Roslaunch 相关内容整理

tongzhuodenilove@163.com

Roslaunch

1. 概述

该 roslaunch 包中包含 roslaunch 工具，（用来读取 roslaunch 中.launch/XML 文件格式）。也包含了各种其他支持的工具来帮助你使用这些文件。许多 ROS 包都有 "launch files",可以运行下面指令:

```
$ roslaunch package_name file.launch
```

这些启动文件通常启动提供一些聚合功能包的系列节点。了解更多有关的主要 roslaunch 工具和其他命令行工具，请参阅：

- roslaunch Command-line Usage

roslaunch 使用 XML 文件描述应该运行的节点，参数的设置，以及其他启动一系列节点的属性。规范的 XML 格式，请参见：

- roslaunch .launch/XML format

roslaunch 是为了适应 ROS 复杂的组合结构。了解 roslaunch 结构会使你更好的了解如何构建你的 .launch 文件文件以及更好的调试远程和本地的启动。

- roslaunch Architecture

一个使用 Python API 的简单例子，可以在这里找到：

- roslaunch/API Usage

roscore

roscore 是启动 ROS 系统核心的 roslaunch 的专业化工具。有关更多信息，请参见 roscore 文件。

Stability/Roadmap

启动文件语法本身是稳定的，并将尽一切努力来提供新功能的向后兼容性。

roslaunch 代码的 API 很不稳定，不能直接使用。为了支持正在计划中的新功能，对可编程的 API 可能需要做出大量的，不兼容的变化。

正计划的 roslaunch 有许多新的特点，包括在启动文件语法，用于启动文件更有效的 GUI 工具，互联网 API，单独或很多启动文件之间更好的协调等。

教程

1) Roslaunch tips for large projects

本教程介绍了写作的大项目 roslaunch 文件的一些提示。重点是如何构建启动文件以便它们可以在不同的情况下尽可能的重用。我们将使用 2dnav_pr2 包作为一个案例来研究。

2) How to Roslaunch Nodes in Valgrind or GDB

调试 roscpp nodes 时用 roslaunch 启动，你可能希望在调试程序如 gdb 或 valgrind 中启动那个节点，这样就很简单 。

2. Roslaunch / XML

1．计算次序

roslaunch 与 XML 文件单一传递。include 是按照深度优先遍历的顺序处理。标签 tag 进行串行处理并且最后的设置有效。因此，若一个参数被多次设置，最后指定的值将被使用。

依靠重写覆盖的行为是不可取的。因为没有谁能保证重写是正确的（如，inckude 文件中参数名改变了）。相反，比较推荐的是使用$(arg)/<arg>设置来进行重写行为。

2．替代参数（substitution args：置换符）

Roslaunch tag 属性可使用置换符，这要在启动节点之前解决。目前支持的置换参数如下：

```
1). $(env NVIRONMENT_VARIABLE)
```

替代当前的环境变量的值。如果环境变量没有设置，启动将失败。该值不能由< env >标签重写。

```
2). $(optenv ENVIRONMENT_VARIABLE) $(optenv ENVIRONMENT_VARIABLE
default_value)
```

如果设置了，则替代一个环境变量值。如果默认值 default_value 提供了，环境变量没有设置时将使用默认值。如果没有默认值，将使用空字符串。default_value 可以是用空格分开的多个单词。 例子:

```
<param name="foo" value="$(optenv NUM_CPUS 1)" />
<param name="foo" value="$(optenv CONFIG_PATH /home/wsh/ros_workspace)"/>
<param name="foo" value="$(optenv VARIABLE ros rocks)" />
```

```
3). $(find pkg)
```

e.g. $(find rospy)/manifest.xml. 指定包的相对路径。文件系统到包目录的路径将被内联的替换。由于硬件编码抑制了启动配置的可移植性，我们鼓励使用包相对路径。本地文件系统公约的前后削减问题得到解决（Forward and backwards slashes will be resolved to the local filesystem convention. ）

```
4). $(anon name)
```

e.g. $(anon rviz-1) 产生基于名称的匿名 ID。名字本身是一个独特的标识符：不同的 $(anon foo)用法将创建相同的"匿名"的名字。主要用于"节点名称"属性中以创建匿名节点。ROS 要求每个节点都有唯一的名字。例如：

```
<node name="$(anon foo)" pkg="rospy_tutorials" type="talker.py" />
<node name="$(anon foo)" pkg="rospy_tutorials" type="talker.py" />
```

如果两个节点有一样的名字将会发生错误。

```
5) .$(arg foo)
```

$(arg foo) 计算由<arg>标签指定的值。在声明 arg 的同一启动文件中必须有对应的 <arg>标签。

例如:

```
<param name="foo" value="$(arg my_foo)" />
```

将指定 *my_foo* 到 *foo* 参数.

