3.2 导航节点

3.2.1 ROS Action 的 BT action Node 封装模板

基于 BT 节点的 ROS Action 抽象类

一个 BT (ROS) action Node 至少包含以下内容：

- action client 的建立
- 发送目标请求
- 通信异常处理
- 抢占更新目标
- 输入标准端口
- 输出节点状态
- 中止 ROS action

1. BtActionNode 构造函数

Nav2: node_，callback_group_(强制回调的并发规则)，callback_group_executor_(负责回调的实际执行)，输入输出消息的初始化，createActionClient()函数实例创建 ROS Action。

Navit: 去除 callback_group_，callback_group_executor_

输入输出

ROS2: class rclpy.action.client.**ClientGoalHandle**(action_client, goal_id, goal_response)

Goal handle for working with Action Clients. ROS2 中该类可以获取当前目标状态

ROS1: actionlib::ClientGoalHandle< ActionSpec >::**ClientGoalHandle**()

Client side handle to monitor goal progress

特别注意，客户端目标状态在不同版本的区别

2. createActionClient 实例创建 Action 客户端

通过已有 ROS 节点为 BT Action 创建 Action 客户端

Nav2:创建 action client，打印信息，等待相应

类型：rclcpp_action::Client

Navit: ROS1 化，无特殊函数。

类型：actionlib::SimpleActionClient（有继承关系且全局局部服务端都是 SimpleAction）

3. providedBasicPorts 标准端口，参考 BT_doc 2.1.2，**Navit** 无修改变动

4. BT-ROS-action 派生类重写的虚函数，**Navit** 无修改变动

函数	作用
on_tick	-
on_wait_for_result	-
on_success	return BT::NodeStatus::SUCCESS
on_aborted	return BT::NodeStatus::FAILURE
on_cancelled	return BT::NodeStatus::SUCCESS

6.send_new_goal() 发送目标

发送目标至服务端，发送行为可以通过 ROS1 实现。但是，

```
auto send_goal_options = typename rclcpp_action::Client<ActionT>::SendGoalOptions();
send_goal_options.result_callback =
 [this](const typename rclcpp_action::ClientGoalHandle<ActionT>::WrappedResult & result) {
  if (future_goal_handle_) {
   RCLCPP_DEBUG(
    node_->get_logger(),
    "Goal result for %s available, but it hasn't received the goal response yet. "
    "It's probably a goal result for the last goal request", action_name_.c_str());
   return;
  }

  // TODO(#1652): a work around until rcl_action interface is updated
  // if goal ids are not matched, the older goal call this callback so ignore the result
  // if matched, it must be processed (including aborted)
  if (this->goal_handle_->get_goal_id() == result.goal_id) {
   goal_result_available_ = true;
   result_ = result;
  }
 };
```

这里对 client 的 goal_id 通过 ClientGoalHandle 类 **get_goal_id**()进行匹配。获取目标 id 的函数目前在 ROS1 中未查询到。ROS2 SendGoalOptions 理解？(发送选项)

Navit 变更：

```
action_client_->async_send_goal(goal_, send_goal_options));
action_client_->sendGoal(goal_)
```

```
future_goal_handle_ = std::make_shared<std::shared_future<typename
actionlib::ClientGoalHandle<ActionT>::SharedPtr>>(action_client_->sendGoal(goal_));
```

future_goal_handle_会在 tick()作检测，并记录发送时间 time_goal_sent_。

7.is_future_goal_handle_complete() 服务超时检测处理

检查服务端是否已确认(接到)新目标，输入：当前时刻与 send_new_goal 时记录的最后目标的时间。

1. 服务端超时判断，重置 future_goal_handle_返回 false；

2. 服务端状态判断（高度依赖 callback_group_executor）

ROS2 回调执行器强制等待

```
auto timeout = remaining > bt_loop_duration_ ? bt_loop_duration_ : remaining;
auto result = callback_group_executor_.spin_until_future_complete(*future_goal_handle_, timeout);
if (result == rclcpp::FutureReturnCode::SUCCESS) {...}
```

再通过回调执行器状态 FutureReturnCode 来判断服务端状态

Type	brief	Possible States
ROS2 <rclcpp_action::ResultCode >	The possible statuses that an action goal can finish with.	UNKNOWN, SUCCEEDED, CANCELED, ABORTED

