

# Apollo决策技术分享

Apollo PnC Team

Yifei Jiang

2019/03/25

无人车作为一个复杂软硬件结合的系统，其安全可靠运行需要车载硬件、传感器集成、感知预测以及控制规划等多个模块的协同配合。

狭义上的决策规划控制部分，包含了无人车行为决策（Behavior Decision）、动作规划（Motion Planning）以及反馈控制（Feedback Control）这三个模块。决策模块在无人驾驶系统中有着非常重要的作用——不仅保障行车的安全，也为轨迹规划提供了指导和限制信息。



- **目的**

- 保障无人车的行车安全并且遵守交规
- 为路径和速度的平滑优化提供限制信息

- **决策的输入**

- Routing信息
- 道路结构，比如当前车道，相邻车道，汇入车道，路口等信息
- 交通信号和标示，比如红绿灯，人行横道，Stop Sign, Keep Clear等
- 障碍物状态信息，比如障碍物类型，大小，和速度
- 障碍物预测信息，比如障碍物未来可能的运动轨迹

- **决策的输出**

- 路径的长度以及左右限制边界
- 路径上的速度限制边界
- 时间上的位置限制边界

决策模块相当于无人驾驶系统的大脑，保障无人车的行车安全，同时也要理解和遵守交通规则。为了实现这样的功能，决策模块为无人车提供了各种的限制信息包括：

1. 路径的长度以及左右边界限制；
2. 路径上的速度限制；
3. 时间上的位置限制。

此外，决策模块几乎利用了所有和无人驾驶相关的环境信息，包括：

1. Routing信息；
2. 道路结构，比如当前车道，相邻车道，汇入车道，路口等信息；
3. 交通信号和标示，比如红绿灯，人行横道，Stop Sign，Keep Clear等；
4. 障碍物状态信息，比如障碍物类型，大小，和速度；
5. 障碍物预测信息，比如障碍物未来可能的运动轨迹。

正是因为决策模块处理了所有上层业务逻辑信息，并输出了抽象的限制信息，我们保证了下游路径和速度优化的抽象性，并完全和上层的地图，感知，预测的解耦。

# 决策的功能模块



参考路径

交规决策

路径决策

速度决策

决策场景的分类和调度

决策的功能模块

在这次的技术分享讨论中，我将为大家介绍Apollo决策是如何设计与实现的。主要分为以下五大功能：**参考路径，交规决策，路径决策，速度决策，决策场景的分类和调度。**

## 参考路径 ( Reference Line )

Baidu 百度 | apollo

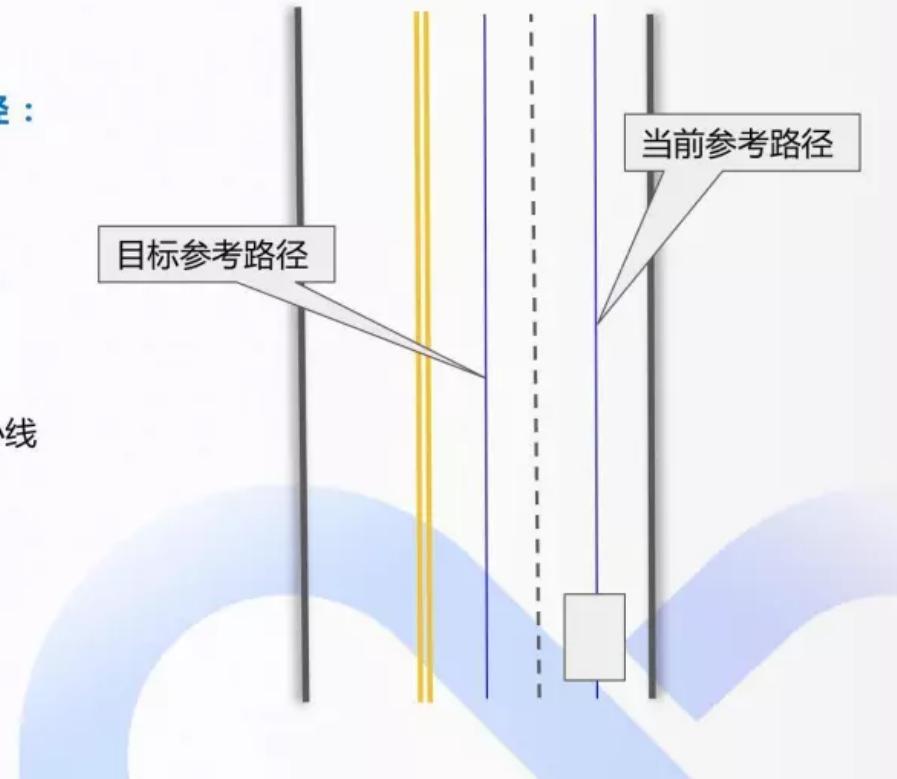
- 没有障碍物情况下的默认行车路径：

- 参考路径需要保证连续和平滑

- 参考路径也用于表达换道的需求：

- 参考路径的一种实现方法：

- 根据Routing找到对应道路中心线
  - 对道路中心线进行平滑



参考路径 (Reference Line) 在Apollo决策中起着非常重要和关键的作用。

- 首先，参考路径是在没有障碍物路况下的默认行车路径。
- 其次，后续的交规决策，路径决策，和速度决策都是基于参考路径或者参考路径下的Frenet Frame完成的。
- 最后，参考路径也用于表达换道的需求，一般换道时会有两条参考路径，分别有不同的优先级。

其中，优先级高的路径表示目标路径，优先级低的路径为当前路径（如上图所示）。参考路径的获取可以有多种方式。目前在Apollo里，[参考路径的计算是根据routing的线路，找到相应的高精地图中的道路中心线，然后进行平滑。](#)

