PNC架构

目录

- 关键函数
- **RightTurnExecutor**

整体逻辑

整个自动驾驶任务划分为不同的component，定义在/onboard/component/目录下，通过channel接收以及发送各个数据，使用DAG来配置消息的流通

决策规划对应的component为executor_component，通过内部的modules来完成决策规划，也有一些lib来完成一些辅助任务

module包含：

world_builder：负责自动驾驶场景的构建。

new_central_deicder：负责场景的切换，决定当前帧该使用的executor。

executor_mananger：负责管理所有的execotur，返回executor的实例，并通过executor来完成当前帧的决策规划

message_exporter：负责最后消息的整理发送

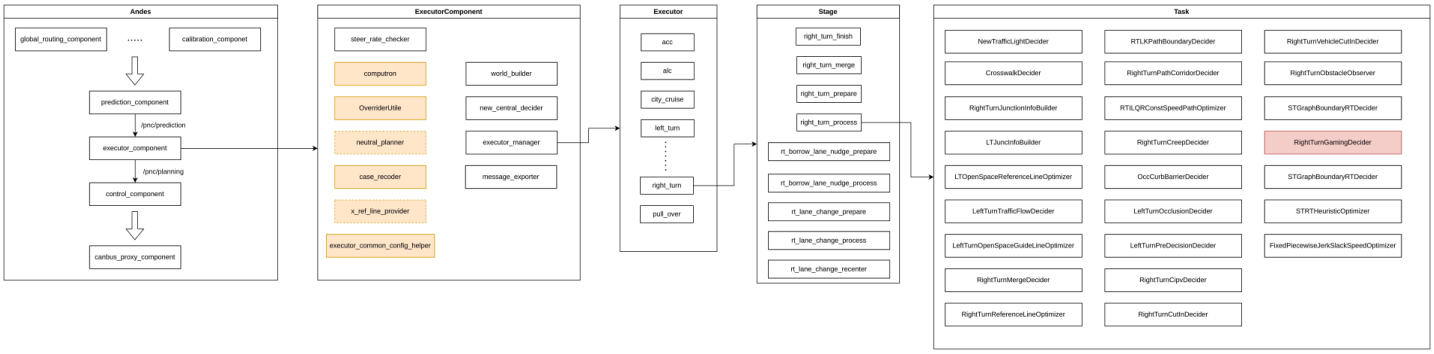
executor为对应各个场景的执行器(city_cruise、right_turn、pull_over)等，executor将场景分为不同的stage(merge_in、lane_change等)，通过状态转移机来进行stage的切换，通过stage来执行

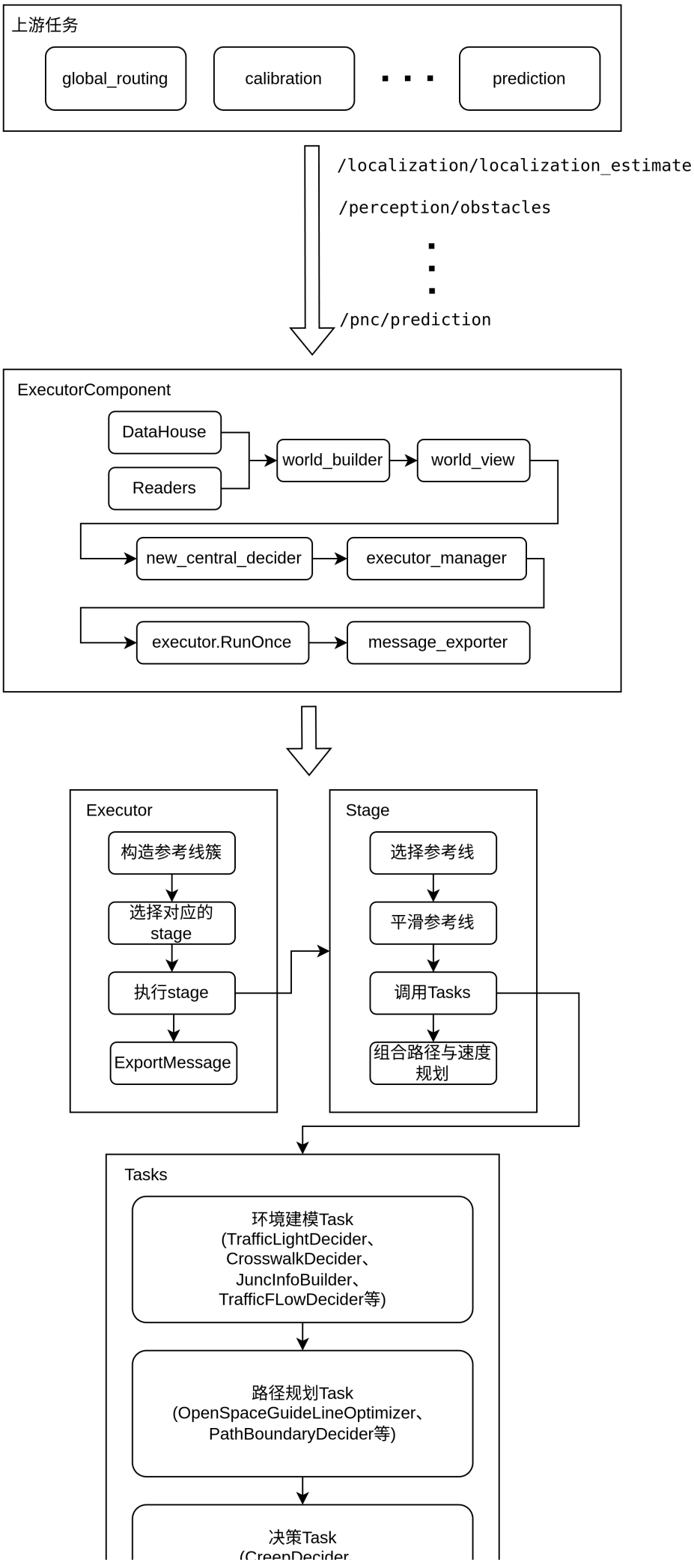
stage通过按顺序执行预定义好的task来工作，task分为decier以及optimizer，decider负责产生决策，而optimizer则完成轨迹的规划

