Expert GUI RPCについて

今井ひかる

2022/12/21

概要

- FelixボードをPCから操作するためにRPC(Remote Procedure call、遠隔手続き呼び出し)という。

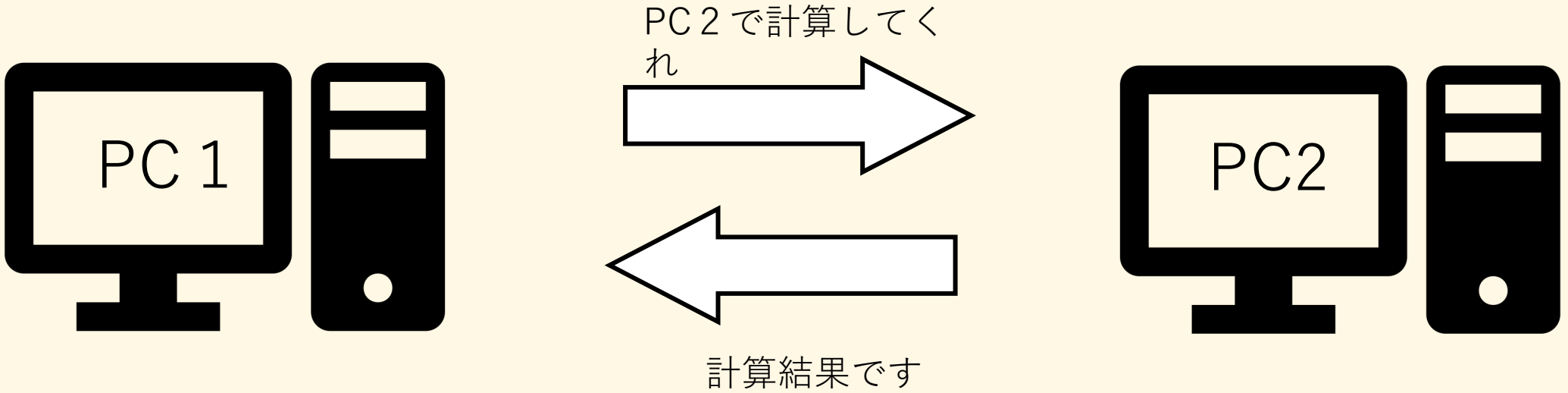
- 前回とは違うRPCを使ってみた。

前回の内容

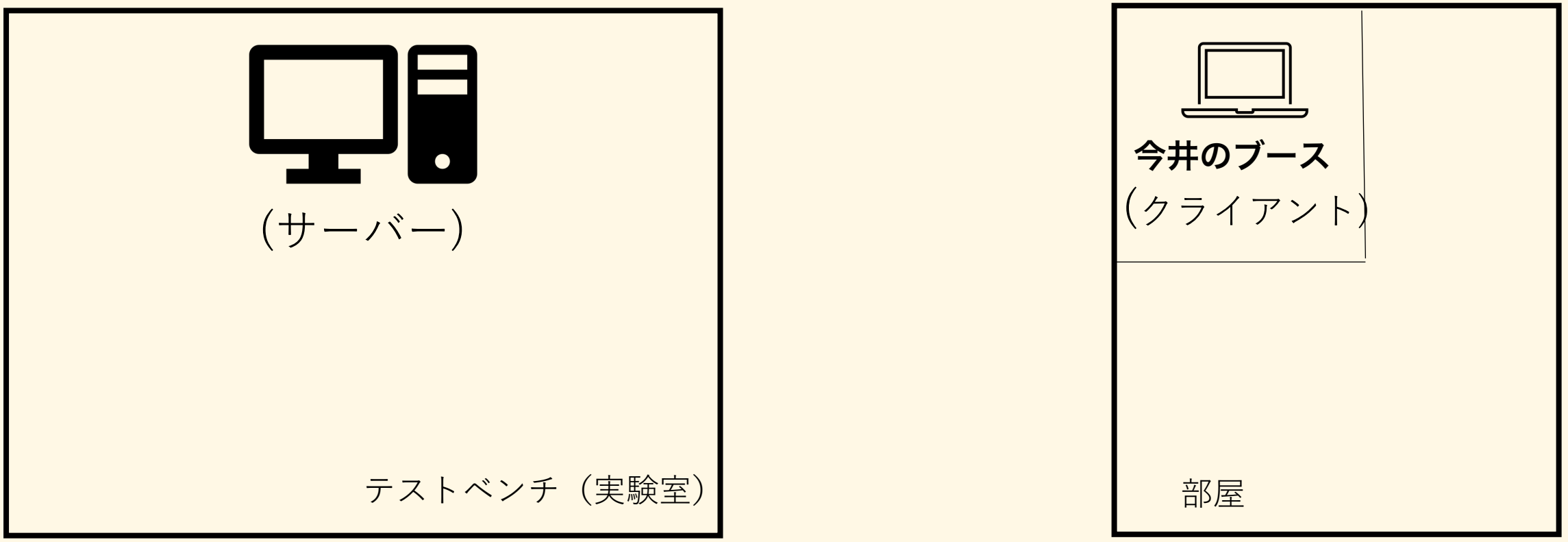
- RPC（リモートプロシージャコール）と呼ばれる技術を使う。

- RPCとは、ネットを通じて、あるPCから別のPCに処理を依頼したり、結果を返したりすること。

- PythonにRPCを簡単にできるライブラリがある。これを使ってみた。



今回試したこと @理研



物理的に離れているテストベンチのPCに命令を送り、値を返してもらう実験

準備

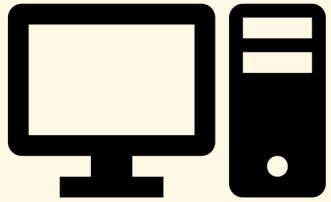
サーバー側に、左のPythonスクリプトを用意する。

```python
from xmlrpc.server import SimpleXMLRPCServer
import subprocess as sp
from Ana import Fitting_GetMean


#                         IP address     port number
with SimpleXMLRPCServer(('172.27.217.44', 8000)) as server:

    # define function which client will use

    def hello():
        return 'Hello, Pyhton!'

    def add_calc(x, y):
        return x + y

    def call_Fitting(direction):

        return Fitting_GetMean(direction)
        # PyROOT
        # return mean of gaus fitting result

    # register user function
