1.概述

```
planner_costmap_ros_ = new costmap_2d::Costmap2DROS("global_costmap", tf_);
planner_costmap_ros_->pause();

//initialize the global planner
try {
  planner_ = bgp_loader_.createInstance(global_planner);
  planner_->initialize(bgp_loader_.getName(global_planner), planner_costmap_ros_);
} catch (const pluginlib::PluginlibException& ex) {
  ROS_FATAL("Failed to create the %s planner, are you sure it is properly registered and that t
  exit(1);
}

//create the ros wrapper for the controller's costmap... and initializer a pointer we'll use with t
controller_costmap_ros_ = new costmap_2d::Costmap2DROS("local_costmap", tf_);
controller_costmap_ros_->pause();
```

move_base构造函数中会构造local_cost_map和global_costmap两个对象，同时构造他们时会根据参数添加相应的层

```
if (private_nh.hasParam("plugins"))
{
  XmlRpc::XmlRpcValue my_list;
  private_nh.getParam("plugins", my_list);
  for (int32_t i = 0; i < my_list.size(); ++i)
  {
    std::string pname = static_cast<std::string>(my_list[i]["name"]);
    std::string type = static_cast<std::string>(my_list[i]["type"]);
    ROS_INFO("Using plugin \"%s\"", pname.c_str());

    boost::shared_ptr<Layer> plugin = plugin_loader_.createInstance(type);
    layered_costmap_->addPlugin(plugin);
    plugin->initialize(layered_costmap_, name + "/" + pname, &tf_);
  }
}
```

这些参数分别在

```
costmap_common_params_apollo.yaml
local_costmap_params_withoutmap.yaml
global_costmap_params_withoutmap.yaml
```

显然第一个为共用的

2.common

robot_radius 原型底盘即为半径 footprint 非原型，以旋转中心为原点，各个顶点按顺序(逆时针/顺时针都可以)的坐标

优先会查找footprint

```
if (nh.searchParam("footprint", full_param_name))
{
  XmlRpc::XmlRpcValue footprint_xmlrpc;
  nh.getParam(full_param_name, footprint_xmlrpc);
  if (footprint_xmlrpc.getType() == XmlRpc::XmlRpcValue::TypeString)
  {
    if (makeFootprintFromString(std::string(footprint_xmlrpc), points))
    {
      writeFootprintToParam(nh, points);
    }
  }
  else if (footprint_xmlrpc.getType() == XmlRpc::XmlRpcValue::TypeArray)
  {
    points = makeFootprintFromXMLRPC(footprint_xmlrpc, full_param_name);
    writeFootprintToParam(nh, points);
  }
}
else if (nh.searchParam("robot_radius", full_radius_param_name))
{
  double robot_radius;
  nh.param(full_radius_param_name, robot_radius, 1.234);
  points = makeFootprintFromRadius(robot_radius);
  nh.setParam("robot_radius", robot_radius);
}
```

3.global_costmap

```
global_costmap:
   global_frame: /odom
   robot_base_frame: /base_link
   update_frequency: 1.0
   publish_frequency: 0.5
   static_map: false
   rolling_window: true
   width: 12
   height: 12
   resolution: 0.05
   transform_tolerance: 0.5
   plugins:
     - {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
     - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

- global_frame 全局坐标系， 现在我们使用没有图的，这里使用/odom
- robot_base_frame 机器人坐标系
- update_frequency map更新的频率(说好的没有图的呢， 之前无图是没有传递加载实际的地图，还是要到心中有图的，这里指的costmap）
- publish_frequency map发布的频率

- `static_map` 该参数一般总是与下一个相反的， 标识使用静态地图，我们没有使用，这里当然是`false`
- `rolling_window` `true` 标识地图跟随便机器人
- `width` `height` `resolution` 地图信息
- `transform_tolerance` tf的超时时间
- `plugins` 图层

4.local_costmap

```
local_costmap:
   global_frame: /odom
   robot_base_frame: /base_link
   update_frequency: 1.0
   publish_frequency: 2.0
   static_map: false
   rolling_window: true
   width: 4
   height: 4
   resolution: 0.05
   transform_tolerance: 0.5
   plugins:
    - {name: obstacle_layer,      type: "costmap_2d::VoxelLayer"}
    - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}
```

与`global_costmap`基本一致