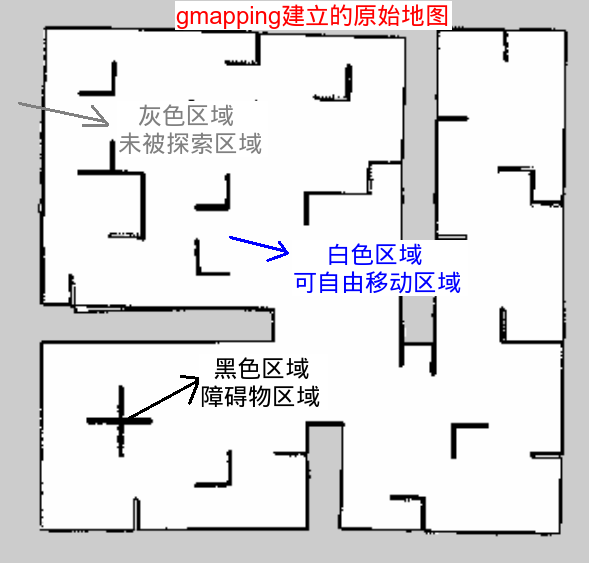
**costmap代价地图**

在机器人进行路径规划时，我们需要明白规划算法是依靠什么在地图上来计算出来一条路径的。依靠的是gmapping扫描构建的一张环境全局地图，但是仅仅依靠一张原始的全局地图是不行的。因为这张地图是静态的，无法随时来更新地图上的障碍物信息。在现实环境中，总会有各种无法预料到的新障碍物出现在当前地图中，或者旧的障碍物现在已经从环境地图中被移除掉了，那么我们就需要来随时更新这张地图。同时由于默认的地图是一张黑白灰三色地图，即只会标出障碍物区域、自由移动区域和未被探索区域。机器人在这样的地图中进行路径规划，会导致规划的路径不够安全，因为我们的机器人在移动时需要与障碍物之间保持一定的安全缓冲距离，这样机器人在当前地图[中移动](https://www.baidu.com/s?wd=%E4%B8%AD%E7%A7%BB%E5%8A%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)时就更安全了。



costmap简单来说就是为了在这张地图上进行各种加工，方便我们后面进行路径规划而存在的。那具体该如何实现costmap呢？在ROS中使用costmap\_2d这个软件包来实现的，该软件包在原始地图上实现了两张新的地图。一个是local\_costmap，另外一个就是global\_costmap，根据名字大家就可以知道了，两张costmap一个是为局部路径规划准备的，一个是为全局路径规划准备的。无论是local\_costmap还是global\_costmap，都可以配置多个图层，包括下面几种：

* Static Map Layer：静态地图层，基本上不变的地图层，通常都是SLAM建立完成的静态地图。
* Obstacle Map Layer：障碍地图层，用于动态的记录传感器感知到的障碍物信息。
* Inflation Layer：膨胀层，在以上两层地图上进行膨胀（向外扩张），以避免机器人的撞上障碍物。
* Other Layers：你还可以通过插件的形式自己实现costmap，目前已有Social Costmap Layer、Range Sensor Layer等开源插件。

在启动move\_base节点时，首先加载了costmap\_common\_params.yaml到global\_costmap和local\_costmap两个命名空间中，因为该配置文件是一个通用的代价地图配置参数，即local\_costmap和global\_costmap都需要配置的参数。然后下面是local\_costmap\_params.yaml专门为了局部代价地图配置的参数，global\_costmap\_params.yaml专门为全局代价地图配置的参数。