另一个例子:

```
<node name="add_two_ints_server" pkg="beginner_tutorials"
type="add_two_ints_server" />

<node name="add_two_ints_client" pkg="beginner_tutorials"
type="add_two_ints_client" args="$(arg a) $(arg b)" />
```

将会从 <add_two_ints>例子中启动 server 和 client, 作为参数值 a 和传递。产生的启动项目能用如下语

句调用：

```
roslaunch beginner_tutorials launch_file.launch a:=1 b:=5
```

置换参数目前在本地机器上已经解决。换句话说，环境变量和 ROS 包的路径能在当前的环境中设置，甚至远程启动的过程也可以。

3． if 和 unless 属性

所有标签 tags 都支持 `if` 和 `unless` 属性（基于计算的值包含或者排除一个 tag）。"1"和"true"是值。"0"和"false"假值。其它值是错的。

```
if=value (optional)            如果 value 值为 true，包括标签和内容。

unless=value (optional)        除非 value 为真，包含标签和内容.
```

例子：

```
<group if="$(arg foo)">
  <!-- stuff that will only be evaluated if foo is true -->
</group>
<param name="foo" value="bar" unless="$(arg foo)" />
```

4． 相关标签

<launch> <node> <machine> <include> <remap> <env> <param> <rosparam> <group> <test> <arg>

5 .例子 .launch XML 配置文件

注：按照惯例，roslaunch XML 文件的扩展名为.launch， 如 example.launch。

5.1 最小例子（Minimal Example）

下面的示例是一个最小的启动配置脚本。它启动了一个节点'talker'，位"rospy_tutorials"包中。这个节点在本地机使用当前配置的 ROS 环境启动（如 ros_root）。

```
<launch>
  <node name="talker" pkg="rospy_tutorials" type="talker" />
</launch>
```

5.2 一个稍微复杂的例子

```
<launch>
  <!-- local machine already has a definition by default.
       This tag overrides the default definition with
       specific ROS_ROOT and ROS_PACKAGE_PATH values -->
  <machine name="local_alt" address="localhost" default="true" ros-
root="/u/user/ros/ros/" ros-package-path="/u/user/ros/ros-pkg" />
  <!-- a basic listener node -->
  <node name="listener-1" pkg="rospy_tutorials" type="listener" />
  <!-- pass args to the listener node -->
  <node name="listener-2" pkg="rospy_tutorials" type="listener" args="-foo
arg2" />
  <!-- a respawn-able listener node -->
```

```
  <node name="listener-3" pkg="rospy_tutorials" type="listener"
respawn="true" />

  <!-- start listener node in the 'wg1' namespace -->

  <node ns="wg1" name="listener-wg1" pkg="rospy_tutorials" type="listener"
respawn="true" />

  <!-- start a group of nodes in the 'wg2' namespace -->

  <group ns="wg2">

    <!-- remap applies to all future statements in this scope. -->

    <remap from="chatter" to="hello"/>

    <node pkg="rospy_tutorials" type="listener" name="listener" args="--
test" respawn="true" />

    <node pkg="rospy_tutorials" type="talker" name="talker">

      <!-- set a private parameter for the node -->

      <param name="talker_1_param" value="a value" />

      <!-- nodes can have their own remap args -->

      <remap from="chatter" to="hello-1"/>

      <!-- you can set environment variables for a node -->

      <env name="ENV_EXAMPLE" value="some value" />

    </node>

  </group>

</launch>
```

5.3 设置参数

也可以在参数服务器上设置参数。在节点启动之前，这些参数被存储在服务器上。如果值是明确的，可以省略 type 类型属性。支持的类型是 str, int, double, bool。也可以不使用 textfile 或 binfile 属性来指定一个文件目录。

例如:

```
<launch>
  <param name="somestring1" value="bar" />
  <!-- force to string instead of integer -->
  <param name="somestring2" value="10" type="str" />

  <param name="someinteger1" value="1" type="int" />
  <param name="someinteger2" value="2" />

  <param name="somefloat1" value="3.14159" type="double" />
  <param name="somefloat2" value="3.0" />

  <!-- you can set parameters in child namespaces -->
  <param name="wg/childparam" value="a child namespace parameter" />

  <!-- upload the contents of a file to the server -->
  <param name="configfile" textfile="$(find roslaunch)/example.xml" />
  <!-- upload the contents of a file as base64 binary to the server -->
  <param name="binaryfile" binfile="$(find roslaunch)/example.xml" />

</launch>
```

3. Roslaunch/XML/launch

<launch> 标签

<launch> tag 是所有 roslaunch 文件的根元素. 它的唯一目的是作为其他元素的容器。

1. 属性

```
deprecated="deprecation message"
```

ROS 1.1: 警告用户 roslaunch 文件不宜使用.

