roborts_decision

张吉祥

2019 年 1 月 26 日

1 行为树 (BT) 及其实现

1.1 BT 的节点类型

行为树可以说是状态机的升级版，应用领域包括 AI Games 和 Robotics。BT 的节点类型见表格 (图1)。

表 1: 行为树节点类型及其返回结果

Node type	**SUCCESS**	**FAILURE**	**RUNNING**
Selector	一个子节点成功	**全部子节点失败**	一个子节点 RUNNING
Sequence	**全部子节点成功**	一个子节点失败	一个子节点 RUNNING
Parallell	If N children succeeds	If M-N children fail	If all children return running
Decorator	Varies	Varies	Varies
Action	Upon completion	When impossible to complete	During completion
Precondition	If true	If false	Never

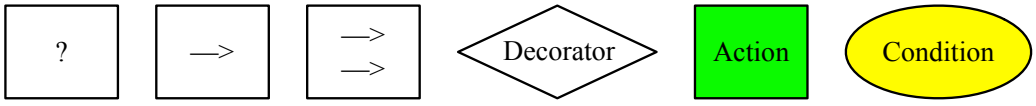
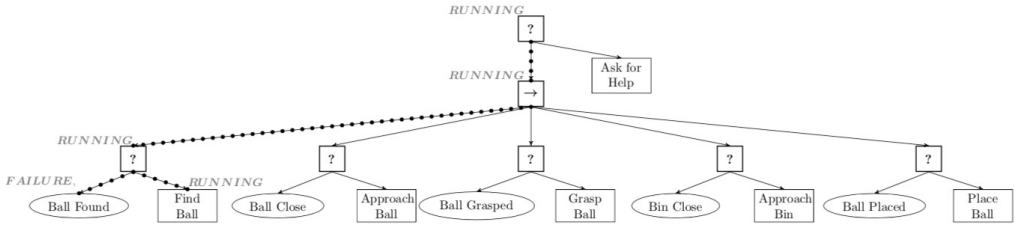
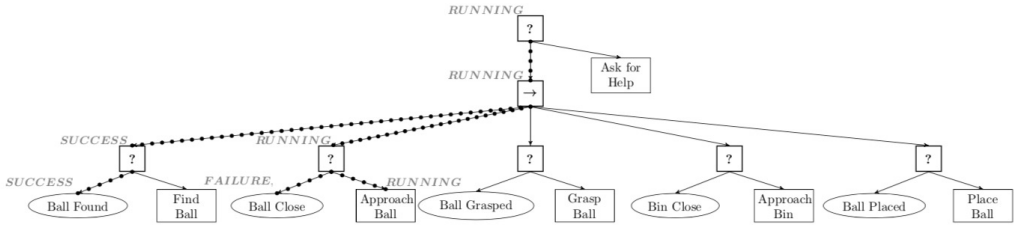