# 交规决策

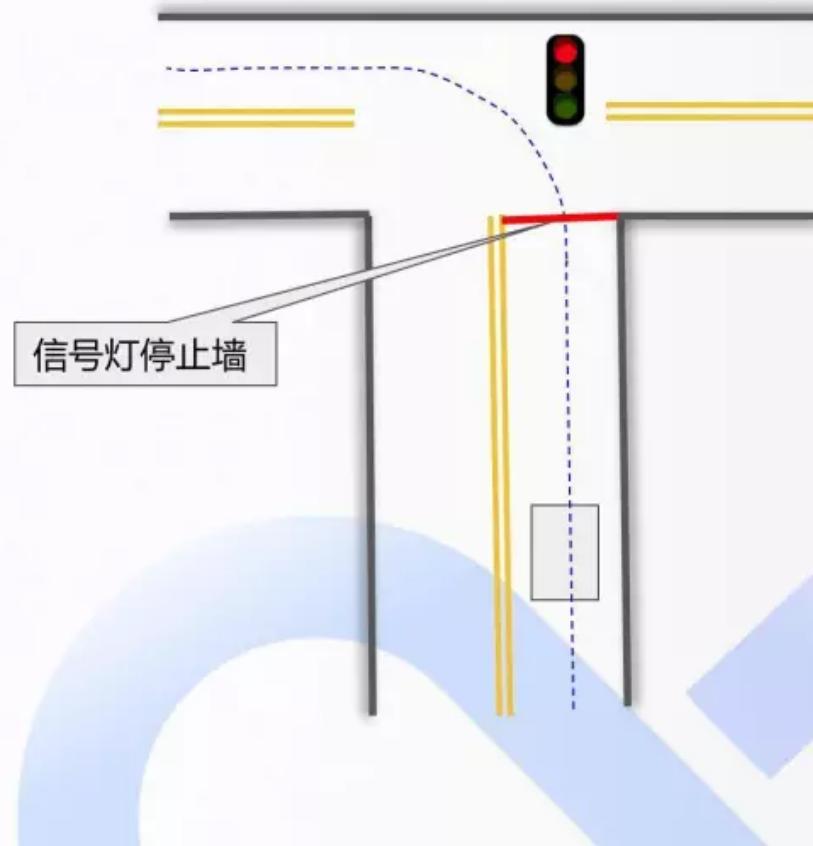
- 处理红绿灯，Stop Sign，人行横道等交通规则

- 输入信息：

- 参考路径
- 高精地图
- 信号灯状态

- 输出：

- 虚拟墙



有了参考路径之后，我们会沿着该路径找到所有在该路径上的交通标示和交通信号灯，并根据一系列的交通规则来决定是否在需要停止在交通标示或者交通信号灯的停止线前。如果需要停止，决策模块会在停止线上生成一个虚拟的墙。虚拟墙会影响后续的路径及速度决策。

如上图例子所示，在无人车的参考路径（蓝色）上有交通灯，根据交通灯的状态：如果是红灯，决策模块会在交通灯的停止线上生成一道虚拟的墙；如果是绿灯，不生成任何虚拟墙。这样我们就完成了绿灯行红灯停的交通规则。其他的交通规则也采取类似的方法，根据规则决定是否在相应的地方放置虚拟墙。

