和卸载。实际中系统以数据流驱动，当数据通过DAG描述的边进行流动到达节点后进行处理。



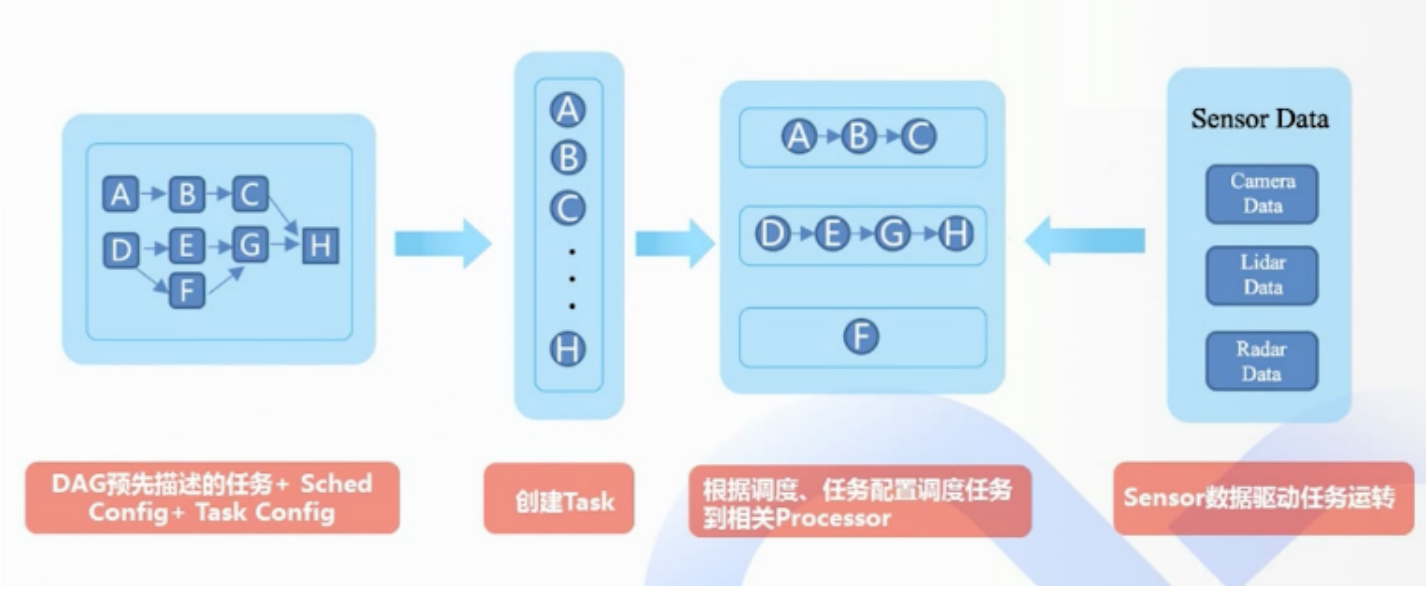
Node 拓扑结构示意图

三. 运行流程和架构

3.1 Cyber RT 运行流程

图中的每一个task相当于一个协程，通过cyber RT内部的调度器将协程分配到各Processor中（即 Native Thread）。

图右侧sensor中的数据是驱动整个系统task运转的源头（数据驱动）



3.2 Cyber RT 架构图

调度的每一个task其实是一个协程。为了让开发者有一个更好的使用方式，设置一个抽象层（Task Abstract Interface），开发者在使用的时候通过创建compoent，compoent中的Proc函数可以理解为是node中注册的callback，当有数据来的时候会调用到指定的compoent的Proc接口处理。

框架结构各层的功能：

第一层：主要是Apollo实现的基础库，比如有Lock-Free的对象池，Lock-Free的队列等。实现基础库可以减少相关的依赖并提高运行效率。

第二层、三层：主要是负责管理Cyber的通信机制，包括服务发现（负责管理通信中的Node节点）和Publish-Subscribe通信机制。并且Cyber也支持跨机、进程间、进程内通信，而且会根据不同的数据传输和业务逻辑自动选择效率最高最匹配的通信方式来进行通信。