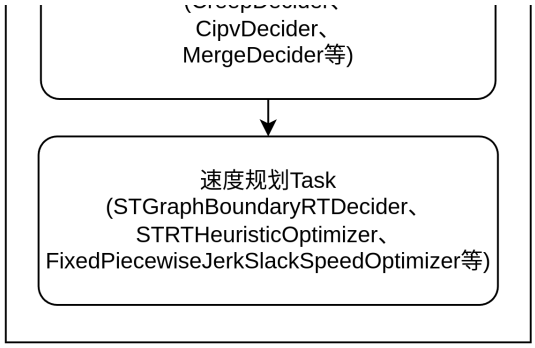
proto定义在asd/anp-commen/proto下

FLAG在asd/anp-common/common下

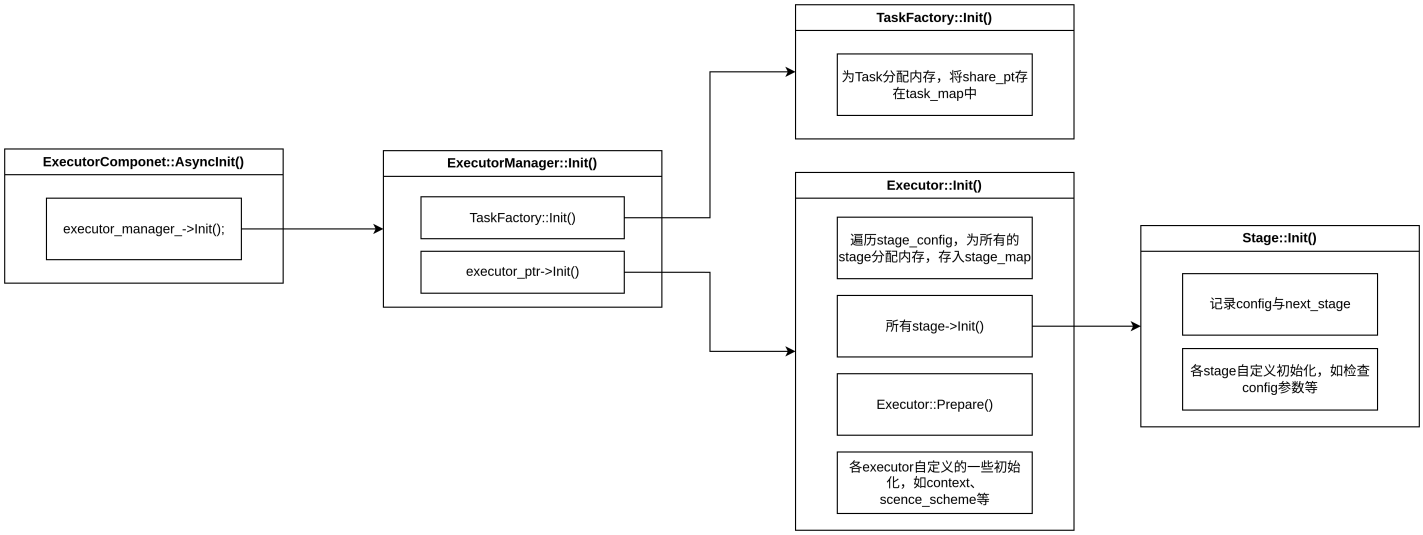
proto配置在./onboard/conf下



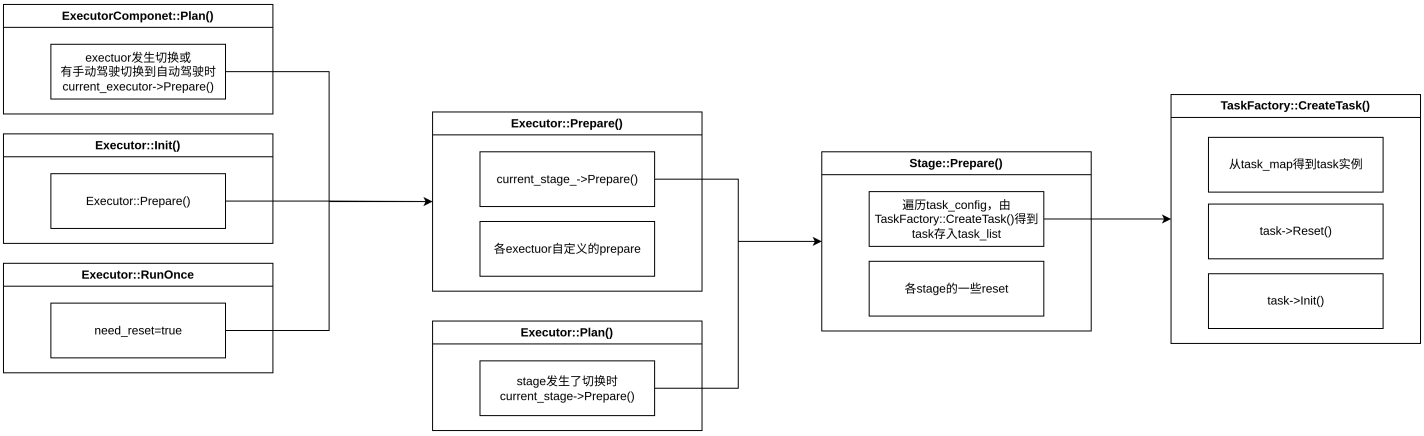

上游任务
global_routing
calibration
prediction
/localization/localization_estimate
/perception/obstacles
/pnc/prediction
ExecutorComponent
DataHouse
Readers
world_builder
world_view
new_central_decider
executor_manager
executor.RunOnce