有了根据交通标示和交通信号灯产生的虚拟墙，再加上从感知模块得到的障碍物信息，我们就可以开始进行路径决策。路径决策的过程类似于下图中的决策树。

### 首先我们判断是否需要进行换道（Lane Change）

- 如果我们有多条参考路径并且当前车辆不在优先级最高的参考路径上，则表明需要进行换道，否则为不需要换道。

- 在明确换道的情况下，我们需要判断当前路况是否可以进行安全换道。

- 如果安全，路径决策会产生换道的路径边界，否则产生车道内的路径边界。

- 如果我们确定当前没有换道需求，路径决策会继续确定是否需要借道避让（Side Pass）。

判断条件主要有两个：

1. 当前车道的可行驶宽度不够；
2. 前方障碍物为静止状态且不是由于车流原因静止。比如，路边卸货车辆。

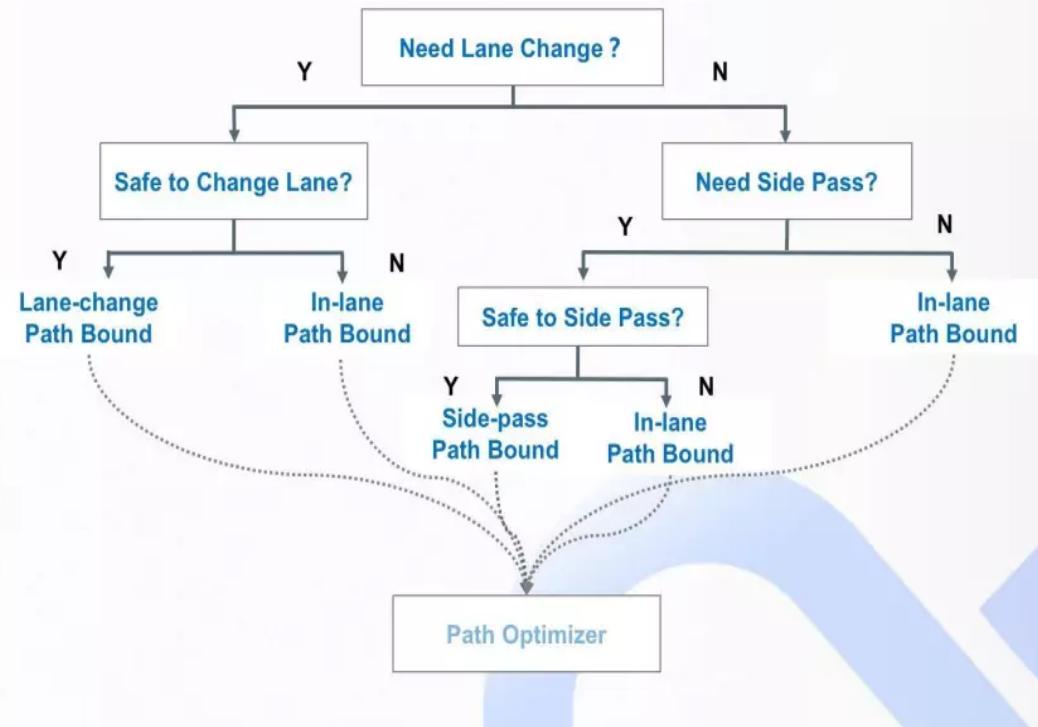
- 如果确定需要借道避让，路径决策会判断是否可以安全借道。

- 如果安全，路径决策会产生街道避让的路径边界，否则产生车道内的路径边界。

- 如果我们确定没有借道避让的需求，路径决策会产生车道内的路径边界。

## 路径决策

Baidu 百度 | apollo

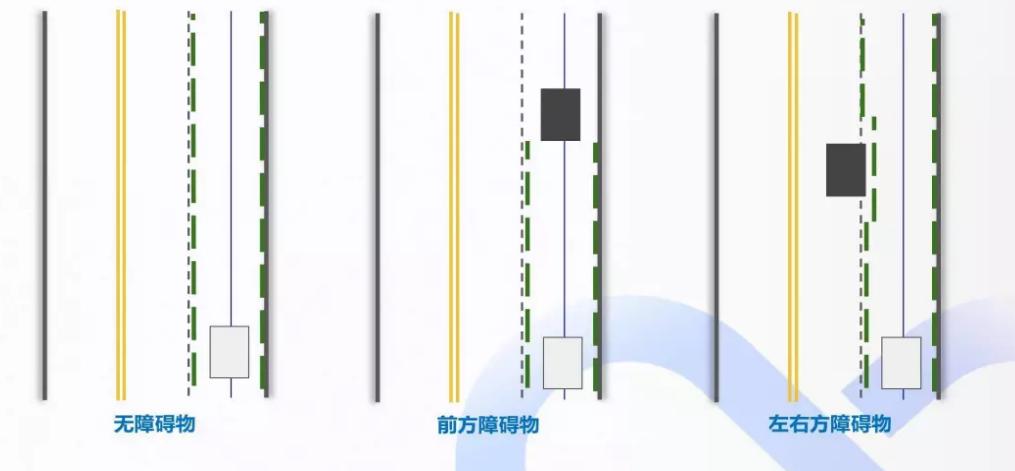


## 车道内径边界

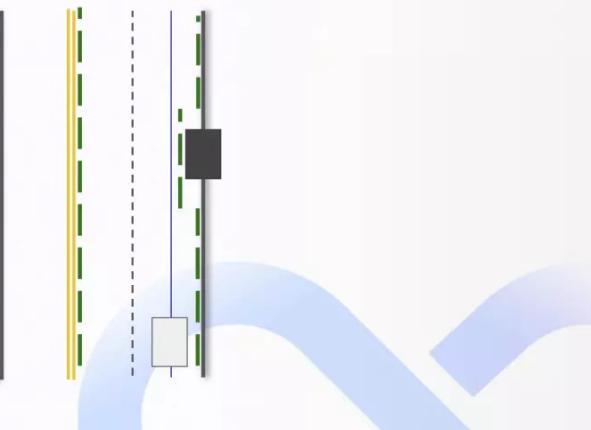
车道内路径边界 的决策有图中所示的三种情况：车道内无障碍物，车道前方有障碍物，车道左右方有障碍物。实际路况中会是其中的一种情况或者多种情况的组合。

- 在无障碍物的情况下（左图），路径边界依据车道边界或者道路边界来生成，并留有一定距离的缓冲（绿色虚线）。
- 在前方有障碍物的情况下（中图），路径边界会被截止在第一个障碍物的后方。
- 如果车道左右方有障碍物（右图），路径边界的产生会依据车道线和障碍物的边界。
- 顺带提一下，图中的蓝色线是车辆的参考路径（Reference Line），路径边界的生成是在参考路径的Frenet Frame下完成的。

## 车道内路径边界 ( In-lane Path Bound )



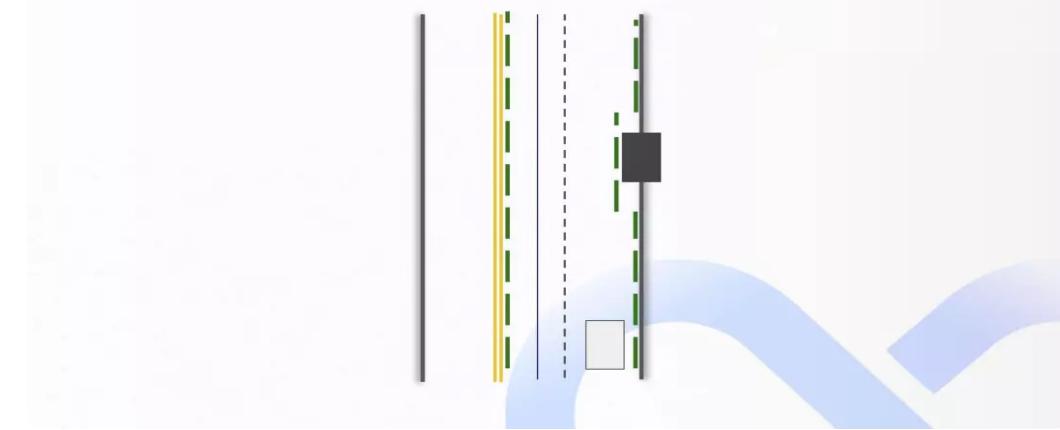
## 借道避让路径边界 ( Side-pass Path Bound )



借道避让路径边界

借道避让的路径边界产生，是在确认可以安全借道之后完成的。而是否可以安全借道的决策，目前是根据一系列的规则来做出的，这个决策也可以依据ST图（后面会为大家介绍）或者数据模型来生成。  
如上图所示，路径的边界是依据本车道和需要借用车道的边界来生成，同时也需要考虑周围的障碍物。