2. 元素

<node>	启动一个节点.
<param>	设置参数服务器上的参数
<remap>	声明一个名称的映射
<machine>	声明启动要使用的机器.
<rosparam>	使用 rosparam 文件设置启动要用的 ROS 参数
<include>	包含 roslaunch 文件.
<env>	制定启动节点的环境变量
<test>	启动一个测试节点 see rostest).
<arg>	声明参数
<group>	共享一个命名空间或映射的封闭的元素组。

4. Roslaunch / XML/ node

1. <node> 标签

<node> 标签指定你想启动的节点。这是 roslaunch 标签中最常见的标签，支持最重要的特性：bringing up and taking down nodes.

roslaunch 不提供节点开始的顺序保证。 这是特意的：没有办法知道哪个节点完全初始化了，所以启动代码必须在启动顺序上鲁棒性比较强。

roslaunch 教程页包含了一些如何充分利用 <node> 标签（如配置一个在 GDB 中启动的节点）的例子。

1.1 例子

```
<node name="listener1" pkg="rospy_tutorials" type="listener.py" args="--
test" respawn="true" />
```

使用命令行参数--test 包 rospy_tutorials 生成的可执行文件 listener.py 来启动 "listener1" 节点。如果节点死了，自动重新加载。.

```
<node name="bar1" pkg="foo_pkg" type="bar" args="$(find
baz_pkg)/resources/map.pgm" />
```

从 foo_pkg 包中启动 bar（障碍）节点.这个例子使用了替代参数来方便的的传递 baz_pkg/resources/map.pgm.

1.2 属性

```
pkg="mypackage"      节点的包.
type="nodetype"      节点类型。必须有相应的同名的可执行文件
name="nodename"        节点名称. 注：名字不能包含命名空间，使用 ns 属性代替.
args="arg1 arg2 arg3" (optional)    传递到节点的参数.
machine="machine-name" (optional)   启动节点的指定机器.
respawn="true" (optional)              如果节点退出，自动重启
respawn_delay="30" (optional, default 0)    ROS indigo 的新特性
```

如果重启是 true，在检测到节点失败之后，试图重启之前，等待 respawn_delay 秒。

```
required="true" (optional)
```

ROS 0.10: 如果节点死掉了，杀死掉全部的 roslaunch.

```
ns="foo" (optional)                                    在'foo'命名空间开启节点。
clear_params="true|false" (optional)    在启动之前，删除节点私有命名空间的所有参数
output="log|screen" (optional)
```

如果是'screen'，节点中的 stdout/stderr 发送到屏幕； 如果是'log'，将发送 stdout/stderr 到 $ROS_HOME/log 中的 log 文件。默认的是'log'。.

```
cwd="ROS_HOME|node" (optional)
```

如果是'node'，节点的工作目录将发送到和可执行的节点相同的目录。在 C Turtle 中，默认的是 'ROS_HOME'。在 Box Turtle (ROS 1.0.x), 默认的是'ros-root'。'ros-root' 在 C Turtle 中不能使用。

```
launch-prefix="prefix arguments" (optional)
```

Command/arguments 预先放在节点的 launch 参数的前端。这是一个强大的功能，使您能够使能 gdb, valgrind, xterm, nice,或其他方便的工具。请看 How to Roslaunch Nodes in Valgrind or GDB 的例子。

1.3 元素

你可以在 <node>标签中，看到下列 XML 标签：

- <env> <remap> <rosparam> <param>

5. Roslaunch/XML/param

1. <param> 标签

<param>标签定义了在参数服务器上设置的参数。您可以指定一个 textfile, binfile 或者 command 属性去设置一个参数的值。<param>标签可以放在<node>标签中，在这种情况下，参数被当成了私有参数。

你也可以在<param>标签中使用~param 语法(see ROS names)设置一组节点的私有参数。声明的参数会被设置为在同一范围内之后的<node>标签中的局部参数。

1.1 属性

```
name="namespace/name"
```

参数名。参数名中可以包含命名空间，但指定的全局名称应该避免。

```
value="value" (optional)          定义了参数的值。如果 uble|bool" (optional)
```

指定参数的类型。如果你不指定类型，roslaunch 会自动确定类型。基本的规则如下：
带省略该属性，则必须指定 binfile, textfile 或 command。

- ✧ type="str|int|do 有'.'的数值是浮点数，否则是整数
- ✧ "true" 和 "false" bool 值(不区分大小写).
- ✧ 其他值是字符串

```
textfile="$(find pkg-name)/path/file.txt" (optional)
```

该文件的内容以字符串形式读取和存储。该文件必须是本地可访问的，尽管强烈建议使用相对包 $(find)/file.txt 指定位置。

```
binfile="$(find pkg-name)/path/file" (optional)
```

该文件的内容以 base64 编码的 XML-RPC 二进制对象读取和存储。。该文件必须是本地可访问的，尽管强烈建议使用相对包$(find)/file.txt 指定位置。

```
command="$(find pkg-name)/exe '$(find pkg-name)/arg.txt'" (optional)
```

