1. 各个文件的解释：
   1. 各文件的调用关系：

explore.cpp/h

* + 1. costmap\_client.cpp/h
    2. frontier\_search.cpp/h
       1. costmap\_tools.h
  1. explore.cpp/h 主文件，包含main函数。
  2. costmap\_client.cpp/h： 通过接受gmapping生成的/map和/tf，更新一张用于算法的地图【程序中的变量名为costmap\_data】。
  3. frontier\_search.cpp/h：从地图costmap\_data中，搜索frontier
  4. costmap\_tools.h：三个在costmap中搜索cell的工具。分别是：提供index cell的4联通cell、index cell的8联通cell、nearestCell

1. 话题：
   1. 发布：

visualization\_msgs::MarkerArray> /frontiers

* 1. 接收：
* geometry\_msgs::PoseWithCovarianceStamped /initialpose
* nav\_msgs::OccupancyGrid /map 【有同步和异步两个接受渠道】
* map\_msgs::OccupancyGridUpdate /move\_base/global\_costmap/costmap\_updates
* tf变换： std::string tf\_prefix = tf::getPrefixParam(param\_nh); robot\_base\_frame\_ = tf::resolve(tf\_prefix, robot\_base\_frame\_);

1. 【构造函数】：costmap\_client.cpp/h中的构造函数：

Costmap2DClient::Costmap2DClient(ros::NodeHandle& param\_nh ,

ros::NodeHandle& subscription\_nh,

const tf::TransformListener\* tf): tf\_(tf)

1. 为什么要用三个ros node
2. 在这个构造函数中，必须得全部获得tf和map，否则报错。
3. 【c++】：为什么tf\_被const修饰，却还能赋值

const tf::TransformListener\* const tf\_; 【此时 \*tf\_ 和 tf\_ 都被修饰了，那么 tf\_ 中存放的内存单元的地址和内存单元中的内容都不可变。[链接](http://c.biancheng.net/view/218.html)】

const tf::TransformListener\* tf = &tf\_listener\_;

tf\_ = tf;

1. 【c++】double上的无穷大

output.min\_distance = std::numeric\_limits<double>::infinity();

1. 【函数】：获取机器人的pose：

geometry\_msgs::Pose Costmap2DClient::getRobotPose() const

1. const修饰函数：<https://blog.csdn.net/qq_32739503/article/details/83341222>
2. tf\_->transformPose(global\_frame\_, robot\_pose, global\_pose); 将base\_link下的robot\_pose【已知】变化为map【global\_frame\_，已知】下的global\_pose【计算结果】，tf\_【计算工具】作为tf::TransformListener\*的ros节点，是实时从ros中接受tf关系的。根据robot\_pose.setIdentity(); 可知 robot\_pose就是机器人 frame的原点
3. tf::poseStampedTFToMsg(global\_pose, msg);
4. 【函数】：更新全部的地图和局部的地图

void Costmap2DClient::updateFullMap(const nav\_msgs::OccupancyGrid::ConstPtr& msg)

1. 【函数】：msg中的内容是0~100，而 costmap中的值，是0~255。利用cost\_translation\_table\_\_这个静态array（static const std::array<unsigned char, 256>），可以快速实现[0,100]到[0,255]的映射：

for (size\_t i = 0; i < costmap\_size && i < msg->data.size(); ++i) {

unsigned char cell\_cost = static\_cast<unsigned char>(msg->data[i]); costmap\_data[i] = cost\_translation\_table\_\_[cell\_cost];

}

解析：用char作为data的数据类型。

1. 【函数】：接收/map

中通过以下两个：

* 1. costmap\_sub\_ = subscription\_nh.subscribe<nav\_msgs::OccupancyGrid>( costmap\_topic, 1000,

[this](const nav\_msgs::OccupancyGrid::ConstPtr& msg) { updateFullMap(msg);});

* 1. auto costmap\_msg = ros::topic::waitForMessage<nav\_msgs::OccupancyGrid>

来接受同一个topic，其中waitForMessage为同步，subscribe为异步

目的是什么？

1. C++学习：
   1. costmap\_sub\_ = subscription\_nh.subscribe<nav\_msgs::OccupancyGrid>(

costmap\_topic, 1000,

[this](const nav\_msgs::OccupancyGrid::ConstPtr& msg) {

updateFullMap(msg);

}

回调函数： [this](const nav\_msgs::OccupancyGrid::ConstPtr& msg){}

[this] this 是 C++ 中的一个关键字，也是一个 const 指针，它指向当前对象，通过它可以访问当前对象的所有成员。

);

[](){} 匿名函数：<https://www.cnblogs.com/Brickert/p/13164291.html>

<https://blog.csdn.net/yhj_911/article/details/125346795>

* 1. ros::topic::waitForMessage<nav\_msgs::OccupancyGrid>( costmap\_topic, subscription\_nh)