**2.  配置costmap\_common\_params.yaml**

|  |
| --- |
| #Description:  # 代价地图通用参数配置文件,就是全局代价地图和局部代价地图  # 共同都需要配置的参数,各参数意义如下:  # robot\_radius: 机器人的半径  #  #History:  # 20180613: initial this file.    robot\_radius: 0.2    obstacle\_layer:    enabled: true    combination\_method: 1    track\_unknown\_space: true    obstacle\_range: 2.5    raytrace\_range: 3.0    observation\_sources: laser\_scan\_sensor    laser\_scan\_sensor: {      sensor\_frame: /robot0\_laser\_0,      data\_type: LaserScan,      topic: /robot0/laser\_0,      marking: true,      clearing: true    }    inflation\_layer:    enabled: true    cost\_scaling\_factor: 5.0    inflation\_radius: 0.36  static\_layer:    enabled: true |

* robot\_radius：设置机器人的半径，单位是米。由于在stdr中机器人是圆形的，所以可以直接设置该参数。如果你的机器人不是圆形的那就需要使用footprint这个参数，该参数是一个列表，其中的每一个坐标代表机器人上的一点，设置机器人的中心为[0,0]，根据机器人不同的形状，找到机器人各凸出的坐标点即可，
* obstacle\_layer:配置障碍物图层
* enabled:是否启用该层
* combination\_method:只能设置为0或1，用来更新地图上的代价值，一般设置为1;
* track\_unknown\_space:如果设置为false，那么地图上代价值就只分为致命碰撞和自由区域两种，如果设置为true，那么就分为致命碰撞，自由区域和未知区域三种。意思是说假如该参数设置为false的话，就意味着地图上的未知区域也会被认为是可以自由移动的区域，这样在进行全局路径规划时，可以把一些未探索的未知区域也来参与到路径规划，如果你需要这样的话就将该参数设置为false。不过一般情况未探索的区域不应该当作可以自由移动的区域，因此一般将该参数设置为true;
* obstacle\_range:设置机器人检测障碍物的最大范围，意思是说超过该范围的障碍物，并不进行检测，只有靠近到该范围内才把该障碍物当作影响路径规划和移动的障碍物;
* raytrace\_range:在机器人移动过程中，实时清除代价地图上的障碍物的最大范围，更新可自由移动的空间数据。假如设置该值为3米，那么就意味着在3米内的障碍物，本来开始时是有的，但是本次检测却没有了，那么就需要在代价地图上来更新，将旧障碍物的空间标记为可以自由移动的空间。
* observation\_sources:设置导航中所使用的传感器，这里可以用逗号形式来区分开很多个传感器，例如激光雷达，[碰撞传感器](https://www.baidu.com/s?wd=%E7%A2%B0%E6%92%9E%E4%BC%A0%E6%84%9F%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，超声波传感器等，我们这里只设置了激光雷达;
* laser\_scan\_sensor:添加的激光雷达传感器
* sensor\_frame:激光雷达传感器的坐标系名称;
* data\_type:激光雷达数据类型;
* topic:该激光雷达发布的话题名;
* marking:是否可以使用该传感器来标记障碍物;
* clearing:是否可以使用该传感器来清除障碍物标记为自由空间;
* inflation\_layer:膨胀层，用于在障碍物外标记一层危险区域，在路径规划时需要避开该危险区域
* enabled:是否启用该层;
* cost\_scaling\_factor:膨胀过程中应用到代价值的比例因子，代价地图中到实际障碍物距离在内切圆半径到膨胀半径之间的所有cell可以使用如下公式来计算膨胀代价：exp(-1.0 \* cost\_scaling\_factor \* (distance\_from\_obstacle – inscribed\_radius)) \* (costmap\_2d::INSCRIBED\_INFLATED\_OBSTACLE – 1),公式中costmap\_2d::INSCRIBED\_INFLATED\_OBSTACLE目前指定为254，注意： 由于在公式中cost\_scaling\_factor被乘了一个负数，所以增大比例因子反而会降低代价
* inflation\_radius:膨胀半径，膨胀层会把障碍物代价膨胀直到该半径为止，一般将该值设置为机器人底盘的直径大小。
* Static\_layer:静态地图层，即SLAM中构建的地图层
* enabled:是否启用该地图层;

