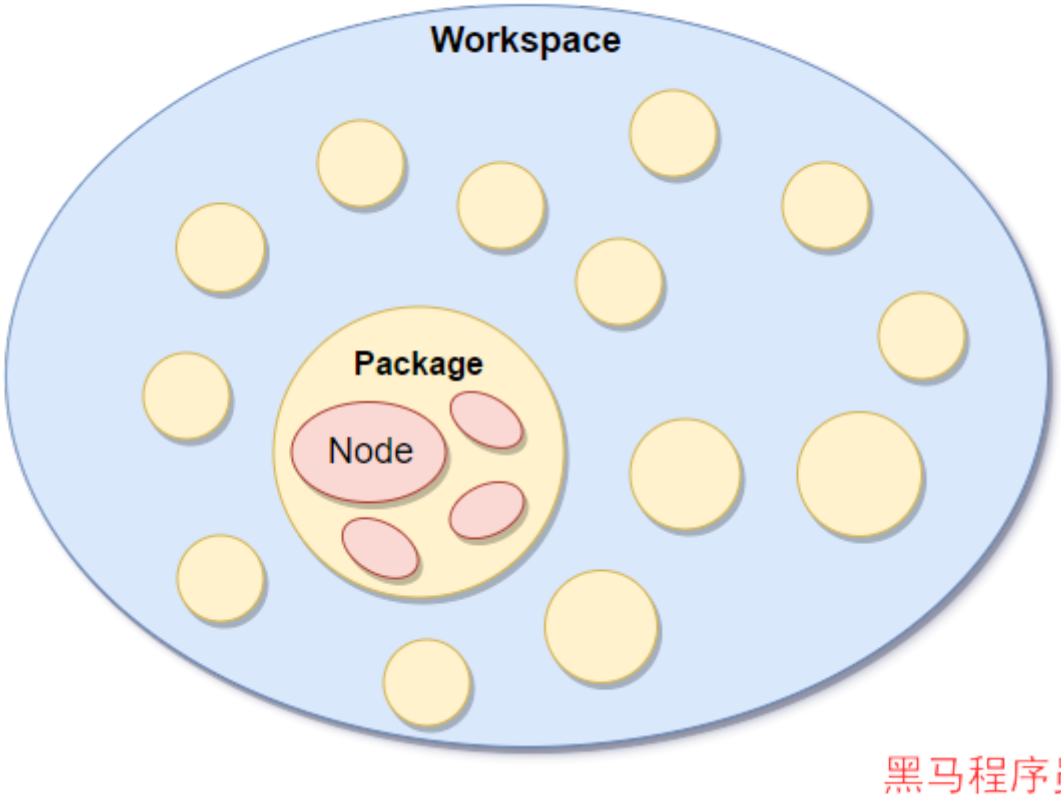
工作目录结构

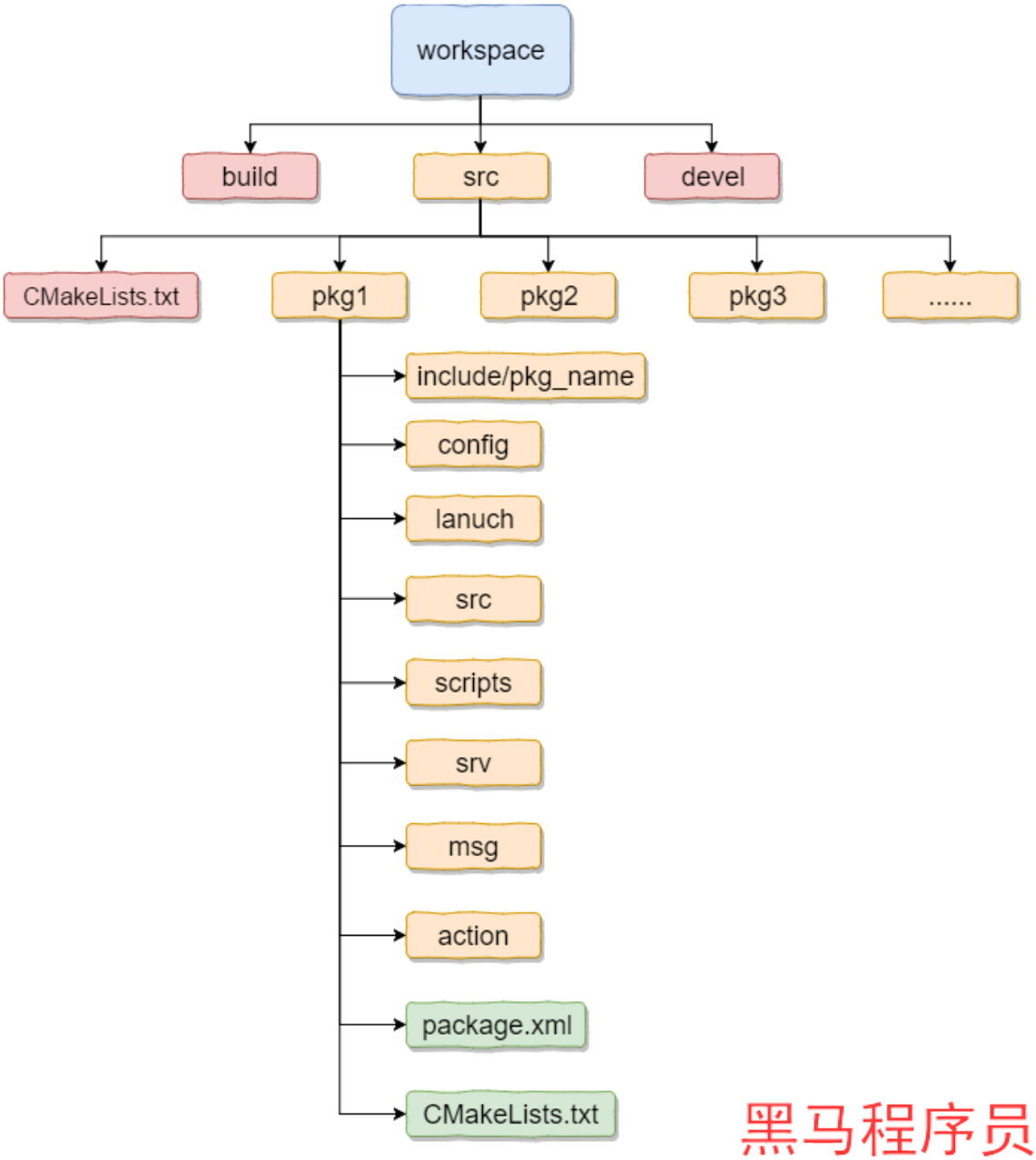


`Workspace`，`Package`，`Node`是工程结构中的几个关键词，也是核心概念。以上视图我们初步的认知他们的包含关系。

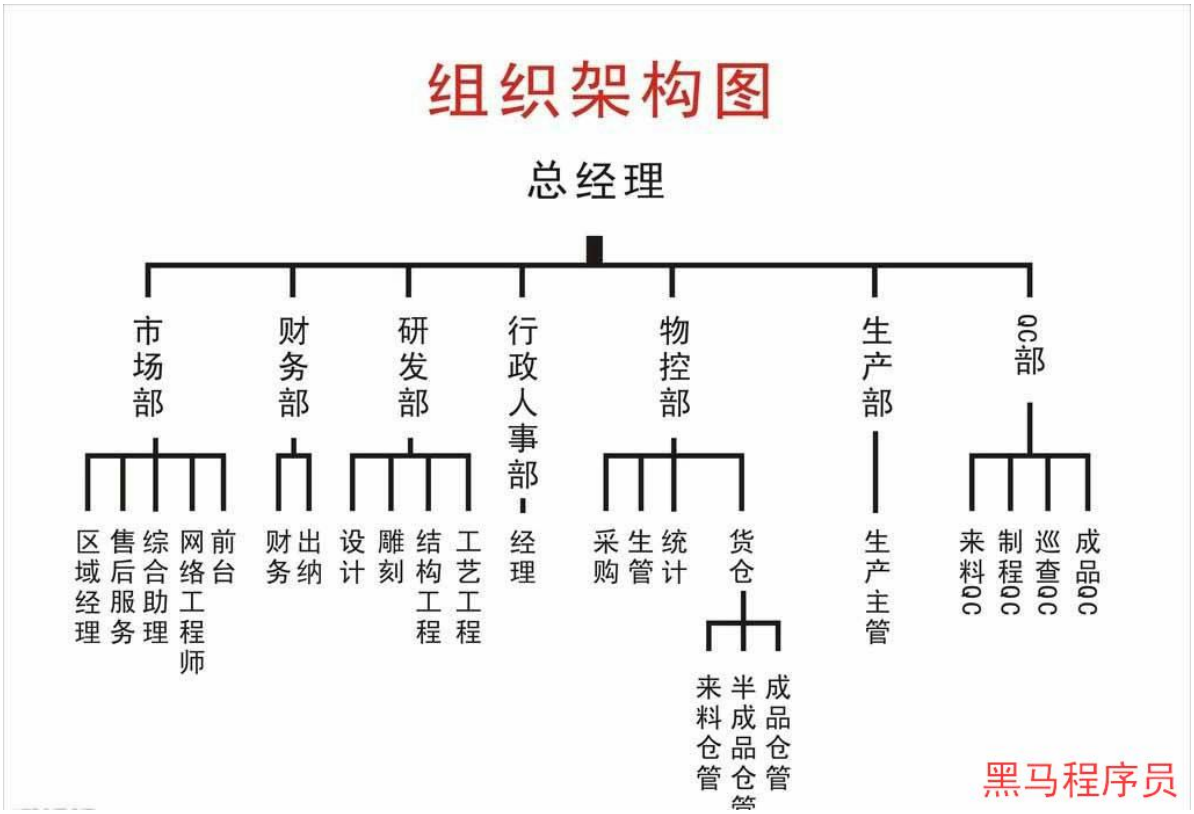
标准的`Workspace`工作目录结构如下：

```
workspace
├── build
├── devel
│   ├── setup.bash
└── src
    ├── CMakeLists.txt
    ├── pkg1
    │   ├── CMakeLists.txt
    │   ├── include
    │   ├── package.xml
    │   └── src
    ├── pkg2
    │   ├── CMakeLists.txt
    │   ├── include
    │   ├── package.xml
    │   └── src
    └── pkg3
        ├── CMakeLists.txt
        ├── include
        ├── package.xml
        └── src
```