第四层：主要是对传输过后的数据进行缓存，并会根据不同传感器得到的数据进行融合得到一个可处理可读的数据发送给另一个模块。

第五、六层：主要是管理每一个任务的调度和数据处理。

第七层：提供给开发者的一些API接口，让开发者有更多可操作性，提高开发效率。

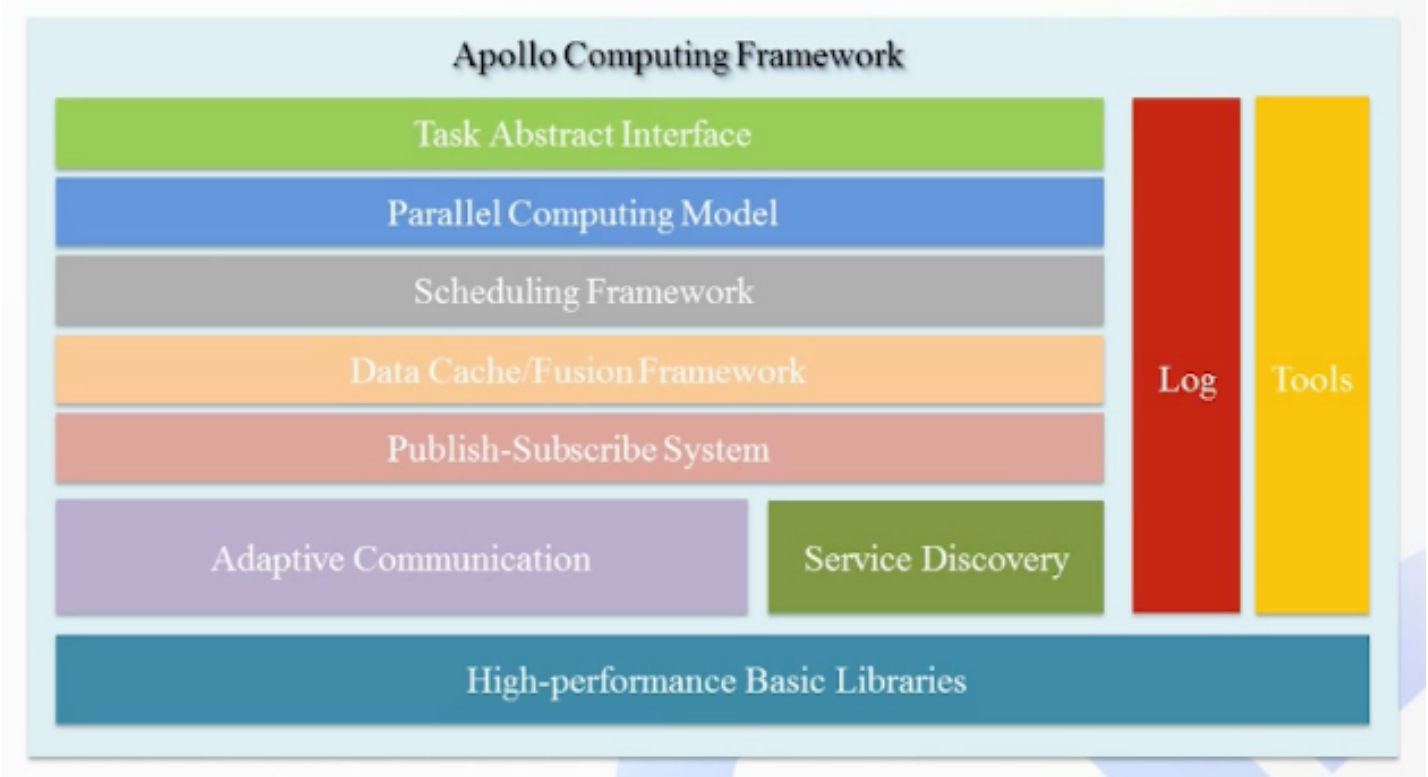