## 换道路径边界 ( Lane-change Path Bound )



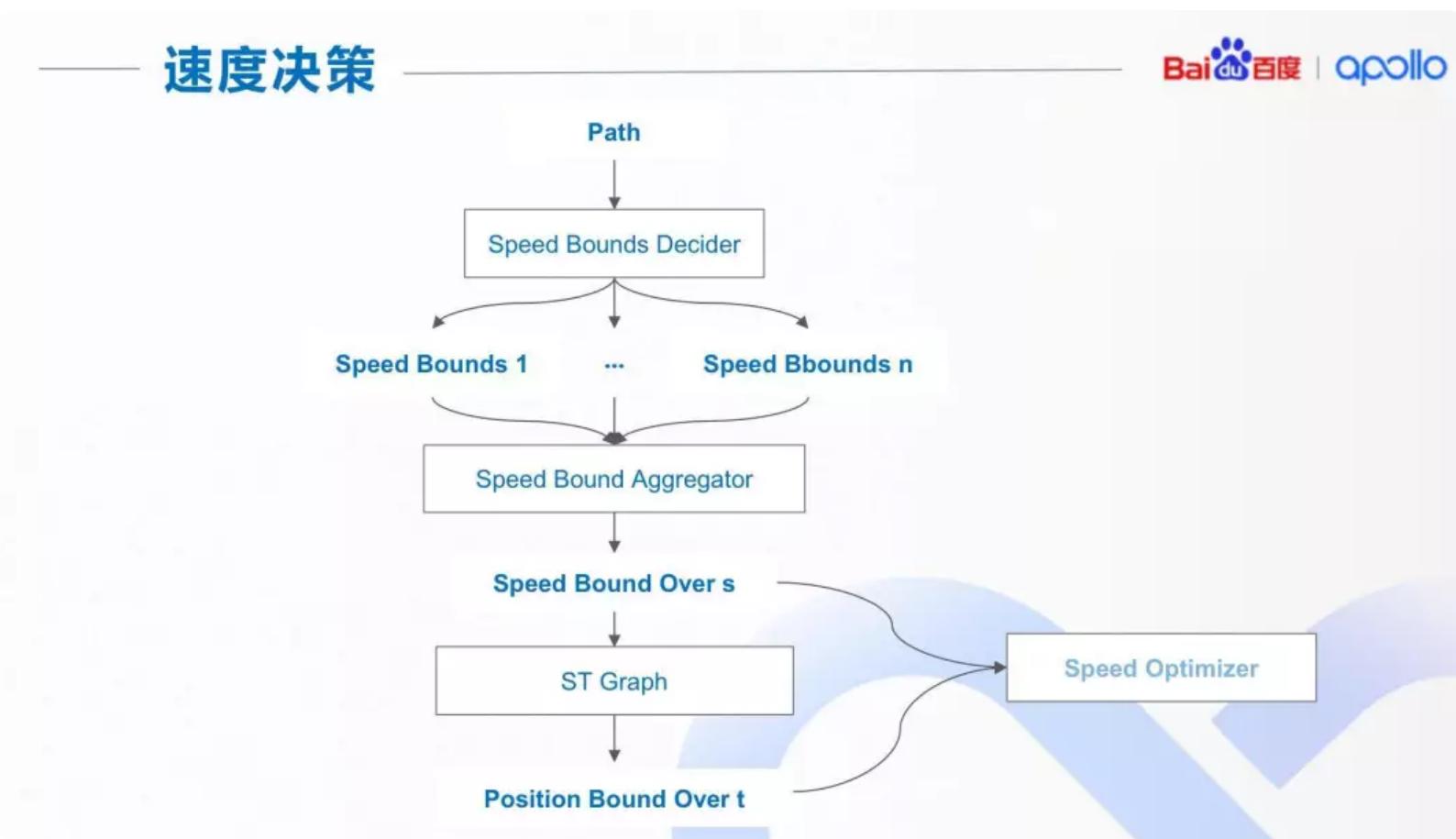
换道路径边界

换道路径边界的产生和借道避让相似（如上图所示），主要的区别是 Reference Line（参考线）在换道时是在目标车道上，而借道避让时是在本车道上。另外需要强调的是，在做路径决策时，我们只考虑静止障碍物。**动态障碍物是在速度规划时考虑**。

有了路径边界后，我们调用路径优化器（Path Optimizer）得到在边界限制内的平滑路径。得到平滑的路径后，我们就可以在路径上进行速度决策。速度决策的流程如下图所示。

### 速度决策

我们首先对一整条路径或者部分路径产生一个或者多个速度限制边界（Path Bounds），然后对多个速度边界进行集成，得到最终的路径上的速度限制边界（Speed Bound Over s）。



得到速度限制之后，我们在利用ST图来得到时间上的位置限制边界（Position Bound Over t）。最后我们把速度边界和位置边界传给速度优化器（Speed Optimizer）得到平滑的速度规划。

我们在很多情况下，出于行车安全或者遵守交规的原因，需要对车辆的速度进行限制。比如，当路径旁边有行人时，我们要减速慢行；当我们要借道避让时，也要减速慢行。这样的速度限制可能是对整条路径，比如道路限速，也有可能是对路径中的一小段，比如减速带。

如右上图所示，沿路径（ $s$ ）有三种不同的速度限制：道路限速（黄色），减速带（红色），和行人（绿色）。为了得到整条路径的综合限速，我们把这几种限速集成到一起，如下图所示。

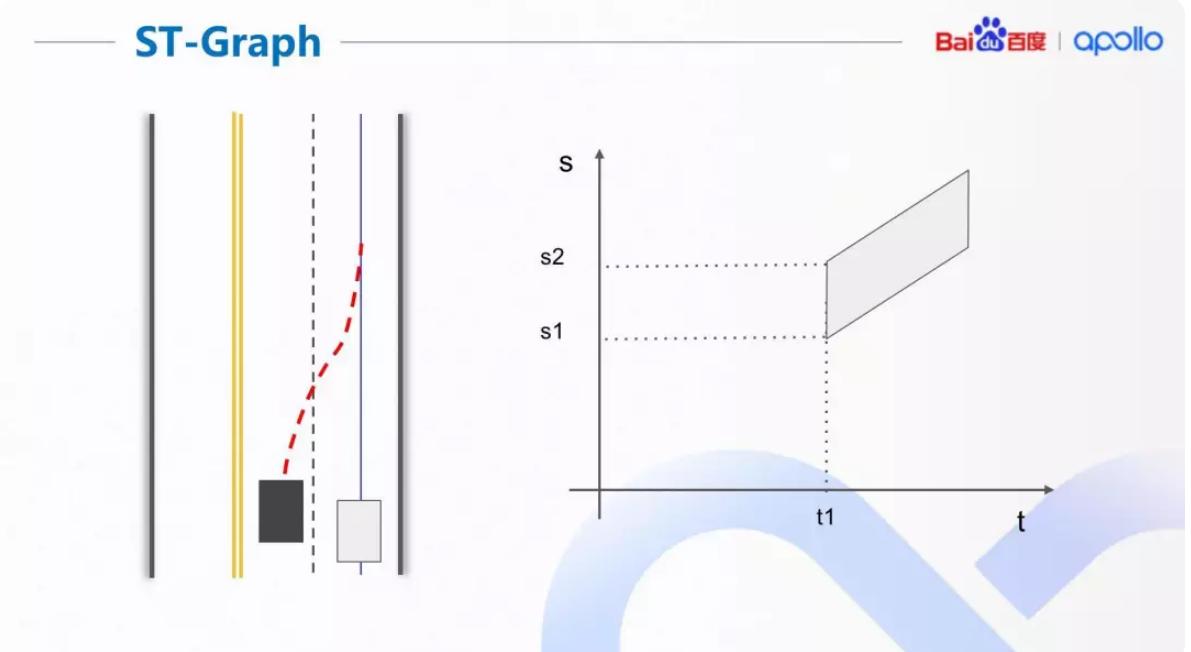
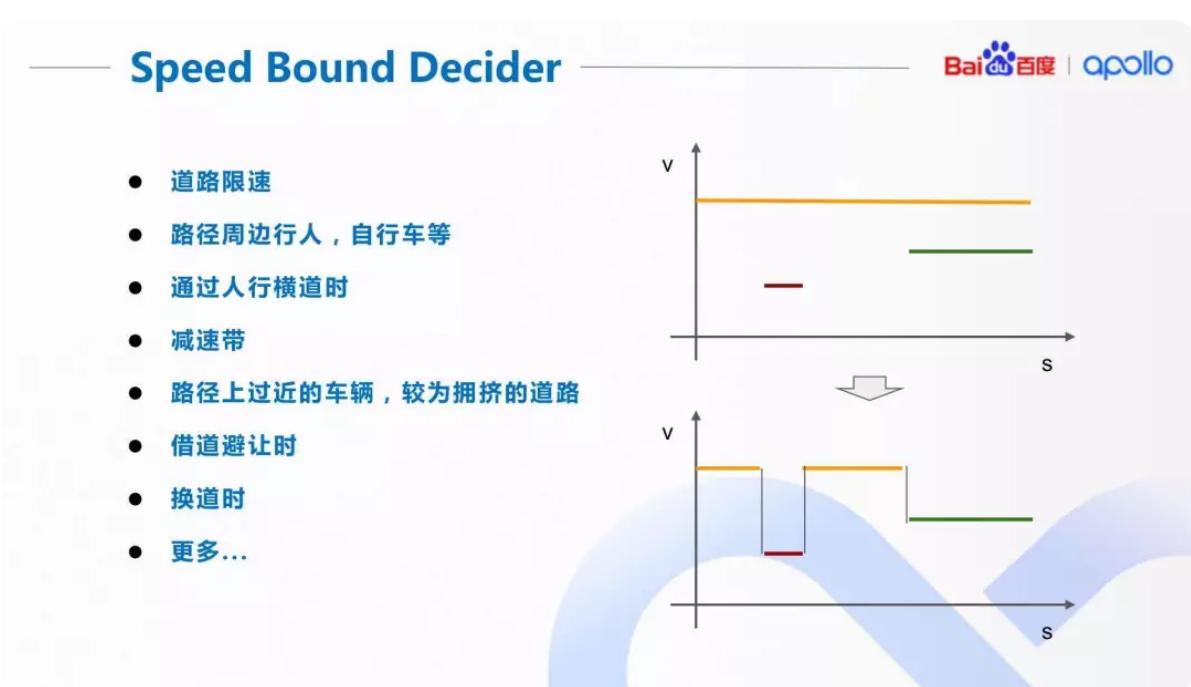
### 综合限速