图 1: 行为树的节点表示

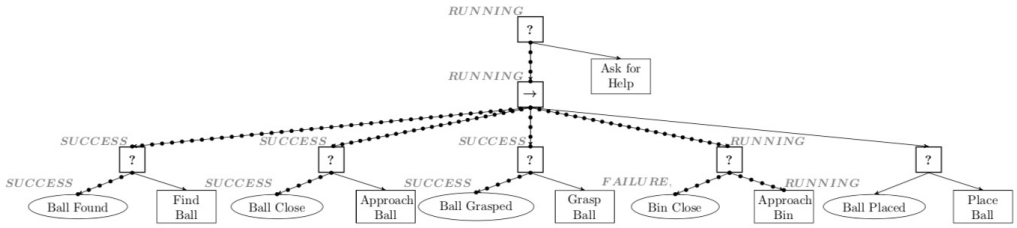
1.2 BT 的运行过程



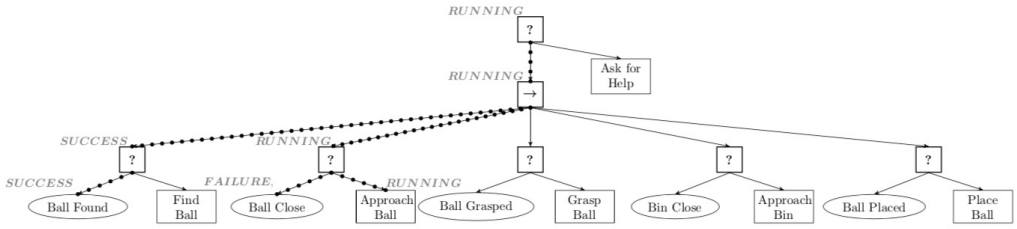
(a) Ticks' traversal when the robot is searching the ball.



(b) Ticks' traversal while the robot is approaching the ball.



(c) Ticks' traversal while the robot is approaching the bin.



(d) Ticks' traversal while the robot is approaching the ball again (because it was removed from the hand).

图 2: BT 运行流程说明

(a) Tick 信号第 1 次触发根节点 Root，Tick 先到达 Ball Found 条件节点，此时机器人不知道球的位置，返回失败；然后 Tick 到达 Find Ball 动作节点，此次返回 RUNNING，RUNNING 结果会一直向上传回

Root 节点，故 Tick 周期的返回状态为 RUNNING；

(b) Tick 信号第 2 次触发根节点 Root，此时机器人直到球的位置，故 Ball Found 条件节点返回成功，此时 Tick 不会传到 Find Ball 动作节点，而会到 Ball Close 条件节点，如果离球太远，返回失败，则到达并执行 Approach Ball 动作节点，此时返回 RUNNING 并向上传回至 Root 节点，故 Tick 周期内的返回状态为 RUNNING；

(c) 同理；

(d) 球从机器人手中掉落，但还在视野中，此次会继续执行 Approach Ball 动作节点，返回状态为 RUNNING。

1.3 软件架构

UML 类图见图3。采用**继承**和**委托**的方式管理各个 Class。Blackboard 作为信息输入源，BT 作为逻辑模块，Action 作为输出行为。

2 BT 使用方法

2.1 程序要点

2.1.1 Topics vs Services vs Actionlib...

Topics should be used for continuous data streams (sensor data, robot state, ...).

Services should be used for remote procedure calls that terminate quickly, e.g. for querying the state of a node or doing a quick calculation such as IK. They should never be used for longer running processes, in particular processes that might be required to preempt if exceptional situations occur and they should never change or depend on state to avoid unwanted side effects for other nodes.

Actions should be used for everything that moves the robot or that runs for a longer time such as perception routines that are triggered by some node and need a couple of seconds to terminate. The most important property of actions is that they can be preempted and preemption should always be implemented cleanly by action servers. Another nice property of actions is that they can keep state for the lifetime of a goal,