通过架构图中可以看出，Cyber RT的主要功能包括以下方面：

（1）消息队列： 主要作用是接收和发送各个节点的消息，涉及到消息的发布、订阅以及消息的buffer缓存等；

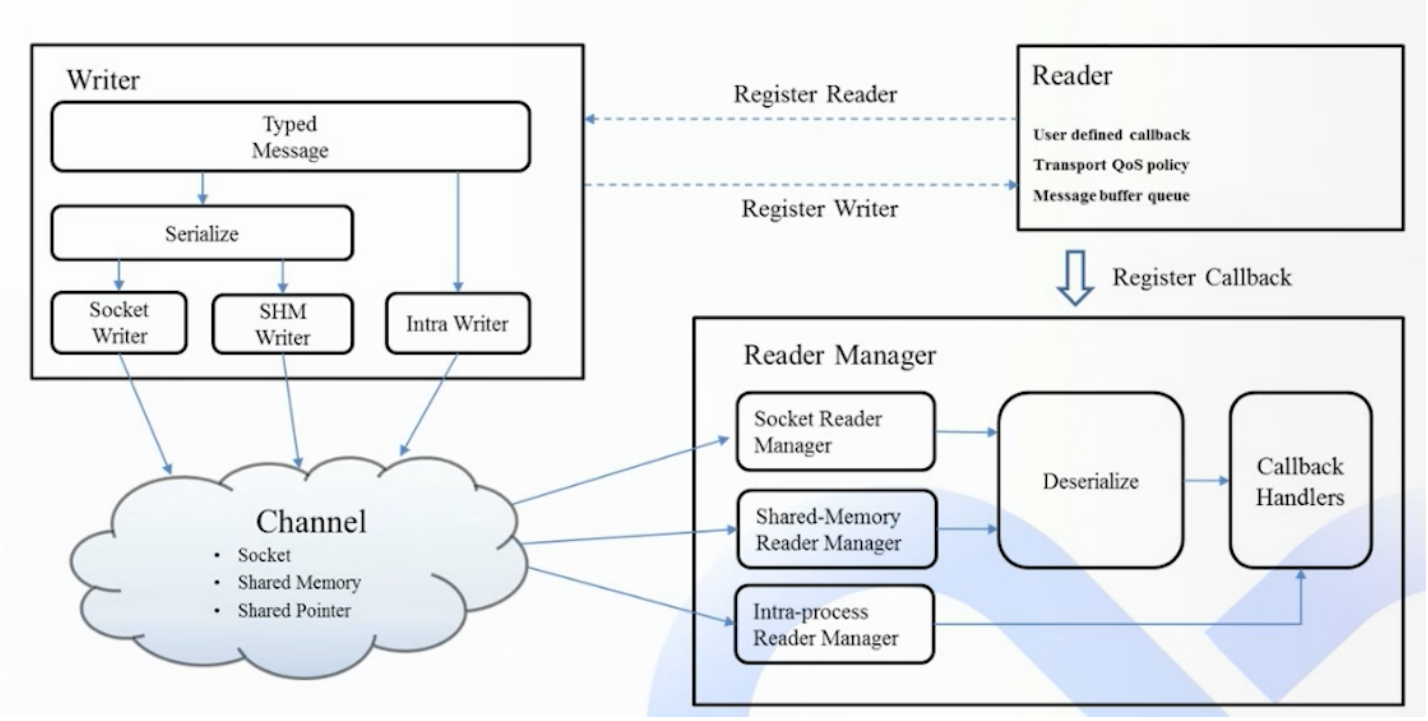
（2）实时调度： 主要作用是调度处理上述消息的算法模块，保证算法模块能够实时调度处理消息；