Navit 变更：无法取消 服务端状态判断 仅仅取消回调执行器的依赖。参见流程图，必须使节点跳出通信异常的判断和处理。

Fcn name	brief	Possible States
<ClientGoalHandle> getCommState()	Get the state of this goal's communication state machine from interaction with the server	CommState: WAITING_FOR_GOAL_ACK, PENDING, ACTIVE, WAITING_FOR_RESULT, WAITING_FOR_CANCEL_ACK, RECALLING, PREEMPTING
<ClientGoalHandle> getTerminalState()	Get the terminal state information for this goal.	TerminalState: RECALLED, REJECTED, PREEMPTED, ABORTED, SUCCEEDED, LOST
<SimpleActionClient> getState()	Get the state information for this goal.	PENDING, ACTIVE, RECALLED, REJECTED, PREEMPTED, ABORTED, SUCCEEDED, LOST.

又因为，ROS1：

```
switch (comm_state_.state_) {
  case CommState::WAITING_FOR_GOAL_ACK:
  case CommState::PENDING:
  case CommState::RECALLING:
    return SimpleClientGoalState(SimpleClientGoalState::PENDING);
  case CommState::ACTIVE:
  case CommState::PREEMPTING:
    return SimpleClientGoalState(SimpleClientGoalState::ACTIVE);
```

所以：

```
if (action_client_->getState() == "ACTIVE") {
  goal_handle_ = future_goal_handle_; // tick()使用
  return true;
}
```

5.tick()

BT 状态 Idle->Running，实现：发送目标，通信维护，目标抢占，输出结果。

user defined callback. May modify the value of "goal_updated_". 目标抢占由变量 **goal_updated_** 实现。

1. future_goal_handle_

ROS2：异步发送目标会自然的返回目标句柄以便直接确认服务端接收状态（好处带 ID）

- If the goal is accepted by an action server, the returned future is set to a `ClientGoalHandle`.
- If the goal is rejected by an action server, then the future is set to a `nullptr`.

Navit： action 通过 connection_monitor 对服务端连接进行判断。

```
if (action_client_->isServerConnected() {...}
```

2. callback_group_executor_.spin_some();
ROS2 节点回调执行机制，未作深究，navit ros::spin();代替。

3. 目标更新/抢占
ROS2：GoalUUID   goal_id
    The unique identifier of the goal.

```
this->goal_handle_->get_goal_id() == result.goal_id

(goal_status == actionlib_msgs::GoalStatus::STATUS_EXECUTING ||
 goal_status == action_msgs::msg::GoalStatus::STATUS_ACCEPTED)
```

Navit：

方法 1：goal_status == actionlib::CommState::StateEnum::**WAITING_FOR_RESULT**
方法 2：action_client_->getState() == actionlib::SimpleClientGoalState::StateEnum::ACTIVE