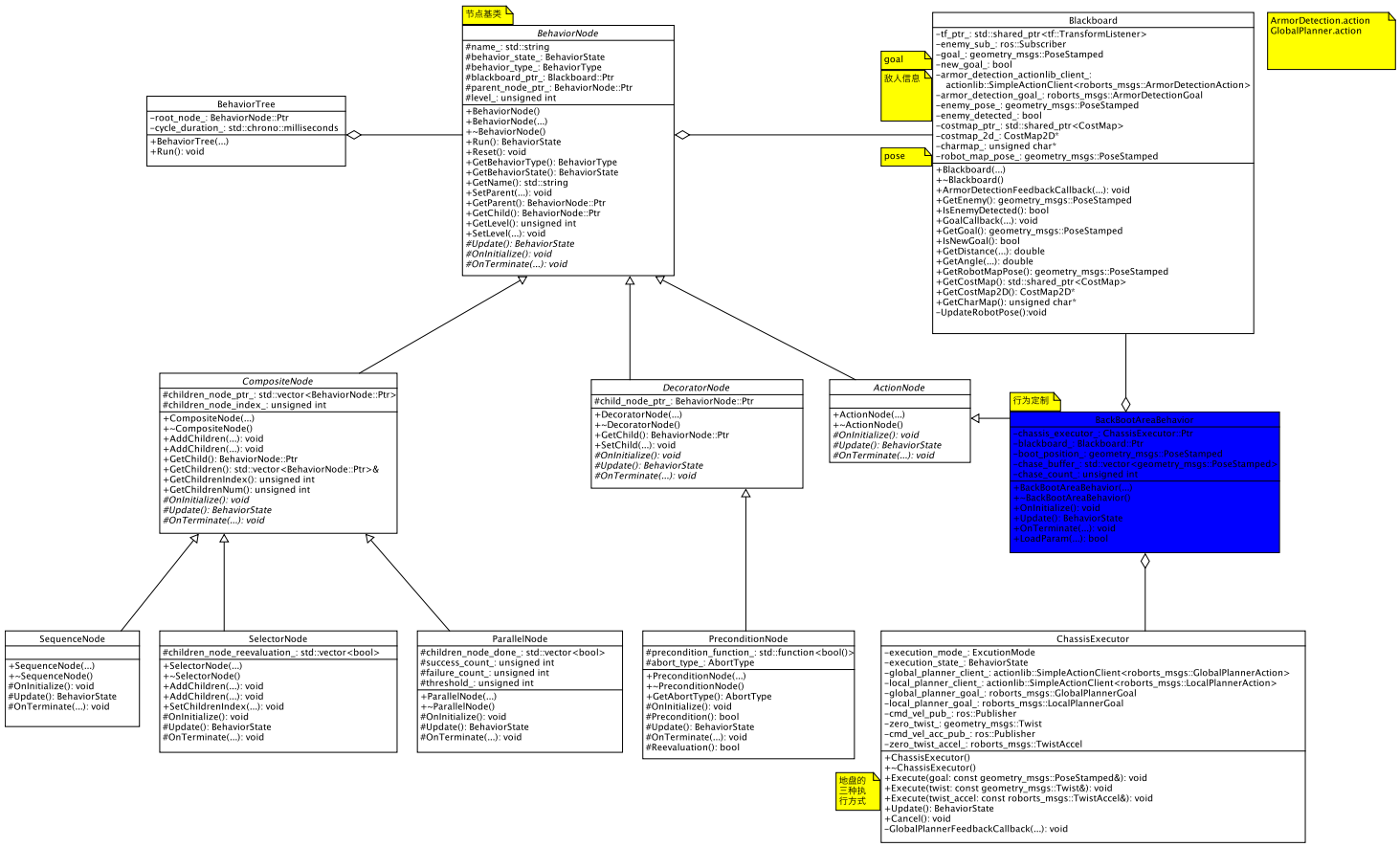

节点基类
BehaviorNode
#name_: std::string
#behavior_state_: BehaviorState
#behavior_type_: BehaviorType
#blackboard_ptr_: Blackboard::Ptr
#parent_node_ptr_: BehaviorNode::Ptr
#level_: unsigned int
+BehaviorNode()
+BehaviorNode(...)
+~BehaviorNode()
+Run(): BehaviorState
+Reset(): void
+GetBehaviorType(): BehaviorType
+GetBehaviorState(): BehaviorState
+GetName(): std::string
+SetParent(...): void
+GetParent(): BehaviorNode::Ptr
+GetChild(): BehaviorNode::Ptr
+GetLevel(): unsigned int
+SetLevel(...): void
#Update(): BehaviorState
#OnInitialize(): void
#OnTerminate(...): void
BehaviorTree
-root_node_: BehaviorNode::Ptr
-cycle_duration_: std::chrono::milliseconds
+BehaviorTree(...)
+Run(): void
Blackboard
-tf_ptr_: std::shared_ptr<tf::TransformListener>
-enemy_sub_: ros::Subscriber
-goal_: geometry_msgs::PoseStamped
-new_goal_: bool
-armor_detection_actionlib_client_:
actionlib::SimpleActionClient<roborts_msgs::ArmorDetectionAction>
-armor_detection_goal_: roborts_msgs::ArmorDetectionGoal
-enemy_pose_: geometry_msgs::PoseStamped
-enemy_detected_: bool
-costmap_ptr_: std::shared_ptr<CostMap>
-costmap_2d_: CostMap2D*
-charmap_: unsigned char*
-robot_map_pose_: geometry_msgs::PoseStamped
+Blackboard(...)
+~Blackboard()
+ArmorDetectionFeedbackCallback(...): void
+GetEnemy(): geometry_msgs::PoseStamped
+IsEnemyDetected(): bool
+GoalCallback(...): void
+GetGoal(): geometry_msgs::PoseStamped
+IsNewGoal(): bool
+GetDistance(...): double
+GetAngle(...): double
+GetRobotMapPose(): geometry_msgs::PoseStamped
+GetCostMap(): std::shared_ptr<CostMap>
+GetCostMap2D(): CostMap2D*
+GetCharMap(): unsigned char*
-UpdateRobotPose():void
goal
故人信息
pose
ArmorDetection.action
GlobalPlanner.action
CompositeNode
#children_node_ptr_: std::vector<BehaviorNode::Ptr>
#children_node_index_: unsigned int
+CompositeNode(...)
+~CompositeNode()
+AddChildren(...): void
+AddChildren(...): void
+GetChild(): BehaviorNode::Ptr
+GetChildren(): std::vector<BehaviorNode::Ptr>&
+GetChildrenIndex(): unsigned int
+GetChildrenNum(): unsigned int
#OnInitialize(): void
#Update(): BehaviorState
#OnTerminate(...): void
DecoratorNode
#child_node_ptr_: BehaviorNode::Ptr
+DecoratorNode(...)
+~DecoratorNode()
+GetChild(): BehaviorNode::Ptr