**配置global\_costmap\_params.yaml**

* 全局代价地图是作为进行全局路径规划时的参考，我们需要在config目录中，创建global\_costmap\_params.yaml文件，该文件是为全局代价地图配置的参数，具体配置的参数如下：

|  |
| --- |
| Description:  #  全局代价地图参数配置文件,各参数的意义如下：  #  global\_frame:在全局代价地图中的全局坐标系;  #  robot\_base\_frame:机器人的基坐标系;  #  #History:  #  20180613: initial this file.  global\_costmap:    global\_frame: /map    robot\_base\_frame: /robot0    update\_frequency: 0.5    static\_map: true    rolling\_window: false    transform\_tolerance: 1.0    plugins:      - {name: static\_layer,    type: "costmap\_2d::StaticLayer"}      - {name: obstacle\_layer,  type: "costmap\_2d::ObstacleLayer"}      - {name: inflation\_layer, type: "costmap\_2d::InflationLayer"} |

* global\_frame:全局代价地图需要在哪个坐标系下运行;
* robot\_base\_frame:在全局代价地图中[机器人本体](https://www.baidu.com/s?wd=%E6%9C%BA%E5%99%A8%E4%BA%BA%E6%9C%AC%E4%BD%93&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)的基坐标系，就是机器人上的根坐标系。通过global\_frame和robot\_base\_frame就可以计算两个坐标系之间的变换，得知机器人在全局坐标系中的坐标了。
* update\_frequency:全局代价地图更新频率，一般全局代价地图更新频率设置的比较小;
* static\_map:配置是否使用map\_server提供的地图来初始化，一般全局地图都是静态的，需要设置为true;
* rolling\_window:是否在机器人移动过程中需要滚动窗口，始终保持机器人在当前窗口中心位置;
* transform\_tolerance:坐标系间的转换可以忍受的最大延时;
* plugins:在global\_costmap中使用下面三个插件来融合三个不同图层，分别是static\_layer、obstacle\_layer和inflation\_layer，合成一个master\_layer来进行全局路径规划。

**配置local\_costmap\_params.yaml**

局部代价地图配置参数所建立的地图主要是为局部路径规划所使用，我们可以在config目录下，创建local\_costmap\_params.yaml文件，完整内容如下：

|  |
| --- |
| #Description:  #  本地代价地图需要配置的参数，各参数意义如下:  #  global\_frame:在本地代价地图中的全局坐标系;  #  robot\_base\_frame:机器人本体的基坐标系;  #  #History:  #  20180613: initial this file.    local\_costmap:    global\_frame: /map\_static    robot\_base\_frame: /robot0    update\_frequency: 5.0    publish\_frequency: 3.0    static\_map: false    rolling\_window: true    width: 4.0    height: 4.0    resolution: 0.05    transform\_tolerance: 0.5    plugins:      - {name: obstacle\_layer,  type: "costmap\_2d::ObstacleLayer"}      - {name: inflation\_layer, type: "costmap\_2d::InflationLayer"} |

* global\_frame:在局部代价地图中的全局坐标系，一般需要设置为odom\_frame，但是由于stdr没有这个坐标系，我就拿/map\_static来代替了;
* robot\_base\_frame:机器人本体的基坐标系;
* update\_frequency:局部代价地图的更新频率;
* publish\_frequency:局部代价地图的发布频率;
* static\_map:局部代价地图一般不设置为静态地图，因为需要检测是否在机器人附近有新增的动态障碍物;
* rolling\_window:使用滚动窗口，始终保持机器人在当前局部地图的中心位置;
* width:滚动窗口的宽度，单位是米;
* height:滚动窗口的高度，单位是米;
* resolution:地图的分辨率，该分辨率可以从加载的地图相对应的配置文件中获取到;
* transform\_tolerance:局部代价地图中的坐标系之间转换的最大可忍受延时;
* plugins:在局部代价地图中，不需要静态地图层，因为我们使用滚动窗口来不断的扫描障碍物，所以就需要融合两层地图（inflation\_layer和obstacle\_layer）即可，融合后的地图用于进行局部路径规划;