得到了路径上的速度边界后，我们就可以利用ST图来求解时间上的位置边界。上图是一个简单的ST图的例子，我们用这个例子来简单解释我们为什么需要ST图以及如何从ST图上得到时间上的位置边界。

左图是一个简单的驾驶场景，右侧灰色方框代表自动驾驶主车，蓝线是主车的路径；左侧黑色方框代表障碍车，红线是障碍车的预测行驶轨迹。

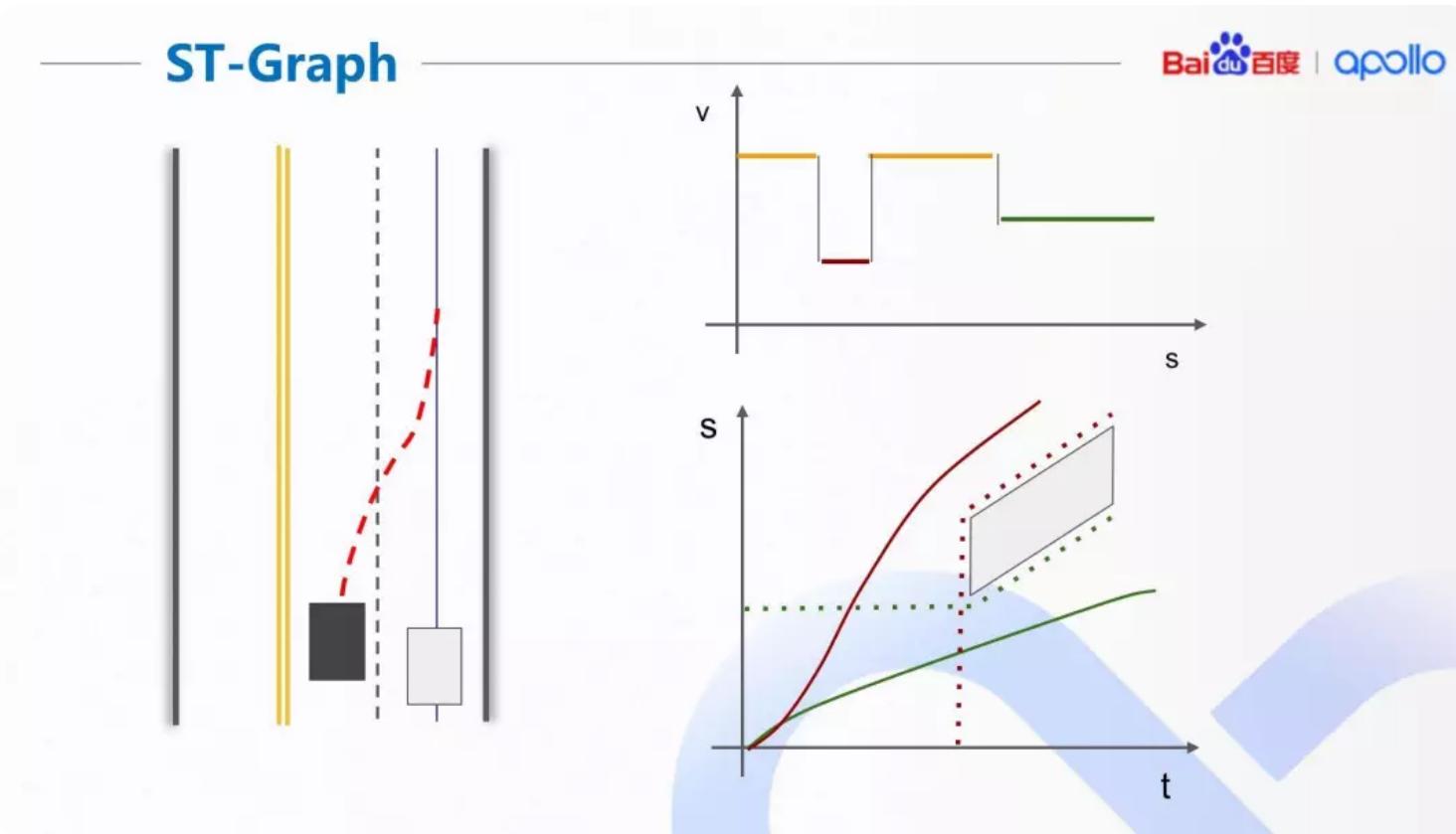
把障碍车的预测轨迹和主车的路径的交汇关系在ST图中表示出来，就如下图所示。

- $t_1$ 为障碍车预测轨迹和主车路径的交汇时间；
- $s_1, s_2$ 为交汇时障碍车在主车路径上的位置；
- $s_1$ 代表车尾位置， $s_2$ 代表车头位置。