+SetChild(...): void
#OnInitialize(): void
#Update(): BehaviorState
#OnTerminate(...): void
ActionNode
+ActionNode(...)
+~ActionNode()
#OnInitialize(): void
#Update(): BehaviorState
#OnTerminate(...): void
行为定制
BackBootAreaBehavior
-chassis_executor_: ChassisExecutor::Ptr
-blackboard_: Blackboard::Ptr
-boot_position_: geometry_msgs::PoseStamped
-chase_buffer_: std::vector<geometry_msgs::PoseStamped>
-chase_count_: unsigned int
+BackBootAreaBehavior(...)
+~BackBootAreaBehavior()
+OnInitialize(): void
+Update(): BehaviorState
+OnTerminate(...): void
-LoadParam(...): bool
SequenceNode
+SequenceNode(...)
+~SequenceNode()
#OnInitialize(): void
#Update(): BehaviorState
#OnTerminate(...): void
SelectorNode
#children_node_reevaluation_: std::vector<bool>
+SelectorNode(...)
+~SelectorNode()
+AddChildren(...): void
+AddChildren(...): void
+SetChildrenIndex(...): void
#OnInitialize(): void
#Update(): BehaviorState
#OnTerminate(...): void
ParallelNode
#children_node_done_: std::vector<bool>
#success_count_: unsigned int
#failure_count_: unsigned int
#threshold_: unsigned int
+ParallelNode(...)
+~ParallelNode()
#OnInitialize(): void
#Update(): BehaviorState
#OnTerminate(...): void
PreconditionNode
#precondition_function_: std::function<bool()>
#abort_type_: AbortType
+PreconditionNode(...)
+~PreconditionNode()
+GetAbortType(): AbortType
#OnInitialize(): void
#Precondition(): bool
#Update(): BehaviorState
#OnTerminate(...): void
#Reevaluation(): bool
ChassisExecutor
-execution_mode_: ExcutionMode
-execution_state_: BehaviorState
-global_planner_client_: actionlib::SimpleActionClient<roborts_msgs::GlobalPlannerAction>
-local_planner_client_: actionlib::SimpleActionClient<roborts_msgs::LocalPlannerAction>
-global_planner_goal_: roborts_msgs::GlobalPlannerGoal
-local_planner_goal_: roborts_msgs::LocalPlannerGoal
-cmd_vel_pub_: ros::Publisher
-zero_twist_: geometry_msgs::Twist
-cmd_vel_acc_pub_: ros::Publisher
-zero_twist_accel_: roborts_msgs::TwistAccel
+ChassisExecutor()
+~ChassisExecutor()
+Execute(goal: const geometry_msgs::PoseStamped&): void
+Execute(twist: const geometry_msgs::Twist&): void
+Execute(twist_accel: const roborts_msgs::TwistAccel&): void
+Update(): BehaviorState
+Cancel(): void
-GlobalPlannerFeedbackCallback(...): void
地盘的
三种执
行方式