命令的输出以字符串形式读取存储。强烈建议使用相对包$(find)/file.txt 指定文件的参数。把文件参数用单引号括起来（由于 XML 疏散要求）。

1.2 例子

```
<param name="publish_frequency" type="double" value="10.0" />
```

6. Roslaunch /XML/ remap

<remap> 标签

<remap>标签允许你通过名称映射参数到 ROS 节点（通过更结构化的方式而不是直接设置节点参数属性来启动的节点）。

<remap>标签适用于在其范围内随后的所有声明(<launch>, <node> or <group>)。

了解如何映射解析了的名字，请看 Remapping Arguments 和 Names. 。

属性

- **from="original-name"** 你正映射的名称

  to="new-name" 目标名称

例子

给定一个订阅"chatter" 话题的 node，但是你仅仅有一个发布 "hello" 话题的节点。 他们的类型是一样的，你想把 hello" 话题的节点传送到想要 "chatter"话题的新节点，只需如下:

```
<remap from="chatter" to="hello"/>
```

7. Roslaunch / XML/machine

<machine> 标签

<machine> 标签声明了运行 ROS 节点的机器。如果所有节点都在本地启动，则不需此参数。它主要用于声明 SSH 和远程机器的 ROS 环境变量的设置，但你也可以用它来声明本地机器的信息。

属性

```
name="machine-name"
```

指定机器的名称。对应于<node> 标签的 machine 属性

```
address="blah.willowgarage.com"
```

网络/主机地址

```
env-loader="/opt/ros/fuerte/env.sh"
```

New in Fuerte

指定远程计算机上的环境文件。环境文件必须是一个设置了所有需要的环境变量的 shell 脚本，然后在所提供的参数执行 exec。看一个文件例子，ROS Fuerte. 版本的 env.sh。对于 env-loader 的例子请看下面：

```
default="true|false|never" (optional)
```

将本机设为指定节点的默认机器。默认设置仅适用于在同一范围内后面定义的节点。注意：如果没有默认的机器，将使用本地机器。可以设置 default="never"阻止对机器的选择，在这种情况下，机器只能显式分配。

```
user="username" (optional)
```

登陆机器 SSH 用户名. 如果不要求，可省略。

```
password="passwhat"(strongly discouraged)
```

SSH 密码. 强烈建议配置 SSH 关键字和 SSH 代理代替，这样可以使用证书登陆。

```
timeout="10.0" (optional)
```

在机器 roslaunch 之前的秒数（时间），认为机器启动失败。默认是 10 秒。当使用这个设置来允许速度较慢的连接时，需要改变这个参数通常是一种（你整体的 ROS 图会有通讯上的问题）征兆。

下面的属性在 ROS Electric 和早期版本可用：

```
ros-root="/path/to/ros/root/" (optional)
```

机器的 ROS_ROOT. ROS_ROOT 默认设置在本地环境中

```
ros-package-path="/path1:/path2" (optional)
```

机器的 ROS_PACKAGE_PATH. 默认设置是在本地环境中。

元素

只适用于 Electric 和早期版本：可以在 <machine> 标签中使用下列的 XML 标签:

<env> 在本机器上启动的所有进程中设置一个环境变量

例子

<machine> 标签的语法在 ROS Electric,很不相同，请注意下面例子的版本提示 s

基本原则(ROS Electric and Previous Only)

下面的例子主要是配置“节点“footalker”在另一个机器运行。除了重写将用到机器的 ROS_ROOT 和 ROS_PACKAGE_PATH ，在远程机器上还设置了一个 LUCKY_NUMBER 环境变量

```
<launch>
  <machine name="foo" address="foo-address" ros-root="/u/user/ros/ros/"
ros-package-path="/u/user/ros/ros-pkg" user="someone">
    <env name="LUCKY_NUMBER" value="13" />
  </machine>
  <node machine="foo" name="footalker" pkg="test_ros" type="talker.py" />
</launch>
```

使用 env-loader 基本原则(ROS Fuerte and later)

New in Fuerte

下面的例子主要是配置“节点“footalker”在另一个机器运行。使用默认 env-loader 文件（Fuerte.版本）。

```
<launch>
  <machine name="foo" address="foo-address" env-
loader="/opt/ros/fuerte/env.sh" user="someone"/>
  <node machine="foo" name="footalker" pkg="test_ros" type="talker.py" />
</launch>
```

这是一个 env-loader 脚本的例子。如果你想使用不同的环境配置，使用一个不同的 setup 文件替代 /opt/ros/fuerte/setup.sh：

```
#!/bin/sh
. /opt/ros/fuerte/setup.sh
exec "$@"
```

或者，如果你喜欢从 rosws 工作空间的源：

```
#!/usr/bin/env bash
source /home/username/path/to/workspace/setup.bash
exec "$@
```

8 . Roslaunch/ XML/ rosparam

<rosparam>标签

＜rosparam ＞标签使得（从参数服务器中装载/卸载参数的）rosparam YAML 文件能够使用。可以用来去除参数。 <rosparam>能放在<node>标签中，这种情况下，参数被当做私有参数。