    #                        func_val        name
    server.register_function(hello        ,"hello")
    server.register_function(add_calc     ,"add")
    server.register_function(call_Fitting,"fit")
    print("Waitting...")

    # waitting forever
    server.serve_forever()
```

あらかじめ、使う関数を定義しておく

関数を登録

rpc_server.py



（サーバー）

テストベンチ（実験室）

時間(s)	加速度(x)	加速度(y)	加速度(z)	加速度(abs)
0.0016	0.1108	0.2239	9.8649	9.8681
0.0084	0.1039	0.2147	9.8681	9.8710
0.0184	0.1061	0.2103	9.8699	9.8727
0.0284	0.1085	0.2241	9.8757	9.8789
0.0384	0.1102	0.2257	9.8643	9.8675
0.0484	0.1058	0.2200	9.8687	9.8717

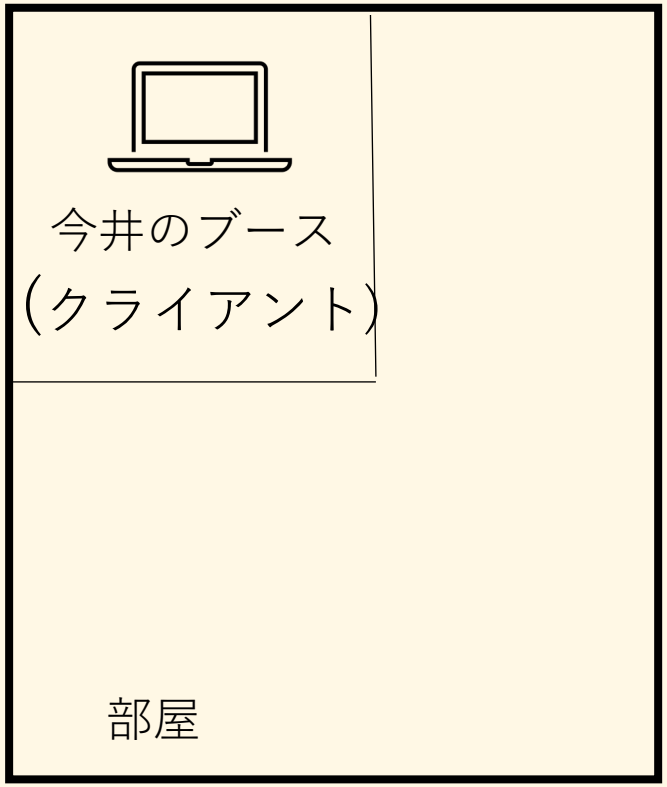
サーバにtxtデータがある。

準備

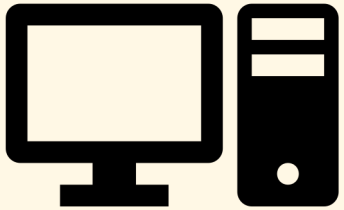
クライアント側に、左のPythonスクリプトを用意する。

```python
import xmlrpc.client

# ---- client side -----

with xmlrpc.client.ServerProxy('http://172.27.217.44:8000/') as proxy:

    # you can use function is registerd to server
    # usage --> proxy.func()

    print(proxy.hello())
    print(proxy.add(100, 200))
    print(proxy.fit("z") )
```