message_exporter
Executor
构造参考线簇
选择对应的stage
执行stage
ExportMessage
Stage
选择参考线
平滑参考线
调用Tasks
组合路径与速度规划
Tasks
环境建模Task
(TrafficLightDecider、
CrosswalkDecider、
JuncInfoBuilder、
TrafficFLowDecider等)
路径规划Task
(OpenSpaceGuideLineOptimizer、
PathBoundaryDecider等)
决策Task



Init流程

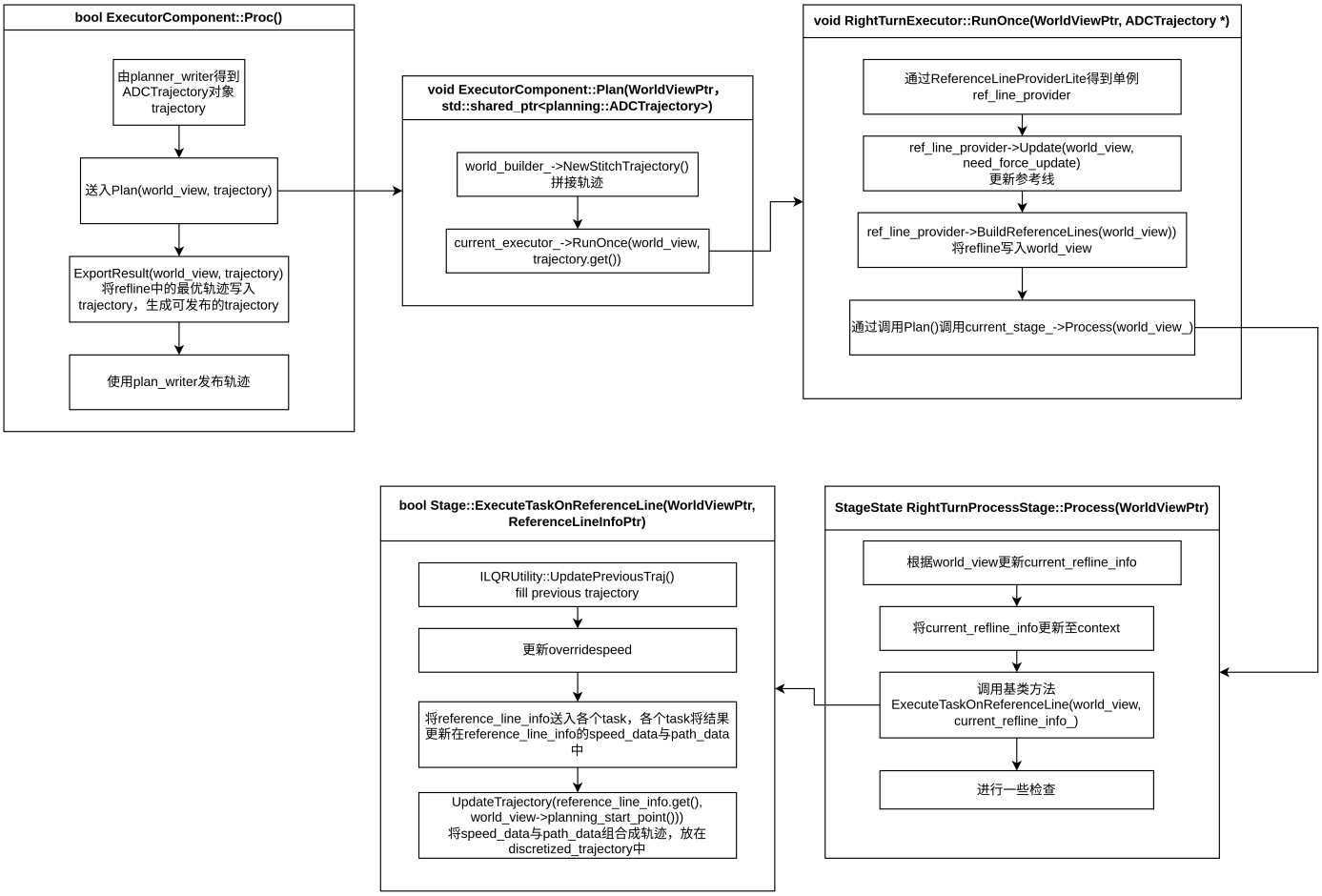


Prepare流程



轨迹生成流程

通过reference_line_info完成规划轨迹的传递，在每个stage的最后将所规划的速度与路径组合成轨迹，通过messege_export完成最终轨迹的确定与待发布轨迹的生成