删除(delete)和转储(dump)命令在命令以及（在其他任何上传到参数服务器的）参数加载之前运行。删除和转储命令按照声明的顺序运行。

加载命令被认为是"附加物（additive）"：如果声明一个字典或命名空间的参数，这些参数将被添加到命名空间声明的其他参数上。同样，加载命令可以覆盖之前声明的参数。

＜rosparam ＞标签可以涉及 YAML 文件或包含原始 YAML 文本。如果这个 YAML 文本定义了一个字典，可以省略参数属性。

属性

```
command="load|dump|delete" (optional, default=load)
```

rosparam 命令.

```
file="$(find pkg-name)/path/foo.yaml" (load or dump commands)
```

rosparam 文件名.

```
param="param-name"          Name of parameter.
ns="namespace" (optional)    Scope the parameters to the specified
namespace.
subst_value=true|false (optional)
```

在 YAML 文本中允许使用置换参数（substitution args）。

例子

```
<rosparam command="load" file="$(find rosparam)/example.yaml" />
<rosparam command="delete" param="my/param" />
<rosparam param="a_list">[1, 2, 3, 4]</rosparam>
<rosparam>
  a: 1
  b: 2
</rosparam>
```

置换使得 roslaunch 参数能够代表全部或者部分的 YAML 字符串来使用。例如：

```
<arg name="whitelist" default="[3, 2]"/>
<rosparam param="whitelist" subst_value="True">$(arg whitelist)</rosparam>
```

也可以用在 yaml 字符串中嵌入 (`$find ...`) 以及嵌入到较小程度上替换模式。

roscpp 代码中访问列表的例子:

```
XmlRpc::XmlRpcValue v;
    nh_.param("subscribed_to_nodes", v, v);
    for(int i =0; i < v.size(); i++)
    {
      node_names_.push_back(v[i]);
      std::cerr << "node_names: " << node_names_[i] << std::endl;
    }
```

9. Roslaunch/XML/include

<include> 标签

<include>标签允许将另一个 roslaunch XML 文件导入到当前文件。文档的当前范围被导入，包括 <group> 和 <remap>标签。除了 <master>标签外所有内容将被导入（ <master>标签只在顶层文件服从）。

属性

```
file="$(find pkg-name)/path/filename.xml"
```

要包含的文件名.

```
ns="foo" (optional)
```

导入相对于 'foo'命名空间的文件。

```
clear_params="true|false" (optional Default: false)
```

在启动之前删除 <include>的命名空间的所有参数。这个特性很危险，使用时必须注意 ns 必须指定。默认的是: false。

元素

<env> <arg>

10. Roslaunch/XML/env

<env>标签

<env> 标签设置启动节点的环境变量，只能在<launch>, <include>, <node> or <machine> 标签范围内使用。 在<launch> 标签内使用时，<env>标签只能在其后声明的节点中应用。

注：标签设定的值通过$(env ...)不可见，所以<env>标签不能用于参数化（parametrize）的启动文件。

属性

```
name="environment-variable-name"
```

你设置的环境变量。

```
value="environment-variable-value"
```

11. Roslaunch/XML /test

<test>标签

<test>标签在语法上类似<node> tag。他们都指定要运行的 ROS 节点，但是<test> tag 表明节点实际上是一个测试节点（*test node*）。*测试节点的更多信息，请看* rostest 文档。

例子

```
<test test-name="test_1_2" pkg="mypkg" type="test_1_2.py" time-limit="10.0"
args="--test1 --test2" />
```

使用--test1 --test2 命令行参数在 在 mypkg package 中生成的可执行文件 test_1_2.py 来启动"test_1_2"节点。如果比 10s 测试时间长，测试将被当做失败而终止。

属性

<test> 标签分享<node>的大部分正常属性，除了：：

- 没有重新加载（respawn）属性(测试节点必须终止，所以他们不能被重新加载)
- 没有输出（output）属性，因为 tests 使用他们自己的输出日志记录机制。
- 机器（machine）属性被忽略

还增加了一些如下，下面文档所示：

必要属性

name="nodename"	*Required*	节点名称。注：不能包含命名空间，使用 ns 属性代替
pkg="mypackage"	*Required*	节点包.
test-name="test_name"	*Required*	记录测试结果的测试名
type="nodetype"	*Required*	节点类型、必须与对应可执行文件有相同的名字。