## ST图

在ST图中，我们的目标是找到一条不和障碍物碰撞的曲线。同时，曲线还需要满足我们之前计算的路径上的速度限制，即曲线的斜率不能超过 $v$ 的限制边界（右上图）。



找到最优的一条曲线后，我们根据曲线计算时间上的位置限制边界。

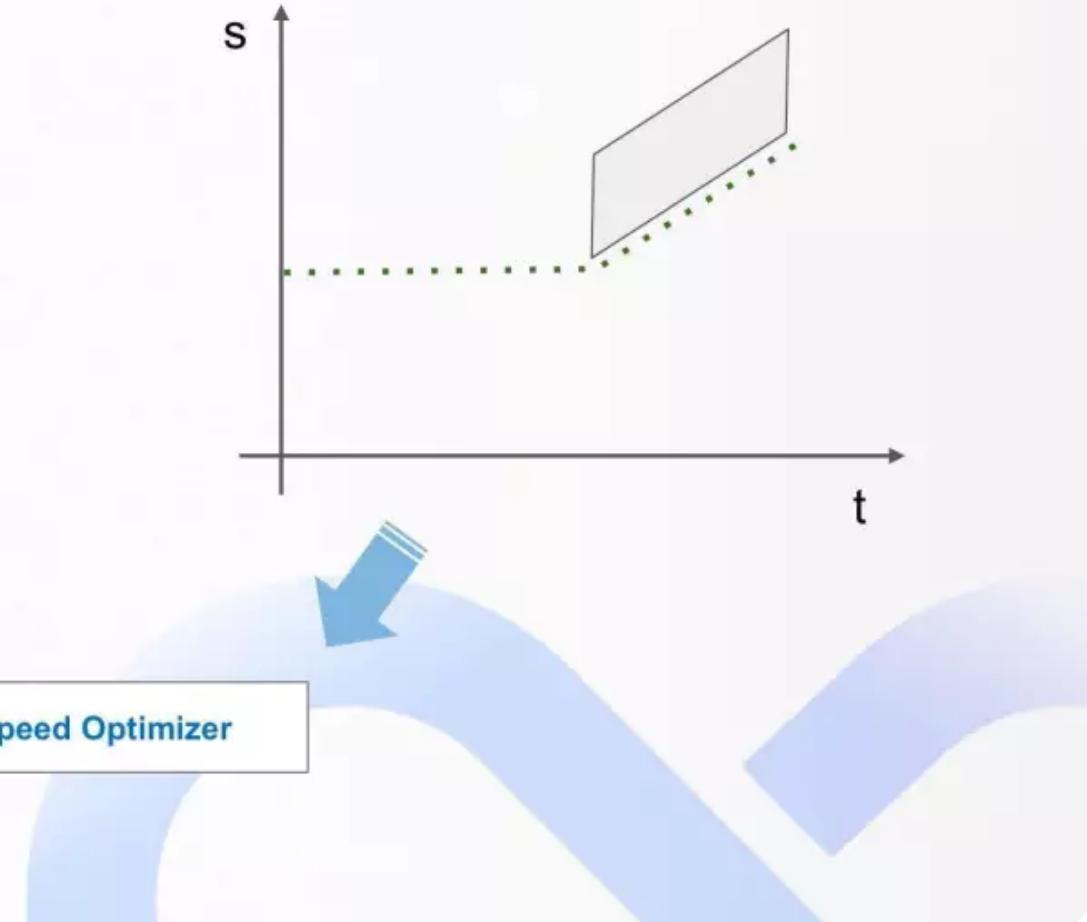
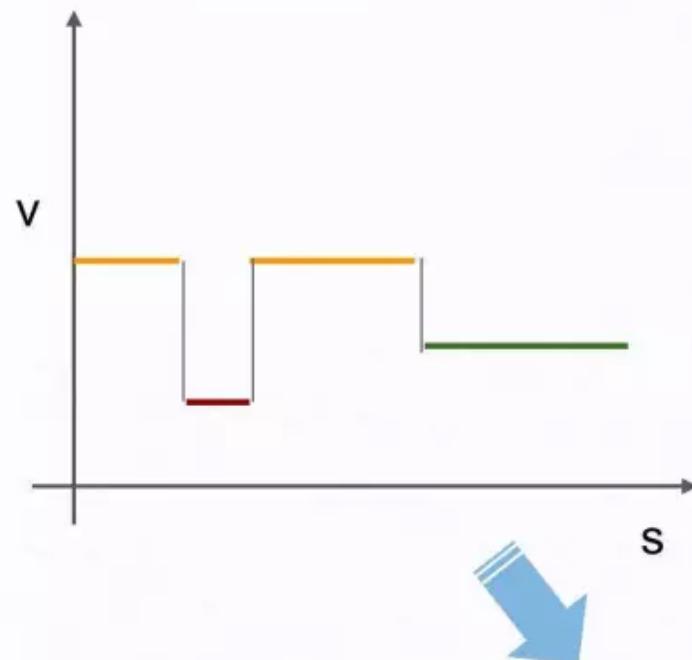
例如，如果我们找到红色的曲线为最优曲线，时间上的位置限制就为红虚线段。

在x, y平面中，就表现为主车在障碍车换道前进行超车。反之，绿色的曲线和绿色虚线段表示主车在障碍车换道后，进行跟随。

有了路径上的速度限制，及时间上的位置限制之后，我们就可以把这两个决策传递给速度优化器得到平滑的速度规划，即在路径上的每个点的时间。生成速度规划后，我们就可以结合路径和速度生成最终的Planning的轨迹。

## 速度优化

Bai<sup>度</sup> | apollo



以上就是Apollo决策的一些基本设计和实现。

下面为大家介绍一下Apollo里关于「决策场景」的设计。关于“场景”的概念是在Apollo3.5中首次提出，一个场景既可以是地图中的一个静态路段，比如十字路口；也可以是无人车想要完成的一个动态目标，比如借道避让（Side Pass）。