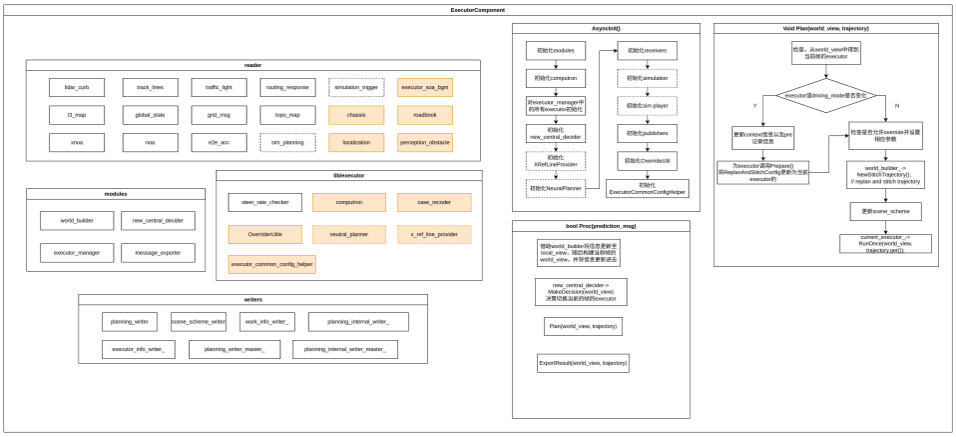


executor_component

声明：./onboard/executor/executor_component.h

component是cybertron的通信节点，通过channel接收以及发送各个数据，通过DAG配置来接收prediction的消息，其他消息是通过DataHouse或Reader来接收的

executor_component是PNC模块的入口



关键成员变量

1. world_builder：负责建立world_view，里面存放了自动加速所需要的全部场景信息，通过调用UpdateXXX将信息更新至world_view

2. new_central_decider：多层有限状态机，根据world_view来对负责executor的的决策切换
3. executor_manager：负责实例化各个executor并返回对应的executor

关键函数

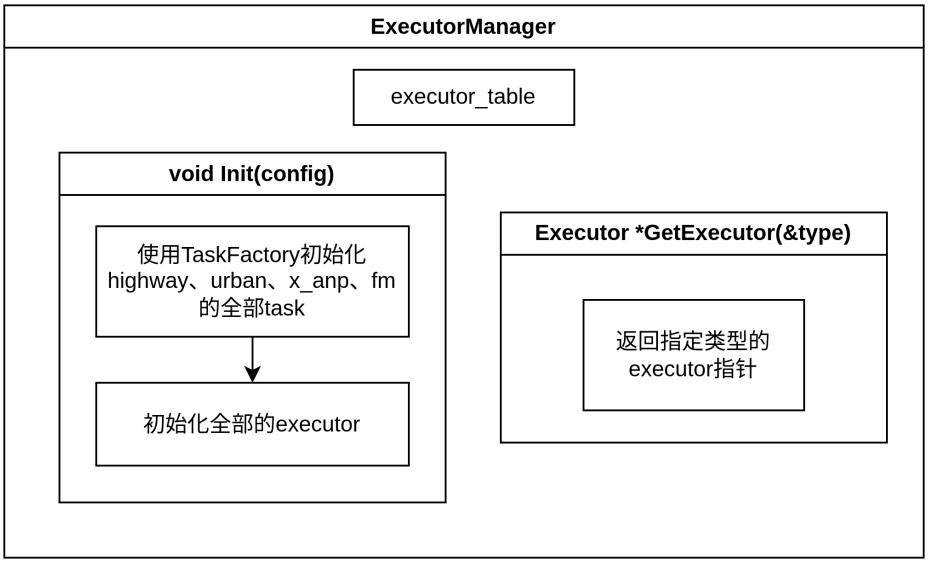
1. AsyncInit()：读取config，初始化成员变量，设置消息接受发送节点
2. Proc()：执行主函数，更新world_view -> new_central_decider_->MakeDecision(world_view)完成executor决策切换 -> Plan()规划 -> 检测发布
3. Plan()：规划函数，获取决策后的executor -> 检查驾驶模式，选择对应的轨迹拼接器 -> 更新scene_scheme -> currennt_executor -> RunOnce()

executor_manager

声明：./modules/executor/executor.h

负责实例化所有的executor，并调用其init函数进行初始化，将所有的executor注册在字典中，通过get方法得到对应的executor

##在executor初始化的时候也会一并将下面的statge与task都初始化了



关键成员变量

1. executor_table_：字典存储对应的executor实例(shared_ptr)

关键函数

1. Init()：初始化task，注册全部的executor
2. GetExecutor()：返回executor指针

new_central_decider

ref： 状态机状态流转细节梳理 CentralDecider 设计文档

声明 ./modules/decider/new_central_decider.h

为3层状态机，

- 第一层为domain的切换，在MIX_AREA、HIGHWAY、URBAN之间切换，根据地图或DV 设置进行切换
- 第二层为func_state，在FSTAT_MANNUAL、FSTAT_ACC、FSTAT_AP、FSTAT_XANP、FSTAT_ANP之间切换，根据设置的config切换，会调用下层的fsm进行切换
- 第三层为state_types，在executor间切换，在func_state下，在根据设置的config切换

各个状态的切换条件称为原子能力

关键成员变量

1. fsm：状态机

函数	工作域	备注
Prepare	首次进入时执行一次	指定状态，并调用对应状态的 Prepare
Entry	每帧都会执行	响应 Event，并调用当前状态的 Entry
Exit	退出时执行一次	回收资源，回收变量

2. cur_domain：当前所在的区域，在MIX_AREA、HIGHWAY、URBAN之间切换
3. cur_func_state：fsm的状态，在FSTAT_MANNUAL、FSTAT_ACC、FSTAT_AP、FSTAT_XANP、FSTAT_ANP之间切换
4. cur_state_types：二维的向量，第一维存放NEST_AUTO_AP、NEST_AUTO_XANP、NEST_AUTO_ANP、NEST_AUTO_ACC，第二维存放对应的execuor

