Servicios

1. Son bloqueantes (síncronos), el programa no puede continuar hasta se reciba resultado

↳ Hacer que el robot cruce una puerta

procese info...

2. Para ver lista de servicios:

rosservice list

3. rosservice info /servicio

↳ Node ⇒ nodo que se carga en memoria y proporciona servicio.

Type ⇒ Mensaje empleado para este servicio.

paquete/mensaje

Args ⇒ Argumentos (info) que hay que pasarle al servicio para invocarlo

rossrv show paquete/mensaje.

los mensajes, son especiales, mensajes de servicio:

rossrv show paquete/mensaje

↳ Siempre tienen 2 partes:

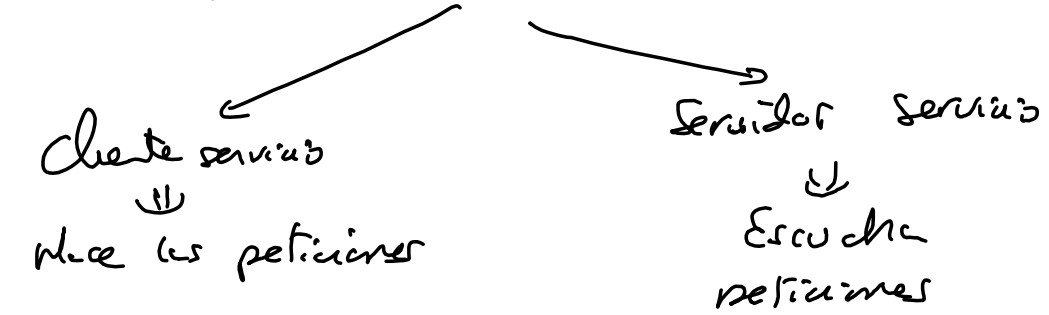
Request ⇒ string dato

Response ⇒ bool exito.

Cuando se crea un mensaje de servicio, al compilar se generan tres objetos:

MiServicio, MiServicioRequest, MiServicioResponse
↑ ↑

En un servicio siempre tenemos dos agentes implicados:



(4.) Como se invoca desde línea de comandos?

rosservice call /servicio TAB - TAB

Autocompleta la estructura del servicio, solo es necesario modificar el contenido del mensaje.

(5.) Vamos a ver cómo crear cliente servicio:

```
#!/usr/bin/env python
import rospy
from ____.srv import
     paquete
```

mensaje

Servicio, ServicioRequest

```
rospy.init_node('cliente_servicio')
rospy.wait_for_service('/servicio')
```
↳ (Esperamos a que el servidor este ejecutándose ofreciendo servicio)

```
# Se crea conexión con el servicio
servicio = rospy.ServiceProxy('/servicio', tipo de mensaje)
# Se crea un objeto de "tipo de mensaje"
peticion = MensajeRequest()
# Se rellena la petición
peticion.dato = "----"
# Se invoca al servicio a través de la
  conexión anterior
resultado = servicio(peticion)
```
↳ Esta llamada es bloqueante, el programa espera a que acabe el servicio

```
# Se muestra resultado
print resultado
```

Vamos a ver ahora como crear un servidor de servicio:

```
#! /usr/bin/env python
import rospy
from std_srvs import Empty, EmptyResponse
```

Mensajes predefinidos en empty_srv

std_srvs ↑ paquete.srv

Empty, EmptyResponse ↑ Mensajes necesarios

```
def mi_callback(request):
    print("Hola has ejecutado servicio")
    return(EmptyResponse)
```

request ↓ en este ejemplo mensaje vacio

return(EmptyResponse) ↑ devuelve mensaje vacio

```
rospy.init_node('mi_primer_servicio')
mi_servicio = rospy.Service('mi_servicio', Empty,
                            mi_callback)
rospy.spin()
```

Tipo Mensaje

mi_callback ↑ Callback (función) que se invocará cuando llegue una petición de servicio

rospy.spin() ↑ importante para que servidor quede esperando peticiones

Vemos a ver como crear mensajes de servicio personalizados

1. Creamos carpeta srv en el paquete del servidor de servicio

```
mkdir srv
```

2. Creamos y editamos fichero con detalle mensaje

```
gedit Mi_Mensaje.srv
```

```
int32 dato
---
bool exito
```

3. A continuación, y de forma similar a como hicimos con mensajes personalizados modificamos los archivos CMakeLists.txt y package.xml

Modificaciones CMakeLists.txt

I) find_package (catkin REQUIRED COMPONENTS
    rospy
    std_msgs
    geometry_msgs
    message_generation
)

II) add_service_files (
    FILES
    MiMensaje.srv
)

III)

generate_messages(
  DEPENDENCIES
  std_msgs
  geometry_msgs
)

⟸ Aquí otro tipo de mensajes necesitamos

// Nombre del paquete

IV)

catkin_package(

  CATKIN_DEPENDS rospy message_runtime

)

Si trabajo con mensajes personalizados es necesario hacer algunos cambios en programas, por ejemplo, veamos lo que sucede en servidor servicio.

```
#! /usr/bin/env python
import rospy
from mi_paquete.srv import MiMensaje, MiMensajeResponse

def mi_callback(request):
    mi_respuesta = MiMensajeResponse()
    mi_respuesta.exito = True
    print("Hola has ejecutado servicio")
    return (mi_respuesta)

rospy.init_node('mi_primer_servicio')
mi_servicio = rospy.Service('mi_servicio', Empty,
rospy.spin()
```