图 3: BT 类图

i.e. if executing two action goals in parallel on the same server, for each client a separate state instance can be kept since the goal is uniquely identified by its id.

三者的使用方法: `http://wiki.ros.org/ROS/Patterns/Communication#Communication_via_Topics_vs_Services_vs_X`

2.1.2 actionlib 使用方法

Goal 为了使用行为完成任务，我们引入了一个目标的概念，它可以通过一个 ActionClient 发送到一个 ActionServer。在移动底盘的情况下，目标将是一个包含关于机器人应该移动到世界何处的信息的、具有固定功能的信息。为了控制倾斜激光扫描仪，目标将包含扫描参数 (最小角度，最大角度，速度等)。

Feedback 反馈为服务器实现者提供了一种方式来告诉一个 ActionClient 关于一个目标的渐进进展。对于移动底盘，这可能是机器人在路径上的当前姿态。为了控制倾斜的激光扫描仪，这可能是在扫描完成之前的时间。

Result 在完成目标后，将结果从 ActionServer 发送到 ActionClient。这与反馈不同，因为它只发送一次。当行动的目的是提供某种信息时，这是非常有用的。对于移动底盘，其结果并不十分重要，但它可能包含机器人的最终姿态。为了控制倾斜的激光扫描仪，结果可能包含从所请求的扫描产生的点云。

.action 文件 行为规范使用.action 文件。.action 文件有目标定义，然后是结果定义，然后是反馈定义，每个部分用 3 个连字符 (—) 分隔。

SimpleActionClient `http://wiki.ros.org/cn/actionlib`

2.1.3 std::make_shared vs std::shared_ptr

示例

```
1 #include <iostream>
2 #include <memory>
3 #include <type_traits>
4
5 struct C
6 {
```

```
7         C(int i) : i(i) {} // < 需要构造函数 (C++20 前)
8         int i;
9 };
10
11 int main()
12 {
13         auto sp = std::make_shared<C>(12);
14         static_assert(std::is_same_v<decltype(sp), std::shared_ptr<C>>);
15         std::cout << sp->i << '\n';
16 }
```

2.1.4 TransformListener::transformPose(...)

```
1 void TransformListener::transformPose(const std::string& target_frame,
2                                       const geometry_msgs::PoseStamped& stamped_in,
3                                       geometry_msgs::PoseStamped& stamped_out) const
```

功能: Transform a Stamped Pose Message into the target frame. This can throw all that lookupTransform can throw as well as tf::InvalidTransform.

2.1.5 catkin/CMakeLists.txt

- Required CMake Version (cmake_minimum_required)
- Package Name (project())
- Find other CMake/Catkin packages needed for build (find_package())
- Enable Python module support (catkin_python_setup())
- Message/Service/Action Generators (add_message_files(), add_service_files(), add_action_files())
- Invoke message/service/action generation (generate_messages())
- Specify package build info export (catkin_package())