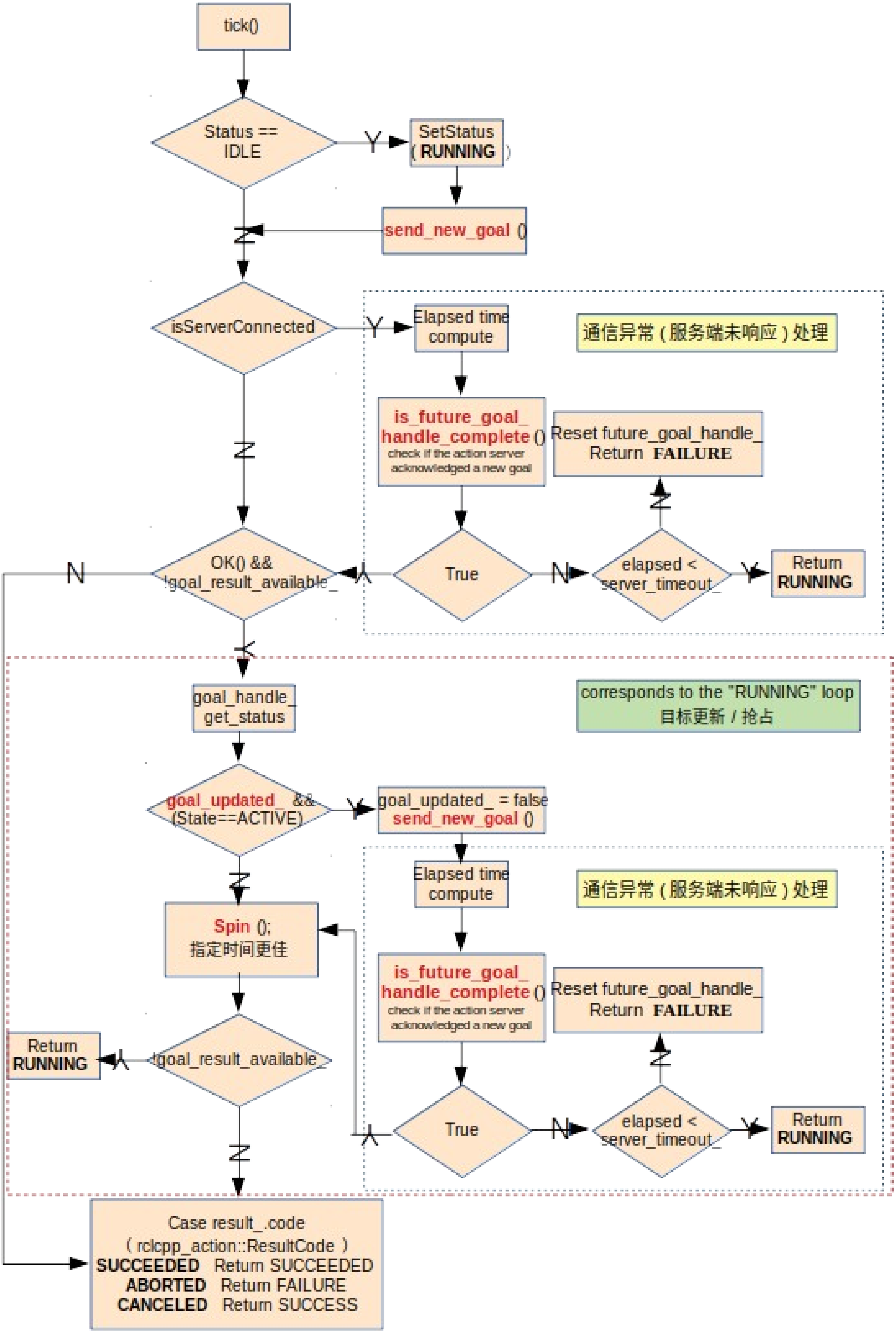
4. BT 输出结果 ResultCode
ROS2：
rclcpp_action::ResultCode:: SUCCEEDED, ABORTED, CANCELED

```
 switch (result_.code) {
   case rclcpp_action::ResultCode::SUCCEEDED:  … }
```

Navit

方法 1：goal_handle_→getTerminalState()   ---------   ABORTED, SUCCEEDED, LOST
方法 2：action_client_→getState()   ---------   ABORTED, SUCCEEDED, ACTIVE

Navit-bt-ros-action tick()流程：


tick()
Status == IDLE
Y
SetStatus (RUNNING)
send_new_goal ()
N
isServerConnected
Y
Elapsed time compute
通信异常(服务端未响应)处理
is_future_goal_handle_complete ()
check if the action server acknowledged a new goal
Reset future_goal_handle_
Return FAILURE
N
OK() && !goal_result_available_
Y
True
N
elapsed < server_timeout_
Y
Return RUNNING
N
Y
goal_handle_ get_status
corresponds to the "RUNNING" loop
目标更新 / 抢占
goal_updated_ && (State==ACTIVE)
Y
goal_updated_ = false
send_new_goal ()
Elapsed time compute
通信异常(服务端未响应)处理
is_future_goal_handle_complete ()
check if the action server acknowledged a new goal
Reset future_goal_handle_
Return FAILURE
N
N
Spin ();
指定时间更佳
Y
True
N
elapsed < server_timeout_
Y
Return RUNNING
Return RUNNING
Y
!goal_result_available_
N
Case result_.code
(rclcpp_action::ResultCode)
SUCCEEDED Return SUCCEEDED
ABORTED Return FAILURE
CANCELED Return SUCCESS