函数	工作域	备注
Prepare	进入 State 时执行一次	初始的一些检测，和基本变量的设置。同时调用子
Entry	每帧都会执行	正常逻辑，同时调用子状态机的 Entry
Exit	退出 State 时执行一次	回收资源，回收变量，同时调用子状态机 Exit
TriggerEvent	将 Event 传递给父状态机的 FIFO	可在 Entry 中调用，实现状态跳转

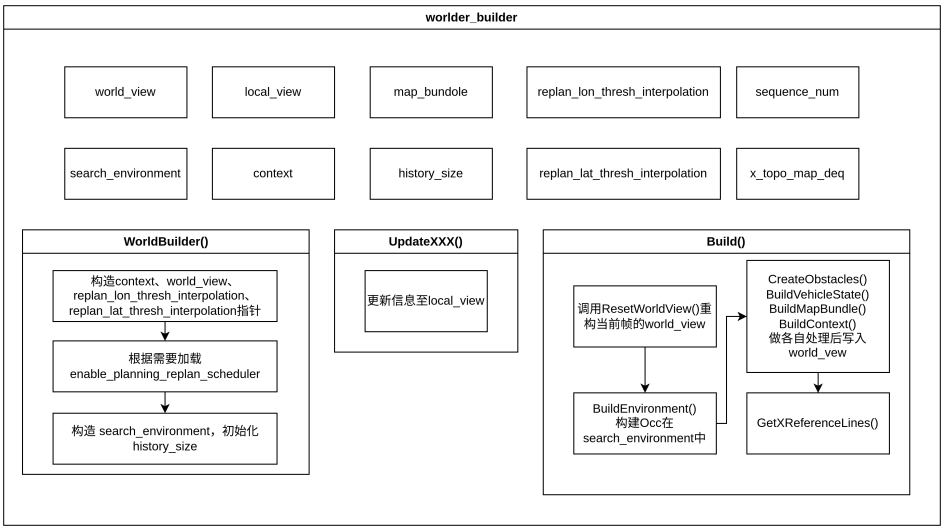
关键函数

1. void NewCentralDecider::MakeDecision(WorldViewPtr world_view)：决策个当前的exectuor，将决策结果写入world_view的higt_level_decison中

world_builder

声明：./modules/executor/world_builder.h

world_view更新的接口，在executor_component中调用各种UpdateXXX函数对world_view进行更新



关键成员变量

1. local_view：存储当前的信息，Update的数据首先先存在local_view中，再更新至world_view
2. search_environment：存储一些环境信息
3. context_：存储executor所要的额外上下文信息
4. world_view_：描述自动驾驶场景的信息

关键函数

1. 构造函数：实例化context与world_view，加载参数，reset search_environment
2. UpdateXXX()：更新信息至local_view
3. UpdatePredicion()：更新prediction_obstacles至local_view，随后调用Build()更新local_view至world_view
4. Build()：构建新的world_view，将local_view的信息写入world_view(Occ、obstacles、VehicleState、MapBundle、Context)，world_view内有一个prev_world_view将所有world_view链接起来，通过history_size_控制链表长度
5. NewStitchTrajectory()：帧间轨迹平滑

local_view

声明 ./lib/executor/common/world_view/local_view.h

临时存储当前帧的环境信息，后续更新至world_view，实例存储在world_builder中，会不断的复制到world_view中

关键成员变量

```
// external environment
```

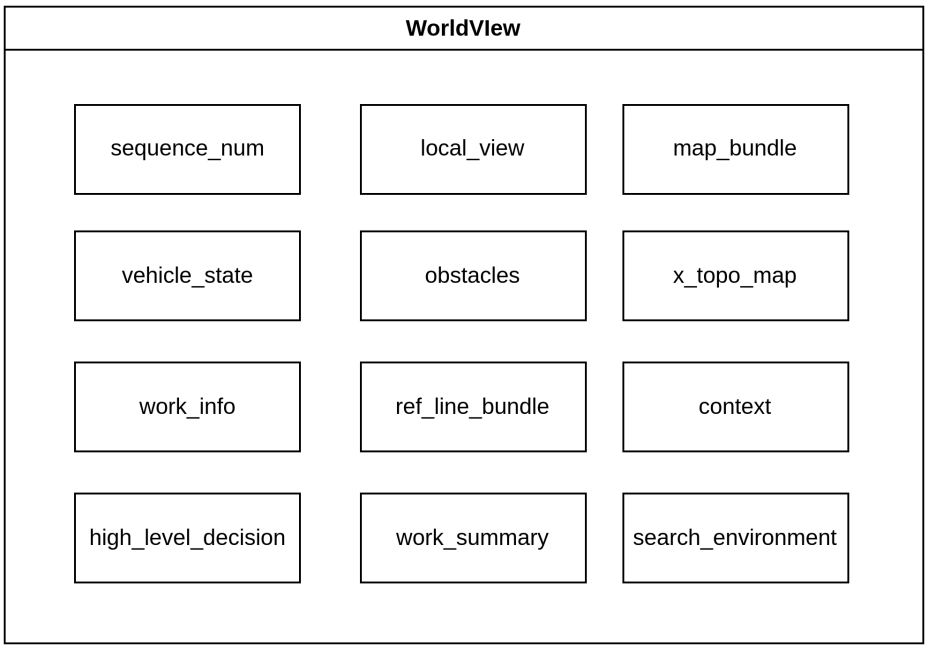
```
1.  std::shared_ptr<const roadbook::RoadTopo> roadbook;
2.  std::shared_ptr<const prediction::PredictionObstacles> prediction_obstacles;
3.  std::shared_ptr<const perception::PerceptionObstacles> perception_obstacles;
4.  std::shared_ptr<const perception::PerceptionLandmarks> perception_lidar_curb;
5.  std::shared_ptr<const perception::PerceptionLandmarks> perception_track_lanes;
6.  std::shared_ptr<const common::traffic_light::TrafficLightDetection> traffic_light_detection;
7.  std::shared_ptr<const perception::topo_map::TopoMap> topo_map;
8.  std::shared_ptr<const perception::PerceptionE2EAcc> perception_e2e_acc;
// internal environment
9.  std::shared_ptr<const chassis::Chassis> chassis;
10. std::shared_ptr<const localization::LocalizationEstimate> localization;
// navigation
11. std::shared_ptr<const noa::NoaMsg> noa;
12. std::shared_ptr<const noa::L3MapMsg> l3_map;
13. std::shared_ptr<const new_router::RoutingResponse> routing_response;
14. std::shared_ptr<const xnoa::XNoaMsg> xnoa;
// status machine
15. std::shared_ptr<const status_machine::GlobalState> global_state;
// others
16. std::shared_ptr<const apollo::vehicle_info::bgmstsfdb> bgm_info;  ???
// grid map
17. std::shared_ptr<const apollo::perception::PerceptionGridMap> grid_msg;
```