可选属性

args="arg1 arg2 arg3"	*可选*	传递参数到节点
clear_params="true|false"	*可选*	在 launch 之前删除节点私有命名空间中所有参数
cwd="ROS_HOME|node"	*可选*	如果'node'，节点的工作目录可可执行文件的目录一样。在 Turtle C 中，默认的是'ROS_HOME'. 在 boxturtle (ROS 1.0.x),默认的是'ros-root'. cturtle 中不赞同使用'ros-root'。
launch-prefix="prefix arguments"	*可选*	.命令/参数预先考虑到节点的启动参数。这是一个强大的功能，可以使能 gdb, valgrind, xterm, nice,和其他手动工具。看例子：Roslaunch Nodes in Valgrind or GDB
ns="foo"	*可选*	在 'foo'命名空间开始节点
retry="0"	*可选*	在认为失败之前重复测试的次数，默认是 0。主要针对有时候预计失败的随机进程中。
time-limit="60.0"	*可选*	在测试被认为失败之前的秒数，默认 60s。

元素

你可以使用下列<test> 中的 XML 标签：

- <env> <remap> <rosparam> <param>

12. Roslaunch/ XML/ arg

<arg> 标签

New in C Turtle

<arg> 标签可以通过指定（通过命令行和<include>传递或者为更高级别文件声明的）值来创建更多能重复使用和配置的文件。*Args 不是全局的。*arg 声明针对单一启动文件，就像是一种方法中的局部参数一样。必须明确传递 arg 值到被包含的文件中，就像在方法调用中一样。

<arg> 能用以下三种方法之一调用：：

```
<arg name="foo" />
```

声明 foo 的存在。Foo 既可以作为命令行参数传递（顶层），也可以通过<include>传递（如果被包含）

```
<arg name="foo" default="1" />
```

声明有默认值的 foo。 Foo 能被覆盖（通过命令行参数传递（顶层），也可以通过<include>传递（如果被包含））。

```
<arg name="foo" value="bar" />
```

声明有常值的 foo。 Foo 不能被重写。这种用法保证了启动文件内部的参数化（而不暴露在更高层次的参数化）。

例子 传递参数到被包含的文件

```
<include file="included.launch">
  <!-- all vars that included.launch requires must be set -->
  <arg name="hoge" value="fuga" />
</include>
included.launch:
<launch>
  <!-- declare arg to be passed in -->
  <arg name="hoge" />
  <!-- read value of arg -->
  <param name="param" value="$(arg hoge)"/>
</launch>
```

通过命令行传递参数

Roslaunch 使用和 ROS 重映射参数相同的语法来指定参数值。

```
roslaunch my_file.launch my_arg:=my_value
```

属性

```
name="arg_name"
```

参数名

```
default="default value" (optional)
```

参数默认值。不能和值属性组合。

```
value="value" (optional)
```

参数值。不能和默认属性组合。

13. Roslaunch/XML/group

<group>标签

<group> 标签很容易将设置应用到一组节点上。有一个 ns 属性让人能够把一组节点推到一个单独的命名空间。也可以使用<remap>标签通过组重新映射设置。

属性

```
ns="namespace" (optional)
```

分配该组的节点到指定的命名空间。命名空间可以是全局的或相对的，虽然全局命名空间不鼓励。

```
clear_params="true|false" (optional)
```

启动前删除该组的命名空间中的所有参数。此功能是非常危险的，应谨慎使用。ns 必须指定。

元素

<group>标签相对于顶级<launch>标签，只是简单的作为其中的标签的一个容器。这意味着可以使用在<launch>标签中正常的任何标签。

<node> <param> <remap> <machine> <rosparam> <include> <env> <test> <arg>

14. Roslaunch/Commandline Tools

该 roslaunch 包带有 roslaunch 工具以及一些辅助启动 ROS 节点进程的工具。

roslaunch

Roslaunch 是开始与停止 ROS 节点的重要工具。能有一个或者更多的 .launch 文件作为参数。

Launch 语法

大多数 roslaunch 命令需要一个启动文件的名称。你可以指定启动文件的路径，也可以指定包名称以及包中的 launch 文件。例如：

```
$ roslaunch roslaunch example.launch
```

或者

```
$ roscd roslaunch
$ roslaunch example.launch
```

```
roslaunch <package-name> <launch-filename> [args]
```

在<package-name>中启动<launch-filename>，例如：

```
$ roslaunch rospy_tutorials talker_listener.launch
```

roslaunch 查找文件（匹配指定包中的文件名）并运行。

```
roslaunch <launch-file-paths...> [args]
```

通过指定相对或者绝对路径启动文件，例如：

```
$ roslaunch pr2_robot/pr2_bringup/pr2.launch
```

```
roslaunch - [args] New in Indigo
```

作为标准输入启动 roslaunch XML，例如：

```
$ rosrun package generate_launch | roslaunch - -p port
```

如果使用-p 选项启动了不同端口的 roscore，则同时需要传递一个-p 标志到 roslaunch 中，例如：

```
$ roslaunch -p 1234 package filename.launch
```

将会自动覆盖 ROS_MASTER_URI 中的端口设置

--wait

延迟启动，直到 roscore 检测到

--local

只启动本地节点。远程节点不运行。

--screen

强制所有节点输出到屏幕。对于节点调试很有用。

-v

启用详细打印。可用于跟踪 roslaunch 文件解析。

--dump-params

打印在 YAML 格式文件的启动参数。

通过 args 传递

如果正在启动要求设置的指定参数。你可以这样做：使用相同的语法重新映射参数：

```
roslaunch my_file.launch arg:=value
```