（3）用户接口： Cyber提供了灵活的用户接口；

（4）开发工具： 提供了一系列的工具包括消息监控(Cyber_monitor)，消息可视化(Cyber_visualizer)，录制/回放工具(Cyber_recorder), ros包录制(rosbag_to_recorder)；



四. 通信组件



4.1 基于信道的通信

基于信道的通信，首先需要定义消息的发送方（Writer）和接收方（Reader），以保证消息可以通过Writer和Reader共同指定的Channel，从一个节点传输到另一个节点，这类通信方式由以下特点：

（1）同一个节点可以同时发送多条消息，也可以同时接收多条消息，即可以同时定义多个Writer和Reader。

（2）基于信道的通信是一种单向通信，消息只能由Writer传输到Reader，而不能够反向传输。

（3）信道中的消息不需要实时应答，也就是说，当一条消息通过Writer送到信道时，可以没有Reader读取消息。当某个Reader读取信道中消息时，信道中可以没有消息。

4.2 基于服务的通信

除了各节点的消息发送和接收之外，很多场景需要在节点之间进行双向通信，并能够获得应答。这就需要利用服务来通信。Service是节点之间通信的另一种方式，不同于Channel的通信方式，Service的一个节点想要获取信息，需要给另一个节点发请求，以此来获得响应，这就完成了节点之间的双向通信。在Service中，发送请求的一方为客户端（Client），接收请求的一方为服务端（Server）。

4.3 三种通信模型

1. **同一进程内通信**

直接传递消息对象的指针，这样可以避免消息数据复制的开销，尤其一些较大的数据。

2. **同主机进程间通信**

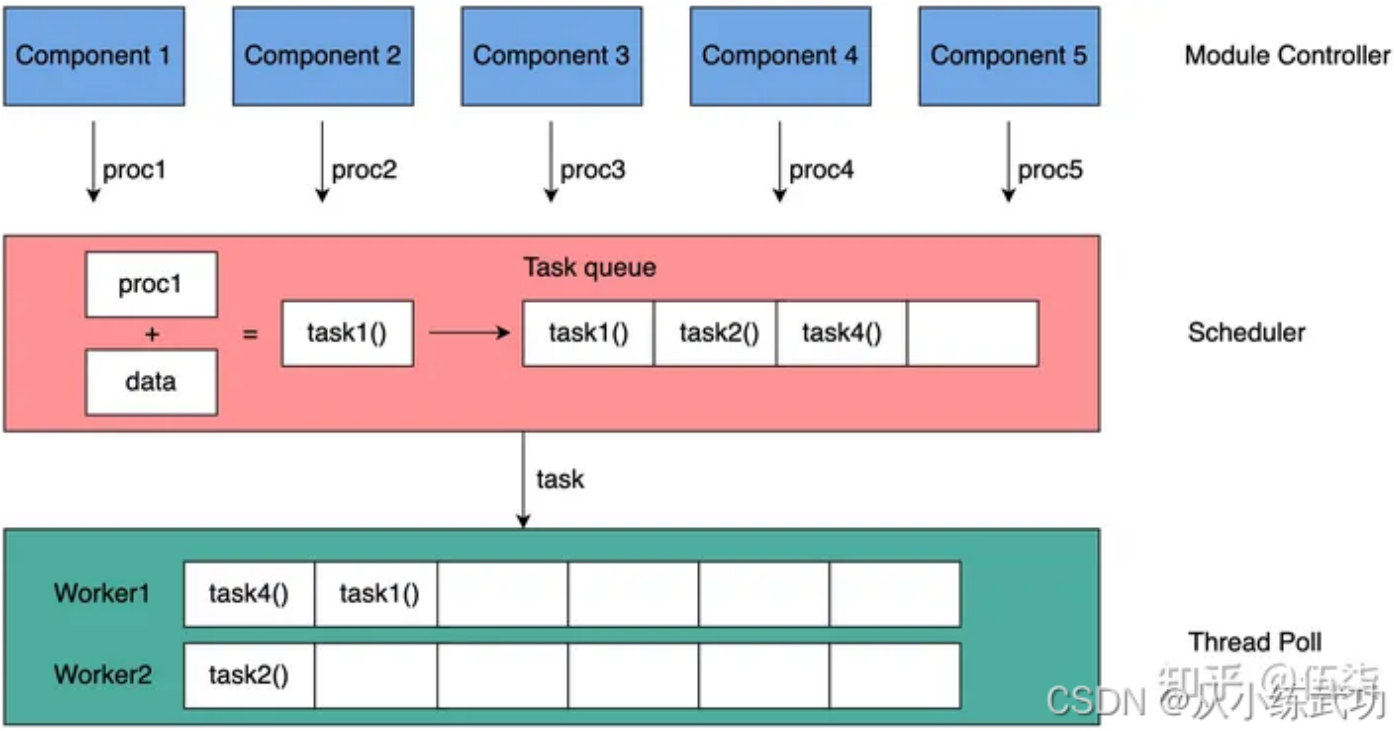
利用共享内存通信，这样不仅可以减少传输中的数据复制，显著提升传输效率，还能够有效满足一对多的传输场景。共享内存通信可以分为3个主要步骤：写入数据、发出可读通知、读取数据。