world_view

声明 lib/executor/common/world_view/world_view.h

存储当前帧的自动驾驶信息，每一帧都会重新构建，并使用指针指向上一帧的world_view，通过world_builder控制指针长度

通过友元以及mutable函数进行成员的更新



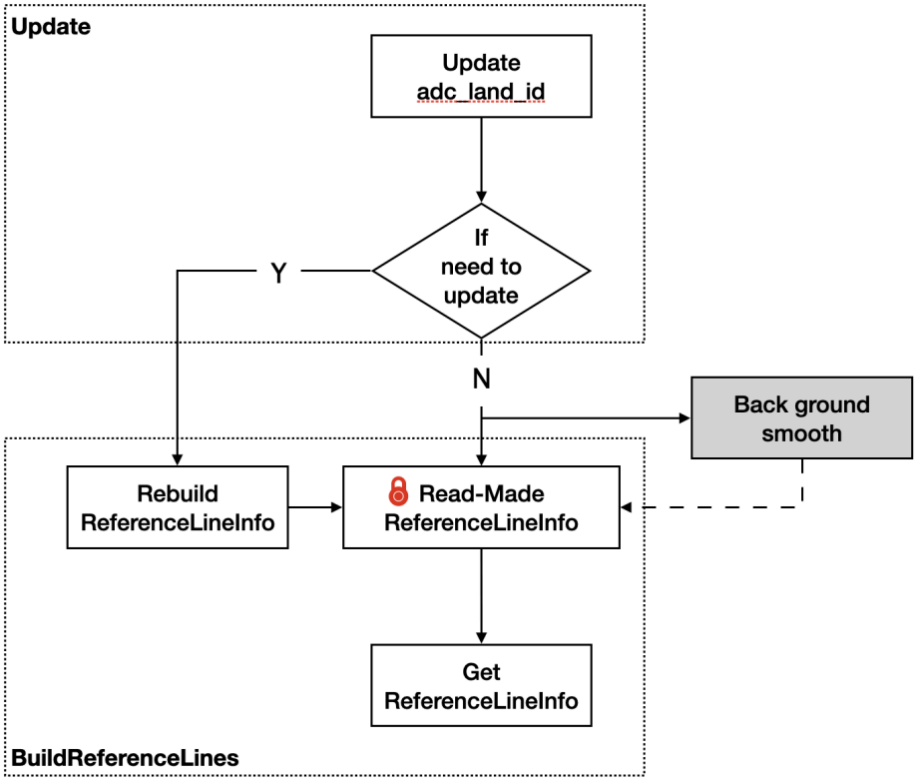
关键成员变量

1. local_view： 在栈中储存，每个world_view都有各自的local_view
2. map_bundle：栈中储存
3. vehicle_state：栈中储存
4. obstacles
5. x_topo_map_
6. latest_x_topo_map_
7. high_level_decision_：FSM所决策的当前帧的executor
8. context_
9. ref_line_bundle_
10. work_summary_：存储当前帧的executor与stage的type、state，以及参考系线、规划的轨迹