无启动（Non-launch）选项

下列选项在没有实际启动的情况下提供了启动信息。这些选项使用相同的启动文件分辨率作为通常的 roslaunch 命令。您可以指定启动文件的文件路径，也可以指定数据包名称，在包中启动文件，如：example.launch 文件在 roslaunch 包中，可以使用：

```
roslaunch --nodes roslaunch $ROS_ROOT/tools/roslaunch/example.launch
```

或

```
roslaunch --nodes roslaunch example.launch
```

```
--nodes <package-name> <launch-file>
--nodes <launch-file>
```

列出 launch 文件中的。对于找出传递到--args 的节点名很有用

```
--args <node-name> <package-name> <launch-file>
--args <node-name> <launch-file>
```

显示用 roslaunch 启动在<launch-file>文件 中的节点 <node-name>时使用的命令行参数。在调试时，需要启动某个特定节点时是很方便的。例如：

```
$ roslaunch --args my_node file.launch | bash
```

也可以使用带有置换参数的此选项（对于 bash，注意使用单引号代替双引号）：

```
$ roslaunch --args '$(anon my_node)' file.launch
```

```
--find <node-name> <package-name> <launch-file>
--find <node-name> <launch-file>
```

打印<node-name>定义所处的 launch 文件名。启动文件经常有许多包含文件（include），这使

得查找到真正的节点定义位置有点困难。例如：

```
$ roslaunch --find /included/talker roslaunch example.launch
--files <filename>
```

打印在 <filename>包含的所有文件，包含文件自身。对于转到其他命令行工具来说是很有用的。例如：

```
$ roslaunch --files foo.launch | xargs grep stuff
```

仅供内部使用的选项

roslaunch 使用的几个仅供内部使用的选项，包含 -c, -u, 和 --core.

环境变量（高级用户）

注：这部分是为高级用户使用

```
ROSLAUNCH_SSH_UNKNOWN
```

当在远程机器上的启动时，SSH 请求存储在本地 known_hosts 文件中的远程机器上的秘钥。你可以设置 roslaunch 忽略这个约束并允许在没有秘钥的情况下连接到机器。这是一个很危险的选项，因为它引入了一个安全漏洞，只有在你了解后果的情况下才能使用。

roslaunch-deps 依赖

roslaunch-deps 报告了.launch 文件依赖的 ROS 包。它也可以追踪命令编译问题，如包的 manifests 中缺少的依赖。编译一个启动文件的所有必须的包的运行的一个命令如下：

```
rosmake `roslaunch-deps file.launch`
```

用法

为了得到 manifest 文件中丢失的依赖，运行带有-w 警告选项的命令：

```
$ roslaunch-deps -w file.launch
```

为了得到更详细的输出来跟踪依赖的来源，运行有-v 选项的命令：

```
$ roslaunch-deps -v file.launch
```

roslaunch-logs 日志

roslaunch 存储在 ROS 日志目录($ROS_ROOT/log 或 $ROS_LOG_DIR)下的子目录一个特定飞运行的日志文件。在一般情况下，日志文件存储在 ros_log_dir / run_id 中，其中 run_id 唯一的 ID(与 roscore 中特定的运行有关)。

用法

roslaunch-logs 要与'cd'命令一起使用例如：

```
cd `roslaunch-logs`
```

你可以输入这个命令，快速进入节点的日志文件的目录。

15 .Metapackages 元软件包

Metapackages 是 ROS 中的专用包(和 catkin)。他们没有安装文件（除了 package.xml manifest），不包含任何测试，代码，文件，或在包中经常看见的其他项。

元软件包的使用和虚包在 debian 包领域中使用的方式一样。一个元包只是一个或多个相关的包装松散组合在一起的引用。

ROS 中大部分的元包对于 rosbuild 栈都是向后兼容的。关于元软件包的内部结构和如何创建一个元软件包的更多的信息，请看: catkin/package.xml#Metapackages

16. Roslaunch/Architecture

roslaunch 通过组合来适应复杂的 ROS 架构：先写一个简单的系统，并将其与其他简单的系统结合，使系统更复杂。在 roslaunch 中，通过以下几种机制表达：

1. `<include>`: 可以容易的包括其他.launch 文件并指定他们的命名空间使得他们的名字与你的名字不冲突。
2. `<group>`: 可以组合节点集合，给他们相同的映射名称。
3. `<machine>`别名:可以在不同的.launch 文件中将 将机器和节点的定义分开，使用别名定义在运行期间使用哪个机器。这样可以重新使用不同机器人的相同节点的定义。例如，一个 aser_assembler 运行在 'foo.willowgarage.com'，可以说它运行在 tilt-laser'机器上。这个机器的定义将会关注哪个主机是 'tilt-laser' 机器。