## 什么是场景



依据场景来做决策和规划有以下两个优点：

1. 场景之间互不干扰，有利于并行开发和独立调参。
2. 在一个场景里我们可以实现一系列的有时序或者依赖关系的复杂任务。这样的场景架构也有利于开发者贡献自己的特有场景，并不影响其他开发者需要的功能。

- 场景可以是地图中有一定特征的路段，比如路口，也可以是无人车想要完成的一系列复杂动作，比如借道避让。
- 场景的优点：
  - 场景之间互不干扰，可以进行并行开发和调参
  - 在一个场景里可以实现一系列的有时序或者依赖关系的复杂任务
- 场景的缺点/难点：
  - 每个场景都需要处理一些基本路况，可能会带来代码的冗余
  - 场景之间无覆盖划分非常困难
  - 需要额外的逻辑来进行场景识别以及处理场景之间的转换

同时场景架构也带来了一些难点：

首先，场景的通用部分会带来代码的冗余，不便于代码的升级维护；  
其次，场景的划分，尤其是保证场景之间没有功能上的覆盖也很困难；  
最后，有了多个场景之后，需要额外的逻辑来进行场景识别以及处理场景之间的转换。

## 场景的分类

场景的划分其实没有特别严格的规定，同时这也取决于自动驾驶的应用场景，比如送货小车和高速卡车在场景的划分上肯定不太一样。上图中，我们给出了Apollo场景分类的一个例子，来尽量保证每个场景之间相对独立，这样的分类设计仅供大家参考。

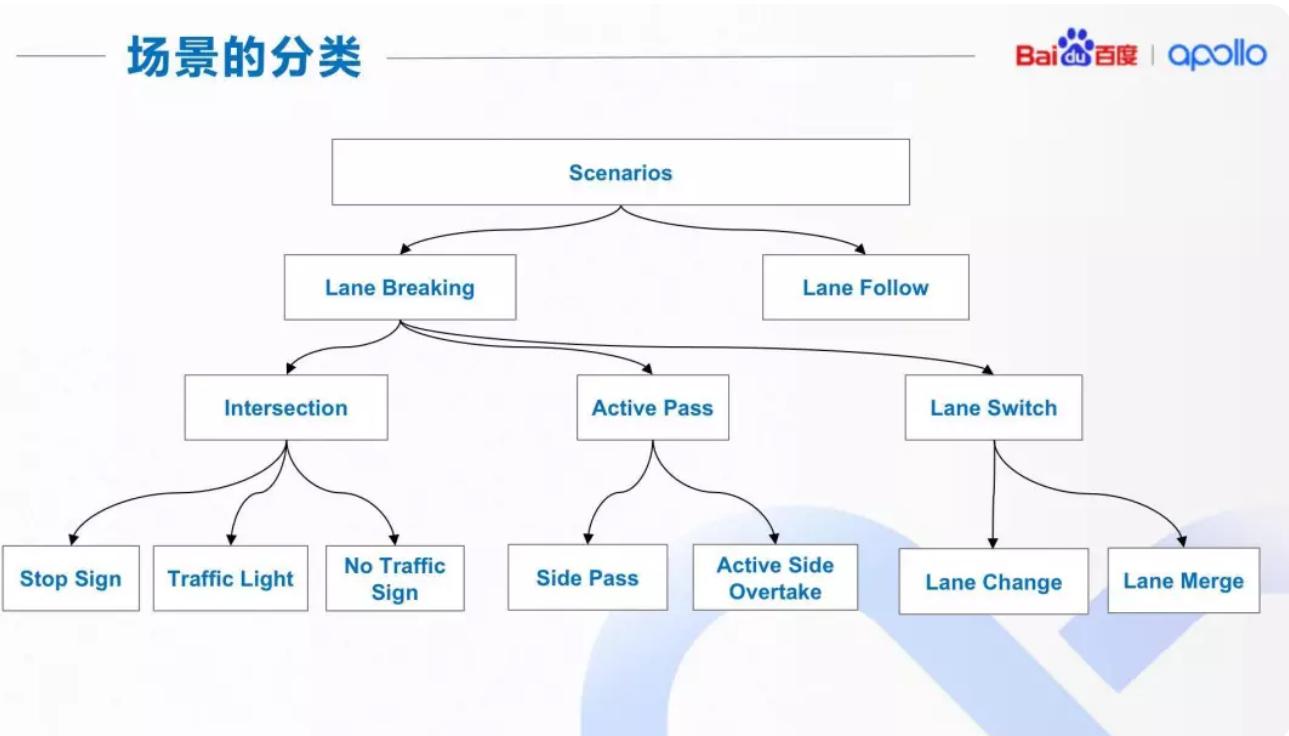
我们把场景分为两大类，Lane Follow 和 Lane Breaking：

- Lane Follow场景下主车沿一条车道驾驶，该车道前方一定距离内没有其他车道与其交汇或者穿越；并且主车也没有切换到其他车道的意图。

- 非Lane follow的其他场景被分类为Lane Breaking。

在Lane Breaking 下，我们又细分为三个小类：

- Intersection是所有的路口场景；
- Active Pass包括了所有临时借道的场景；
- Lane Switch包括了所有换道的场景（包括车道合并）。



有了场景的分类之后，我们就可以对场景进行识别和转换。对于选择哪个场景，我们采用了两层的识别机制。

## 场景的识别和转换



- **识别方法**

- 每一个场景自己会识别自己是否符合当前行车路况
- 场景管理器根据每个场景的返回值，再进行综合判断
- 最终选择一个场景，来负责交规、路径、速度的决策，以及路径、速度的优化

- **场景退出**

- 每个场景自己判断是否当前的路况处在该场景中，来决定是否退出
- 场景内出现错误，场景自己也会决定是否退出
- 一旦进入一个场景，该场景会有较高优先级完成
- 场景完成后会主动退出