ReferenceLineProviderLite

refs: ReferenceLineProvider 设计文档

单例模式，将参考线从每帧的计算中独立出来，当需要更新的时候重新构造参考线，提供向执行器使用。

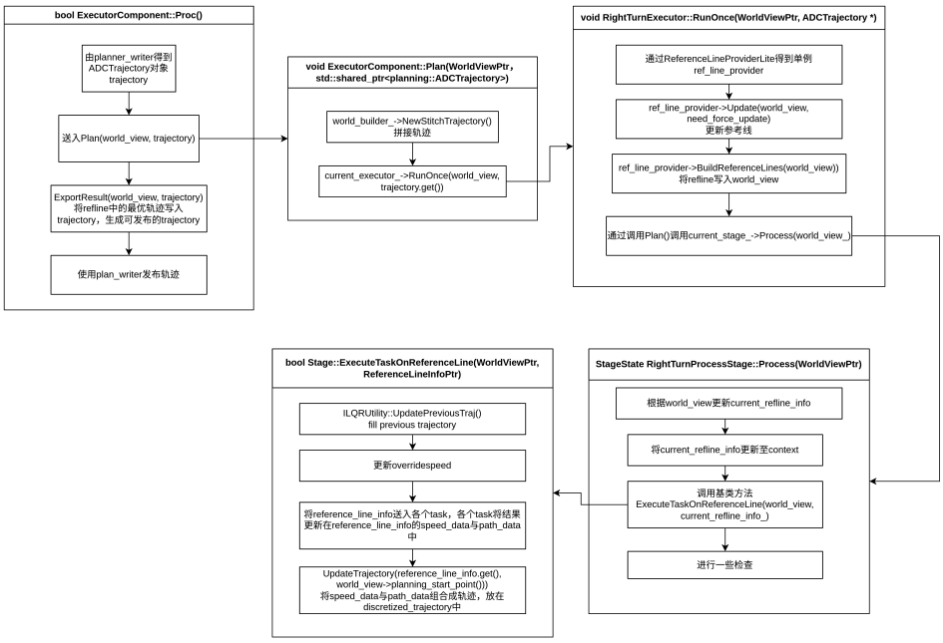


关键函数

1. Update()：拿到线程锁，并进行两个操作：1.更新主车所在的车道 id；2.并检查是否需要更新参考线。如果不需要更新缓存，将触发后台平滑操作
2. BuildReferenceLineInfo()：当检测到需要强更新时，在当前线程内完成参考线的构造，并将结果进行缓存；对缓存中的参考线进行复用，提供给当前帧使用；将生成的多条参考线写入到world_view的reference_line_info中

ReferenceLineInfo

参考线数据结构，存储在world_view中，决策规划Tasks的主要操作对象，每帧重新构建，在Tasks间传递中间数据，规划的轨迹、速度信息都在ReferenceLineInfo里面



关键成员变量

1. ReferenceLine: 参考线信息，包含ReferencePoint
2. NavigationInfo: 高精地图提供的导航信息，包括车道、路口等
3. VehicleInfo: 车辆信息，包含VehicleState，ReferenceLine和起始点
4. BoundaryInfo: 参考线的boundary信息
5. ObstacleInfo: 障碍物信息
6. discretized_trajectory：最终的规划的轨迹

Tasks之间传递的信息

PathBoundary, PathData, SpeedData, SpeedDecision, STGraphData, ilqr_info

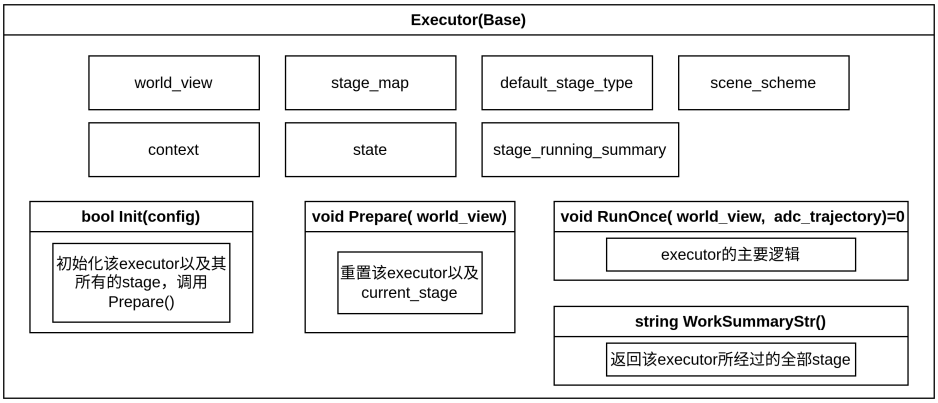
关键函数

1. CombinePathAndSpeedProfile()：将路径规划与速度规划组合为最终的轨迹，存储在discretized_trajectory中

executor

声明：./modules/executor/executor.h

各个场景的执行器，如city_cruise、left_turn等，所有executor均派生自executor基类



关键成员变量

1. world_view_
2. context_：存储该executor所需要的上下文
3. stage_map_：存储该executor的所有stage实例指针