3. **跨主机通信**

利用socket通传输。目前，跨主机通信采用里第三方的开源库Fast RTPS（实时发布订阅协议）。RTPS是DDS标准中的通信协议，而Fast RTPS是支持RTPS协议版本的一个订阅发布消息的组件，具有高性能，实时性高，多平台支持等优点。

五. 调度策略

如何能够有效分配计算资源，是调度主要解决的问题。Cyber的调度主要是基于任务优先级来并行运行，主要的目的就是保证系统在资源有限的情况下来提高运行效率。如图所示是一个任务调度的示意图，首先在系统初始化时，会对每一个模块的Component进行创建并初始化，每个Component都会有一个proc()函数，proc函数主要是描述了每个需要处理的数据和数据处理的方法。在运行时，每当生成新的数据时，调度器会将数据和对应的proc函数绑定程一个可执行的task，然后放到Task queue中执行，Task queue是一个有序的列表，每当任务入队的时候，会根据任务本身的优先级和周期状态对Task queue里的人物进行重新排序，以确保高优先级的任务能被第一时间执行，Thread Pool是实际执行任务的线程池，根据不同的硬件配置会有固定数量的Worker，每个Worker会持续得从Task queue中读取可执行的任务并进行执行。



cyber RT调度策略图（引自 Cyber RT学习笔记 --- 1.Cyber RT框架介绍-CSDN博客 ）