首先，每一个场景自己会根据当前的环境信息，确定是否属于自己的场景，并返回该信息给场景管理器，场景管理器再统一进行选择，以保证场景的选择最优。

场景的退出也由每个场景自己决定，比如场景运行完成或者内部出错。一旦进入一个场景，该场景会有较高优先级来完成。

## 借道避让场景

接下来，我们用借道避让场景的一个实现，来说明场景是如何实现的。

在这个场景中，我们有6个Stage（阶段），每个Stage完成借道避让的一个步骤，类似于有限状态机中的一个状态。

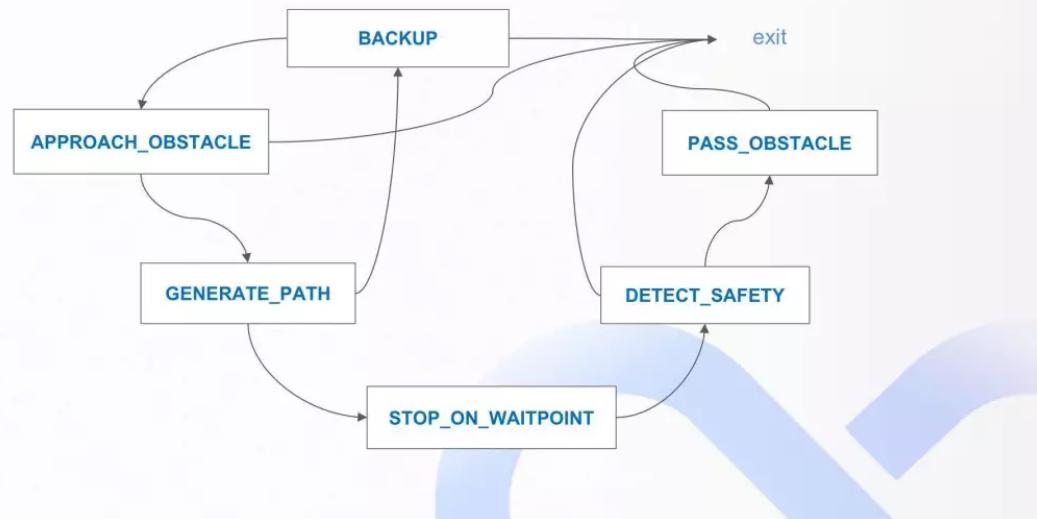
主要步骤/状态有一定的时序依赖关系，如果在一个Stage（阶段）中发现环境变化了，或者出现错误，会出现Stage之间的跳转或者退出该场景。

## 场景的Stage

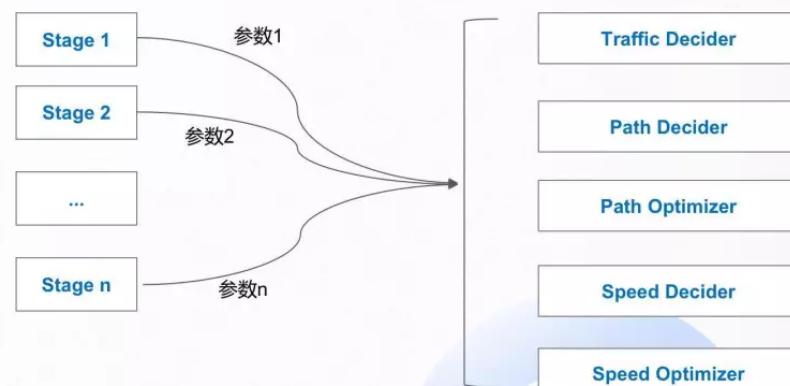
在每一个Stage（阶段）中，都要实现上图中的功能，包括交规决策、路径决策、路径优化、速度决策、速度优化。

我们把每个功能定义为一个或者几个基本的Task（任务），每个Stage（阶段）或者直接调用（使用默认参数），或者修改参数，或者修改输入值。这样的实现可以极大的提高场景之间的代码复用。

## 借道避让场景 (Side Pass)



## 场景的Stage

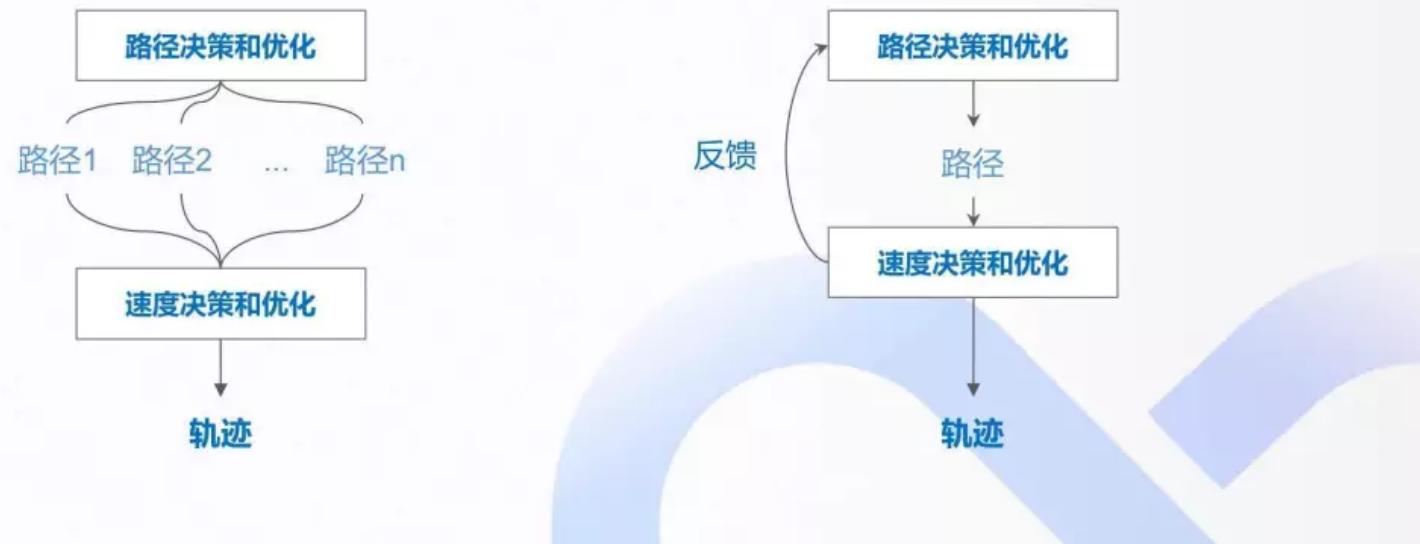


在上述的Apollo决策的实现中，我们使用了路径和速度分离的方法。这样的实现极大的简化了决策和优化问题，但是同时也使得最终轨迹不最优的。

## 讨论

Baidu 百度 | apollo

- 路径和速度分离的决策方法，使得决策和后续的优化问题得到了极大地简化，并且在大部分情况下工作的很好
- 潜在问题：最终轨迹不能保证最优



我们同时也在探索一些方法，使得路径和速度的决策和优化能够关联起来。

一种可能的方法（如左图所示）是路径决策时输出几种不同的路径，在速度决策和优化时，同时考虑多条路径，并选出最优轨迹。

另外一种思路（如右图所示）是速度和路径之间形成反馈环，如果速度决策和优化时发现路径决策不理想，会反馈至路径决策，重新进行路径决策和优化。