答：ROS中有两种获取消息的方式。一种是同步的，一种是异步的。同步的方式:

函数为：boost::shared\_ptr<M const> waitForMessage(const std::string& topic, NodeHandle& nh, ros::Duration timeout)

返回值为所获取的消息。调用后程序将在这里等待该消息，获得后会继续执行后续的程序。

异步的方式:异步的方式是通过subscriber，并使用回调函数的方式来实现的。

1. 【数据结构】frontier

struct Frontier {

std::uint32\_t size; frontier中cell的数量

double min\_distance; frontier中离robot最近的一个cell的距离

double cost; 到机器人坐标的路途cost

geometry\_msgs::Point initial; initial\_cell在地图上的坐标

geometry\_msgs::Point centroid; frontier中所有cell的坐标均值。将会被用作navigation goal。

geometry\_msgs::Point middle; 离robot最近的cell的坐标

std::vector<geometry\_msgs::Point> points; frontier上所有的cell

};

1. 【函数】grid map中index到cell到world

costmap\_->indexToCells(nbr, mx, my);

costmap\_->mapToWorld(mx, my, wx, wy);0

1. 【函数】找到 值为value的cell

nearestCell

1. 【构造函数】FrontierSearch(costmap\_2d::Costmap2D\* costmap,

double potential\_scale, double gain\_scale, double min\_frontier\_size);

1. 【C++】std::find\_if\_not
2. 【函数】规划路径：makePlan()
3. 【函数】todo：

exploring\_timer\_ = relative\_nh\_.createTimer(ros::Duration(1. / planner\_frequency\_), [this](const ros::TimerEvent&) { makePlan(); });

1. 【函数】movebase

move\_base\_client\_.waitForServer();

move\_base\_client\_.sendGoal(

goal, [this, target\_position](

const actionlib::SimpleClientGoalState& status,

const move\_base\_msgs::MoveBaseResultConstPtr& result) {

reachedGoal(status, result, target\_position);

});

move\_base\_client\_.cancelAllGoals();

1. 【函数】 Timers(定时器)

exploring\_timer\_ = relative\_nh\_.createTimer(ros::Duration(1. / planner\_frequency\_),

[this](const ros::TimerEvent&) { makePlan(); });

ros::Timer timer = n.createTimer(ros::Duration(0.1), timerCallback);

其中回调函数为void timerCallback(const ros::TimerEvent& e);

exploring\_timer\_.start();

exploring\_timer\_.stop();

1. 【函数】清空rviz中marker

m.action = visualization\_msgs::Marker::DELETE;

for (; id < last\_markers\_count\_; ++id) {

m.id = int(id);

markers.push\_back(m);

}

1. 【函数】

move\_base\_client\_.sendGoal(

goal, [this, target\_position](

const actionlib::SimpleClientGoalState& status,

const move\_base\_msgs::MoveBaseResultConstPtr& result) {

reachedGoal(status, result, target\_position);

});

self.move\_base.send\_goal(goal，self.donecb，self.activecb，self.feedbackcb)

第一个参数是要发送的目标点后面三个为回调函数，分别是：

当机器人到达的回调函数。

当机器人启动的回调函数。

当机器人以一定频率返回信息的回调函数

1. 【函数】

// find new goal immediatelly regardless of planning frequency.

// execute via timer to prevent dead lock in move\_base\_client (this is

// callback for sendGoal, which is called in makePlan). the timer must live

// until callback is executed.

oneshot\_ = relative\_nh\_.createTimer(

ros::Duration(0, 0), [this](const ros::TimerEvent&) { makePlan(); },

true);

1. 还要看的
   1. 程序里面的todo
   2. ~~ros的周期控制器exploring\_timer\_~~
   3. ~~movebase处的匿名函数 怎么用的？~~
   4. 为什么要用匿名函数，直接调用回调函数不行吗？
   5. ~~costmap\_client这个类中的ros node是怎么一直接收/map topic，并传输到frontier\_search中的。是 多线程吗？？还是靠的ros周期控制器？？~~
      1. ~~frontier\_search中的costmap\_2d::Costmap2D\* costmap\_是一个指针，可以随着costmap\_client中costmap\_的改变而改变。~~
      2. ~~通过global\_pose = costmap\_client\_.getRobotPose();实时获取机器人在map中的坐标~~
      3. ~~costmap\_client不是多线程。说明rosnode注册的subscribe会一直运行~~
   6. ~~为什么tf\_listener\_ 赋值为(ros::Duration(10.0))？~~
   7. ~~为什么ros::NodeHandle& param\_nh ，ros::NodeHandle& subscription\_nh 都没有起ros的名字~~
   8. searchfrom中的 global\_pose是怎么修改的？
   9. 地图更新的太慢？