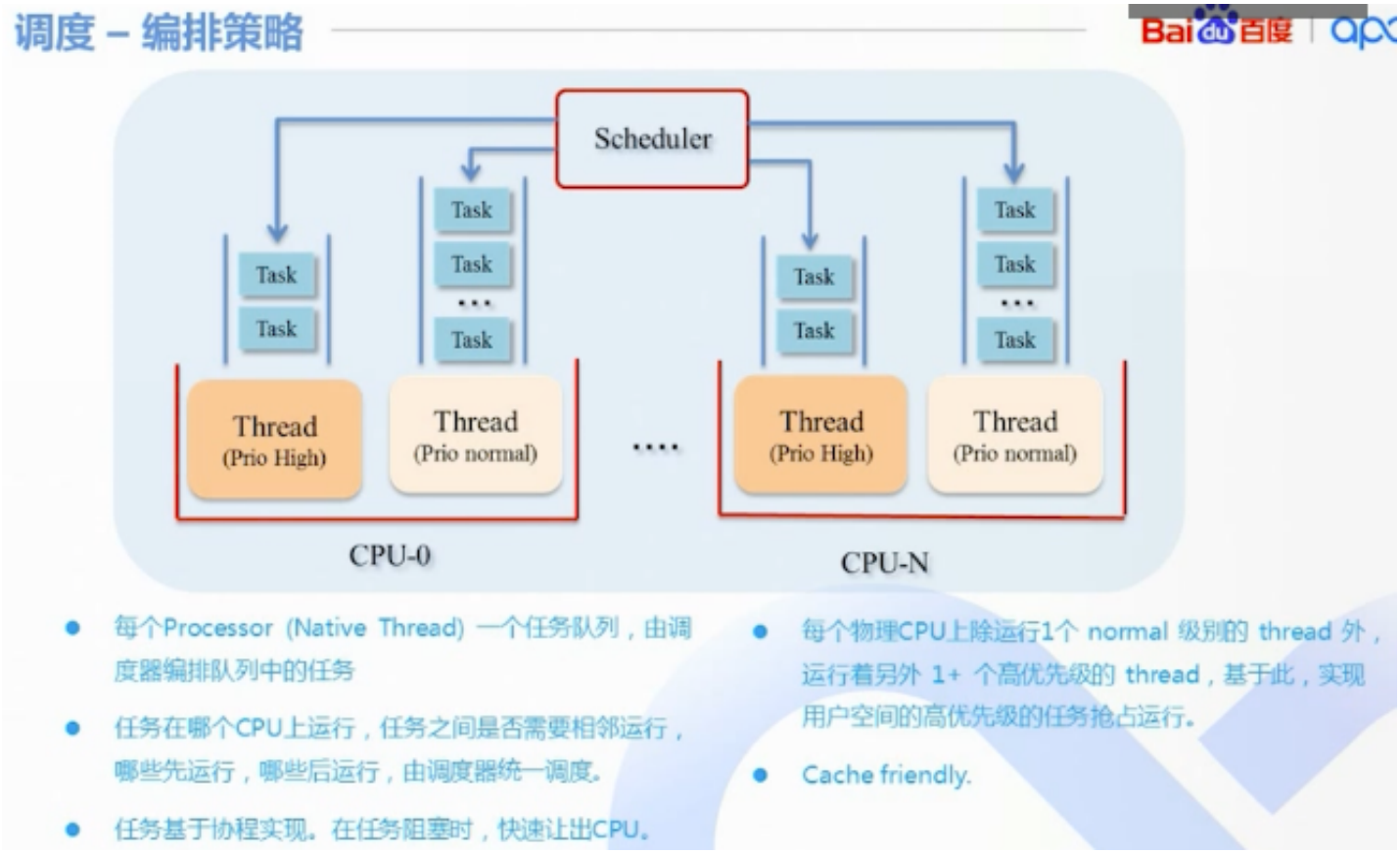
5.1 协程

协程，即线程更上一层的载体，Cyber RT调度器调度有状态的协程在各个线程上运行。协程不仅切换快，而且调度有着高确定性，不像线程的调度完全依赖操作系统。

5.2 编排策略（choreography）

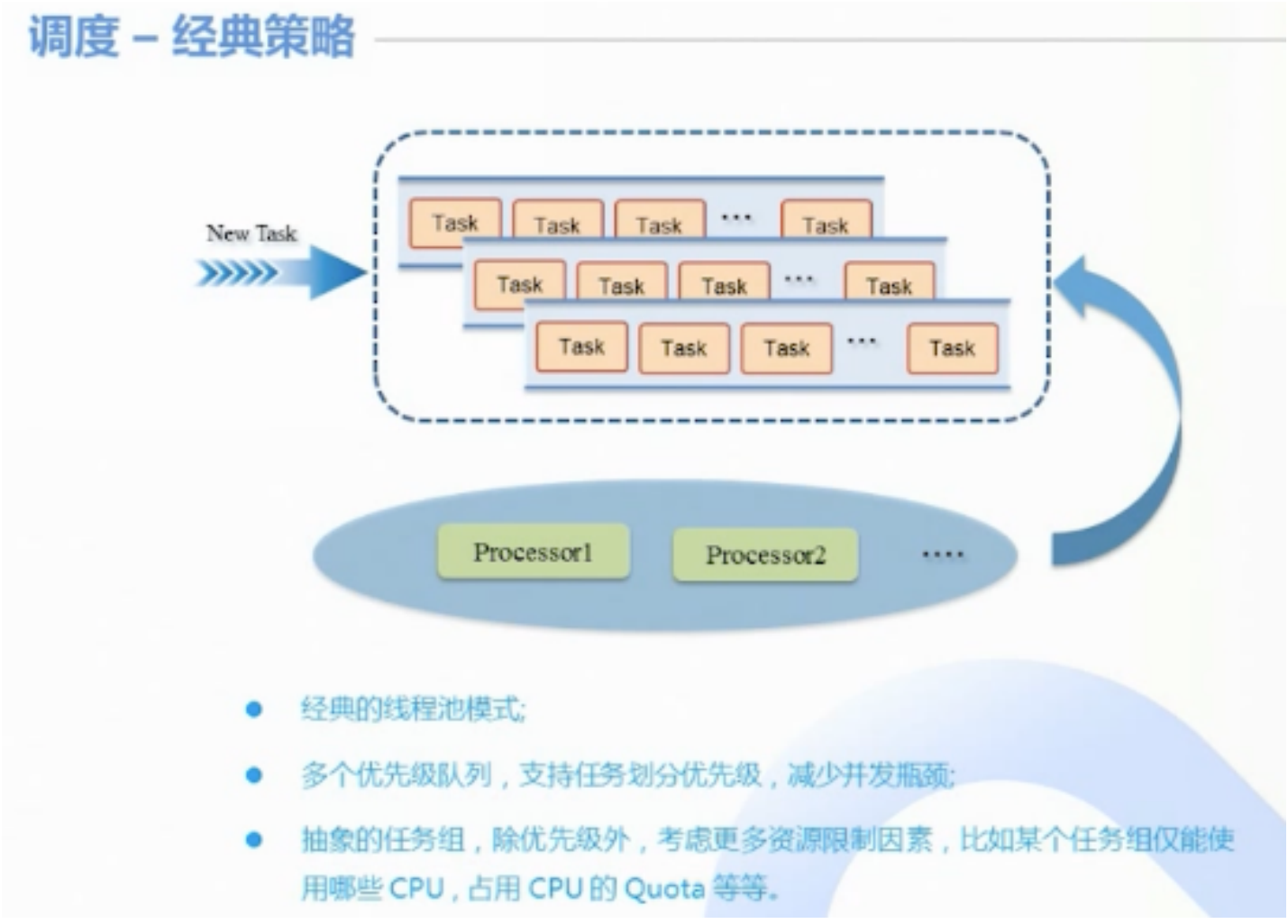
（区别ros，ros中没有调度）

需要对任务足够熟悉，根据任务的依赖执行关系、任务的执行时长、任务CPU消耗情况、消息频率等，对任务进行编排。



5.3 经典策略（classic）