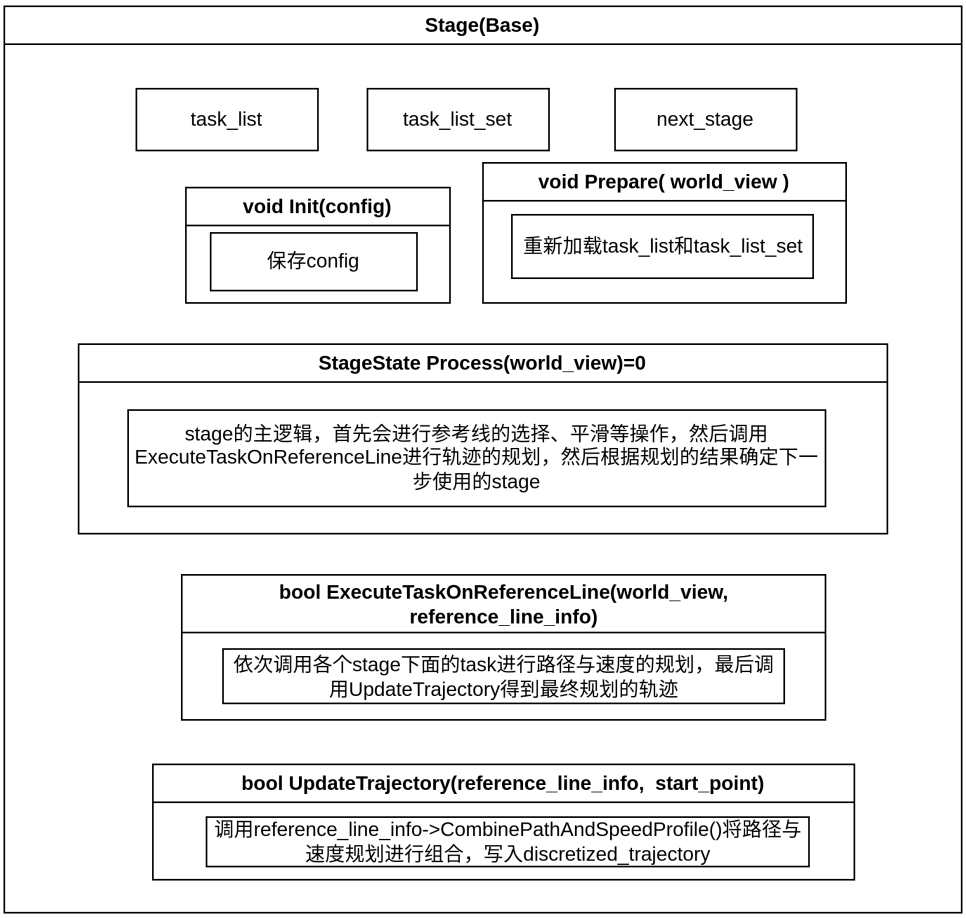
关键函数

1. Init()：加载config，并实例化该executor对应的所有的stage，初始化executor对应的context与scene_scheme，调用Prepare切换默认stage
2. RunOnce()：执行主函数，构造参考线簇 -> Plan() -> 处理一些intervene ->ExportMessage()
3. Plan()：检查drive_mode -> current_stage.Process() ->根据返回的stage_state进行stage切换或其他操作

stage

声明：./lib/executor/lib/stages/stage.h

各个executor对应的不同阶段，如merge_in、lane_change等，所有stage均派生自stage基类



关键成员变量

1. task_list_：存储该stage所有的task，会按顺序依次执行这些task
2. next_stage_：执行完该stage后转移的stage

关键函数

1. Init()：加载config
2. Prepare()：使用TaskFactory实例化所有的task并存放到task_list_中
3. Process()：执行主函数，各个stage需要重写函数实现各自的功能，一般是首先进行参考线的选择，然后调用ExecuteTaskOnReferenceLine()在选择的参考线上调用所有的task进行路径与速度的规划，并组合为最终的轨迹，最后根据执行的状态返回下一帧执行的stage
4. ExecuteTaskOnReferenceLine()：在选择的参考线上执行所有的task进行路径与轨迹的规划，最后会调用UpdateTrajectory()将路径与轨迹组合为轨迹，完成一帧的规划
5. UpdateTrajectory()：调用reference_line_info->CombinePathAndSpeedProfile()将路径与轨迹组合为轨迹

task

声明：./lib/executor/lib/tasks/task.h

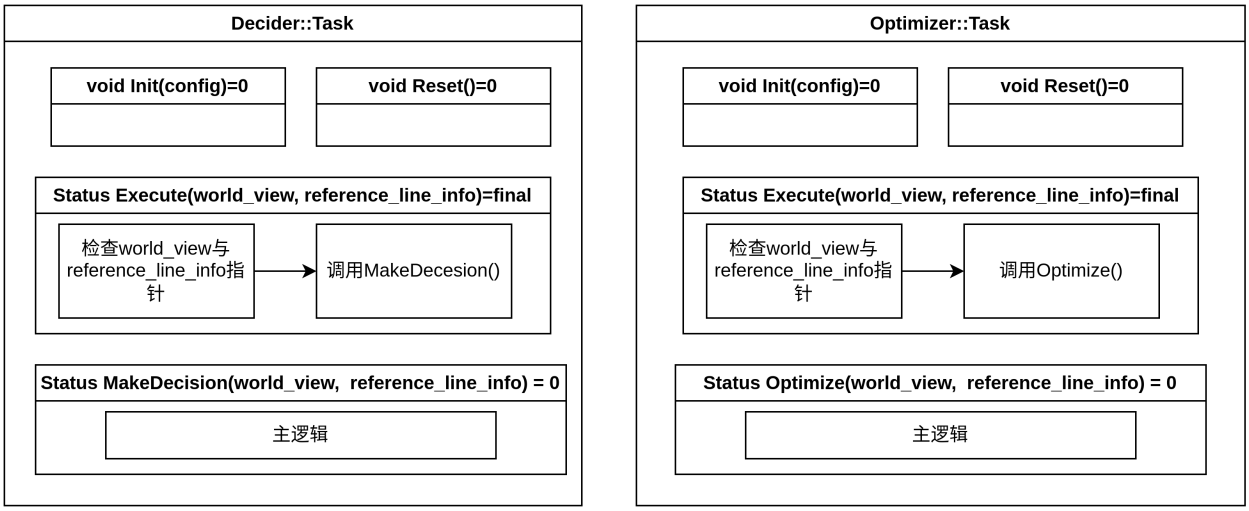
各个stage对应的不同task，分为decider以及optimizer，decider、optimizer基类派生自task基类，所有decider、optimizer均派生自对应的decider、optimizer基类

总体上task可以分为环境建模类：NewTrafficLightDecider、CrosswalkDecider、RightTurnJunctionInfoBuilder、LeftTurnTrafficFlowDecider等

路径规划类：LeftTurnOpenSpaceGuideLineOptimizer、RTLKPathBoundaryDecider

速度决策类：RightTurnCreepDecider、RightTurnCipvDecider、RightTurnMergeDecider等

速度规划类：STGraphBoundaryRTDecider、STRTHeuristicOptimizer、FixedPiecewiseJerkSlackSpeedOptimizer等



关键函数

1. Init()：加载config
2. Execute()：执行主函数，检查输入并调用MakeDecision()或Optimize()
3. MakeDecision()：decider的核心
4. Optimize()：optimizer的核心