完整的机构示意图如下：

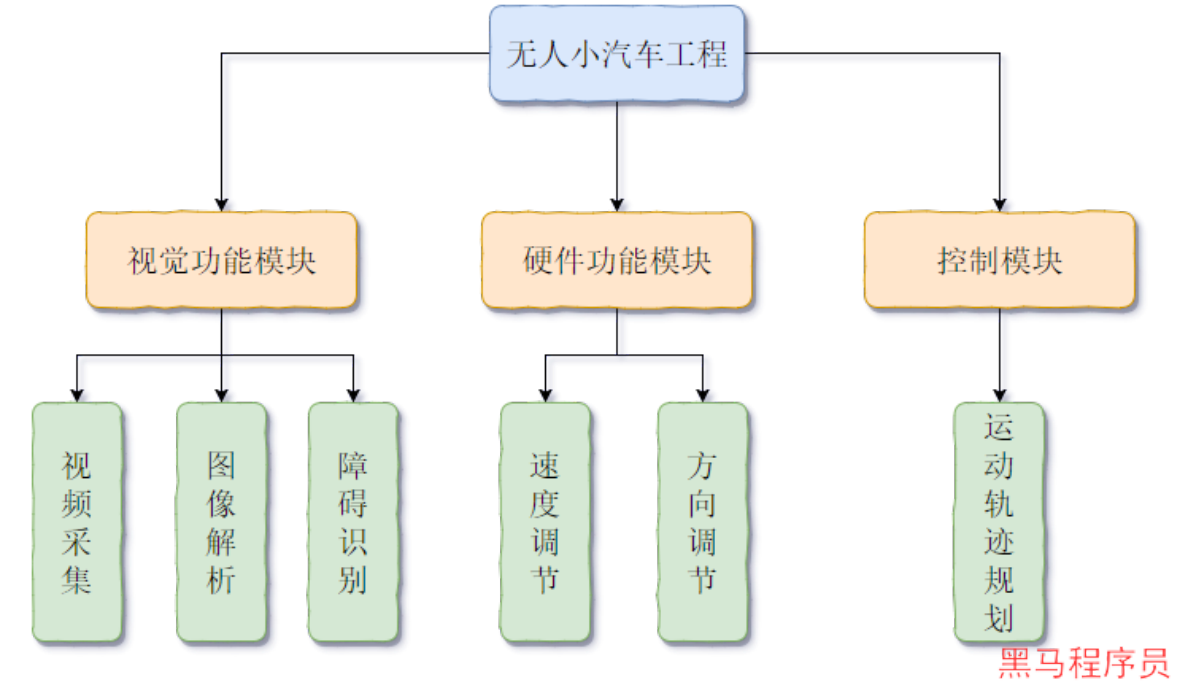

workspace
build
src
devel
CMakeLists.txt
pkg1
pkg2
pkg3
.....
include/pkg_name
config
lanuch
src
scripts
srv
msg
action
package.xml
CMakeLists.txt
黑马程序员