`roslaunch` 也包含了帮助写.launch 文件的各种工具。`<env>` 标签指定某个机器或者节点需要设置的环境变量。 `$(find pkg)` 指定相对于 ROS 包的文件路径，而不是在某个特定机器上的位置。也可以在包含标签中使用 `$(env ENVIRONMENT_VARIABLE)` 语法去加载 基于环境变量(e.g. `MACHINE_NAME`）的`.launch` 文件。

本地进程 Local processes

`roslaunch` 使用 popen 启动本地进程，使用 POSIX 信号杀死他们。 `roslaunch` 不保证节点启动的顺序（由于 ROS 结构原因，没有办法知道节点什么时候初始化完成，所以不能保证按照特定顺序启动）。

远程处理 Remote processes

远程进程与本地进程不同之处在于 `roslaunch` 是如何处理他们的。写启动文件时理解差异是很重要的。一些高级的差异包括：

1. 你的 shell 初始化脚本(`.bashrc`/`.tcshrc`/etc...) 被忽略。启动文件为了能在用户和系统之间的更加方便，launch 文件中的配置需要完全指定。这些设置包括环境变量设置，这往往是隐藏在 shell 的初始化脚本中。
2. 远程处理不能发送 stderr / stdout 到本地控制台。一般情况下，节点应该（使用 ROS 的内置的日志记录机制(i.e. rosconsole for roscpp, logout/logerr for rospy, etc...)）报告重要信息。可通过 rosout 工具查看这个重要的信息。

否则，远程启动和本地启动非常相似。事实上，为了远程启动，roslaunch 创建一个包含在远程机器上想启动的节点的新的`.launch` 文件。然后通过 SSH 远程登陆机器并使用新的`.launch` 文件启动 roslaunch 子进程。

推荐使用 SSH 秘钥，这样 roslaunch 可以在没有提供明确的用户名和密码的情况下远程登陆到机器。

如果远程启动有问题，需要确认在启动文件中指定的远程机器名能不能 SSH 登陆进去。还应该确定每个远程机器能使用他们的主机名 ping 原机。

17. Roslaunch/API 使用

该 roslaunch 包带有一个 Python API 协助 ROS 节点启动。

1. 简单用例

简单用法例子

```
import roslaunch
package = 'rqt_gui'
executable = 'rqt_gui'
node = roslaunch.core.Node(package, executable)

launch = roslaunch.scriptapi.ROSLaunch()
launch.start()
process = launch.launch(node)
print process.is_alive()
process.stop()
```

这个例子是从 rqt_gui 包中启动和停止 rqt_gui 实例的。

<node> 标签的 launch-prefix 属性使得调试 ROS 节点进程简单些，下面是可能会有用的 launch-prefixes 例子:

- launch-prefix="xterm -e gdb --args"：在一个单独的 xterm 窗口 gdb 中运行节点，手动键入 run 来开始
- launch-prefix="gdb -ex run --args"：在同一个 xterm 窗口 gdb 中运行节点，无需键入 run 来开始
- launch-prefix="valgrind"：在 valgrind 中运行节点
- launch-prefix="xterm -e"：在单独的 xterm 窗口运行节点
- launch-prefix="nice"：nice 进程降低 CPU 使用
- launch-prefix="screen -d -m gdb --args"：如果节点正在另一台机器上运行很有用；你可以 SSH 到那台机器上， screen -D -R 来看 GDB 会话（session）
- launch-prefix="xterm -e python -m pdb"：在一个单独的 xterm 窗口 pdb 运行 python 节点，手动键入 run 来开始。
- launch-prefix="/path/to/run_tmux":在一个新的 tmux 窗口运行节点；需要用如下内容创建 /path/to/run_tmux:

```
#!/bin/sh
tmux new-window "gdb --args $*"
```

18. Obtaining core dumps 获得核心转储

当进程崩溃时要获得核心转储，首先设置内核文件大小限制。为了检查限制，运行:

```
$ ulimit -a
```

```
core file size          (blocks, -c) 0          #  <-- Prevents core dumps
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 20
file size               (blocks, -f) unlimited
pending signals                 (-i) 16382
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) unlimited
virtual memory          (kbytes, -v) unlimited
 file locks                     (-x) unlimited
```

设置内核大小为无限制：

```
ulimit -c unlimited
```

现在，粉碎进程时将尝试创建核心文件。目前（2010-07-02）他们将无法创建文件，因为默认的 roslaunch 工作目录$ROS_HOME,包含了一个名为“核心”的目录。这个目录，阻止核心转储被创建。

为了允许核心转储被创建，使用进程默认的 PID 设置内核文件名。以超级用户运行下面命令:

```
echo 1 > /proc/sys/kernel/core_uses_pid
```

现在核心转储显示 $ROS_HOME/core.PID