- Libraries/Executables to build (add_library()/add_executable()/target_link_libraries())
- Tests to build (catkin_add_gtest())
- Install rules (install())

http://wiki.ros.org/catkin/CMakeLists.txt

2.2 例子分析

官方给的代码示例中的行为没有继承 ActionNode，无法构建 BT，故需要重新定义自己的行为 Class。

2.2.1 行为 Class 的定义: 继承 ActionNode，完善三个虚函数

```
...
class BackBootAreaBehavior : public ActionNode {
 public:
  BackBootAreaBehavior(roborts_decision::ChassisExecutor::Ptr & chassis_executor,
              const roborts_decision::Blackboard::Ptr & blackboard,
              const std::string & proto_file_path) : ActionNode::ActionNode("back_boot_area_beha
                                                     chassis_executor_(chassis_executor),
                                                     blackboard_(blackboard) {
...
}
...
virtual void OnInitialize() {
  ROS_INFO("OnInitialize...Done");
}

virtual BehaviorState Update() {
  auto robot_map_pose = blackboard_->GetRobotMapPose();
  auto dx = boot_position_.pose.position.x - robot_map_pose.pose.position.x;
  auto dy = boot_position_.pose.position.y - robot_map_pose.pose.position.y;

  auto boot_yaw = tf::getYaw(boot_position_.pose.orientation);
  auto robot_yaw = tf::getYaw(robot_map_pose.pose.orientation);
```

```
23
24      tf::Quaternion rot1, rot2;
25      tf::quaternionMsgToTF(boot_position_.pose.orientation, rot1);
26      tf::quaternionMsgToTF(robot_map_pose.pose.orientation, rot2);
27      auto d_yaw =  rot1.angleShortestPath(rot2);
28
29      if (std::sqrt(std::pow(dx, 2) + std::pow(dy, 2)) > 0.2 || d_yaw > 0.5) {
30        chassis_executor_->Execute(boot_position_);
31      }
32
33      return chassis_executor_->Update();
34    }
35
36    virtual void OnTerminate(BehaviorState state) {
37      chassis_executor_->Cancel();
38      switch (state){
39        case BehaviorState::IDLE:
40          ROS_INFO("OnTerminate...IDLE");
41          break;
42        case BehaviorState::SUCCESS:
43          ROS_INFO("OnTerminate...SUCCESS");
44          break;
45        case BehaviorState::FAILURE:
46          ROS_INFO("OnTerminate...FAILURE");
47          break;
48        default:
49          ROS_INFO("OnTerminate...ERROR");
50          return;
51      }
52    }
53    ...
54    private:
55    //! executor
56    roborts_decision::ChassisExecutor::Ptr chassis_executor_;
57
```

```
58   //! perception information
59   roborts_decision::Blackboard::Ptr blackboard_;
60   ...
```

2.2.2 运行节点

```
1  int main(int argc, char **argv) {
2          ros::init(argc, argv, "behavior_test_node");
3          std::string full_path = ros::package::getPath("roborts_decision") + "/config/decision.prot
4          auto blackboard = std::make_shared<roborts_decision::Blackboard>(full_path);
5          std::cout << "here1" << std::endl;
6
7          // () is vital!!!
8          auto chassis_executor = std::make_shared<roborts_decision::ChassisExecutor>();
9          std::cout << "here2" << std::endl;
10
11         auto back_boot_area_behavior = std::make_shared<roborts_decision::BackBootAreaBehavior>(ch
12         std::cout << "here3" << std::endl;
13
14         std::string bot_selector = "bot selector";
15         auto bot_selector_ = std::make_shared<roborts_decision::SelectorNode>(bot_selector, blackb
16         std::cout << "here4" << std::endl;
17
18         bot_selector_->AddChildren(back_boot_area_behavior);
19         std::cout << "here5" << std::endl;
20
21         auto root = std::make_shared<roborts_decision::BehaviorTree>(bot_selector_, 250);
22         root->Run();
23         return 0;
24 }
```

编译：catkin_make -j4 behavior_test_node

运行：rosrun roborts_decision behavior_test_node

References

- https://robomaster.github.io/RoboRTS-Tutorial/#/sdk_docs/roborts_decision
- catkin/CMakeLists.txt http://wiki.ros.org/catkin/CMakeLists.txt