工作目录理解



!!!tip
生活中，企业通过部门管理员工。

```
人多瞎胡乱，鸡多不下蛋，一旦项目变大，组织结构和规范就变的非常重要。
```



我们基于ros开发一个无人驾驶的小车，代码结构按照package划分，可以划分如下

- camera视觉包： 负责视频的采集，图像解析，障碍识别等
- hardware硬件包：负责控制小车，硬件的加速减速，方向移动
- motion控制包： 负责用来规划计算 运动轨迹和如何运动

!!!tip
　按照公司结构来去类比，无人小汽车相当于一个公司，下面的模块相当于一个部门，模块下的功能相当于部门里干活的人。

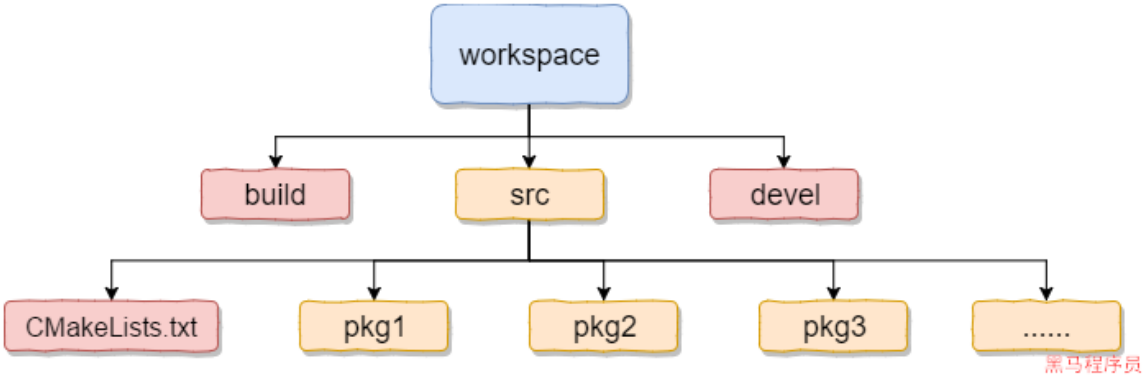
```
按照ROS的项目结构来划分，无人小汽车工程就是一个`workspace`，视觉功能模块就是一个`package`，模块下的视频采集功能就是一个`Node`。

ros的这种结构划分，和公司结构划分是一个道理。功能多，通过只能进行管理划分，规范开发，让开发效率提升，解决一些耦合。
```

工作目录说明

工作空间workspace

我们在开发一个ROS项目的时候，是以工作空间来代表一个项目的。

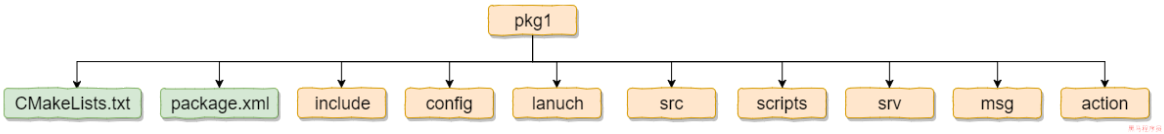


- workspace： 工作空间
- build：ros编译打包的结果产出目录。**我们不需要对这个文件夹做任何编辑操作，属于自动生成。**
- devel: 开发所需要的目录
- src：存放package的目录
- CMakeLists.txt: 整个工作空间编译的脚本。此文件我们通常不用去做修改操作。

工作单元package

一个项目中可以创建多个工作单元，这个工作单元，我们称之为package。

package的文件组成结构为以下：



- pkg1： package的名称，开发过程中根据自己实际情况进行创建设定。
- CMakeLists.txt: 当前package的编译脚本。通常需要为c++代码添加编译时的依赖，执行等操作。
- package.xml: package相关信息。通常添加一些ros库的支持
- include文件夹: 存放c++ 头文件的
- config文件夹：存放参数配置文件，格式为yaml
- launch文件夹：存放.launch文件的。
- src：c++源代码
- scripts：python源代码
- srv：存放定义的service

- msg: 存放自定义的消息协议
- action： 存放自定义的action