较为通用的调度策略，如果对于车上的DAG结构不清楚时，优先使用该策略。



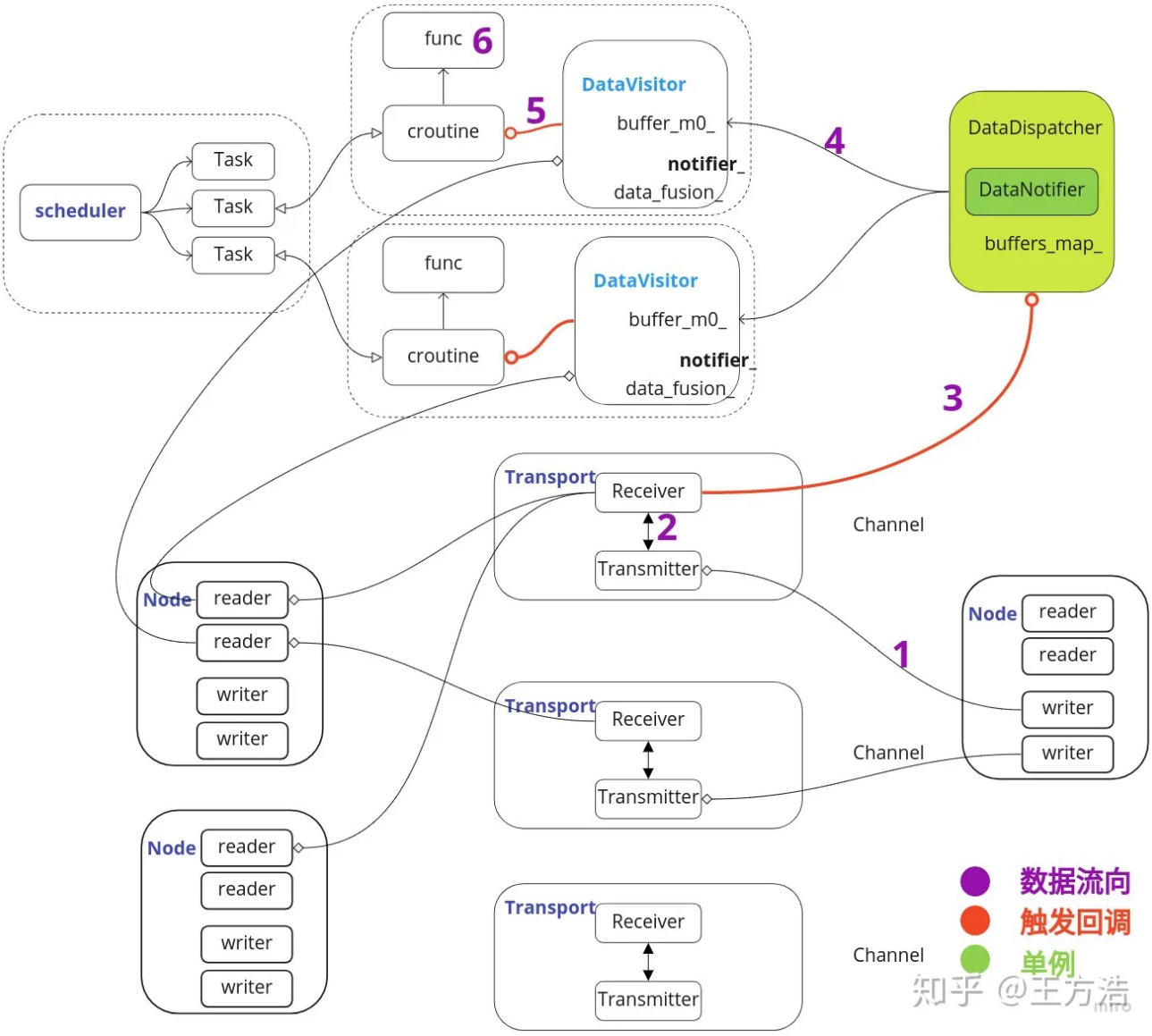
六. 开发流程

参考 Cybertron框架使用示例 *

七. 总结

CyberRT 通过 Node 节点进行通信的底层对接，而 Component 则负责具体业务相关的逻辑;CRoutine 基于消息驱动的基础上，将 Component 中的 Proc 回调作为基础的协程执行单元，然后根据 Sheduler 相应的调度策略进行调度，它保证了多任务的执行顺序；

cyber的数据处理流程



整体数据处理流程图（引自 apollo介绍之Cyber框架(十一)）