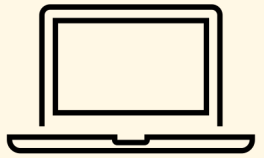
rpc_client.py



実行



サーバー側
（テストベンチPC）

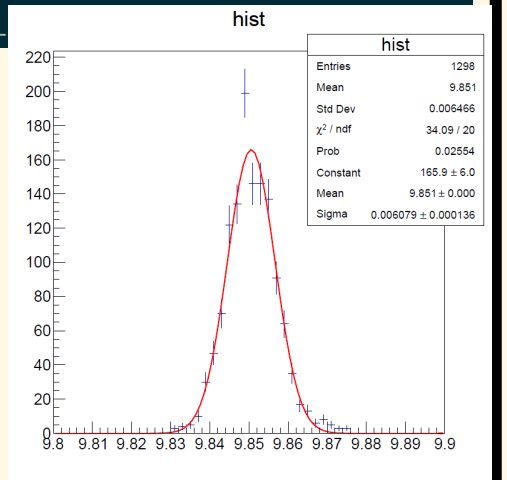


クライアント側
（今井PC）

1. $ python3 rpc_server.pyでプログラムを実行させる。
   → 準備OK

```
riken_intt@DESKTOP-5QU8C9R:/mnt/c/Users/RIKEN_INTT/Desktop/Hikaru/2022_1031$ python3 rpc_server.py
Waitting...
```

```
riken_intt@DESKTOP-5QU8C9R:/mnt/c/Users/RIKEN_INTT/Desktop/Hikaru/2022_1031$ python3 rpc_server.py
Waitting...
172.27.217.12 - - [31/Oct/2022 15:13:34] "POST / HTTP/1.1" 200 -
172.27.217.12 - - [31/Oct/2022 15:13:34] "POST / HTTP/1.1" 200 -
 FCN=7.79905 FROM MIGRAD    STATUS=CONVERGED      65 CALLS          66 TOTAL
                     EDM=7.53848e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     5.40131e+02   1.88505e+01   2.17800e-02  -6.71472e-06
   2  Mean         1.05324e-01   1.08210e-04   1.53887e-07  -1.49160e-01
   3  Sigma        3.81228e-03   8.03720e-05   7.96565e-06  -2.02617e-02
Info in <TCanvas::Print>: pdf file test.pdf has been created
172.27.217.12 - - [31/Oct/2022 15:13:35] "POST / HTTP/1.1" 200 -
```



しかし、計算をしているのは、
サーバー側

サーバー側にしかデータがない。

2. $ python3 rpc_client.pyでサーバー側に登録されている関数を使う

```
hikaru@hikaru:~$ python3 rpc_client.py
Hello, Pyhton!
300
9.850572973236224
```

```python
import xmlrpc.client

# ---- client side -----

with xmlrpc.client.ServerProxy('http://172.27.217.44:8000/') as proxy:

    # you can use function is registerd to server
    # usage --> proxy.func()

    print(proxy.hello())
    print(proxy.add(100, 200))
    print(proxy.fit("z") )
```

見かけ上、サーバー側にある関数を importして使っているように見える

RPCについて

・RPCには様々なものがある。

1. xmlrpc
2. json-rpc
3. Mprpc
4. gRPC

・sPHENIX側から指定は無い。

gRPC

A high performance, open source universal RPC framework

Learn more

Get started!

Go C++ Java Python •••

Quick start

This guide gets you started with gRPC in Python with a simple working example.

Prerequisites

- Python 3.5 or higher
- `pip` version 9.0.1 or higher

If necessary, upgrade your version of `pip`:

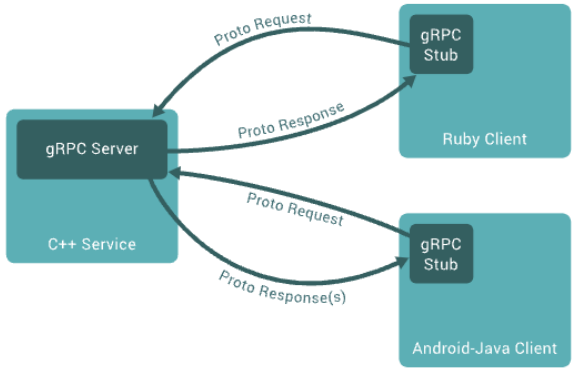
Introduction to gRPC

An introduction to gRPC and protocol buffers.

This page introduces you to gRPC and protocol buffers. gRPC can use protocol buffers as both its Interface Definition Language (**IDL**) and as its underlying message interchange format. If you're new to gRPC and/or protocol buffers, read this! If you just want to dive in and see gRPC in action first, select a language and try its **Quick start**.

Overview

In gRPC, a client application can directly call a method on a server application on a different machine as if it were a local object, making it easier for you to create distributed applications and services. As in many RPC systems, gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a gRPC server to handle client calls. On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.



Basics tutorial

A basic tutorial introduction to gRPC in Python.

This tutorial provides a basic Python programmer's introduction to working with gRPC.

By walking through this example you'll learn how to:

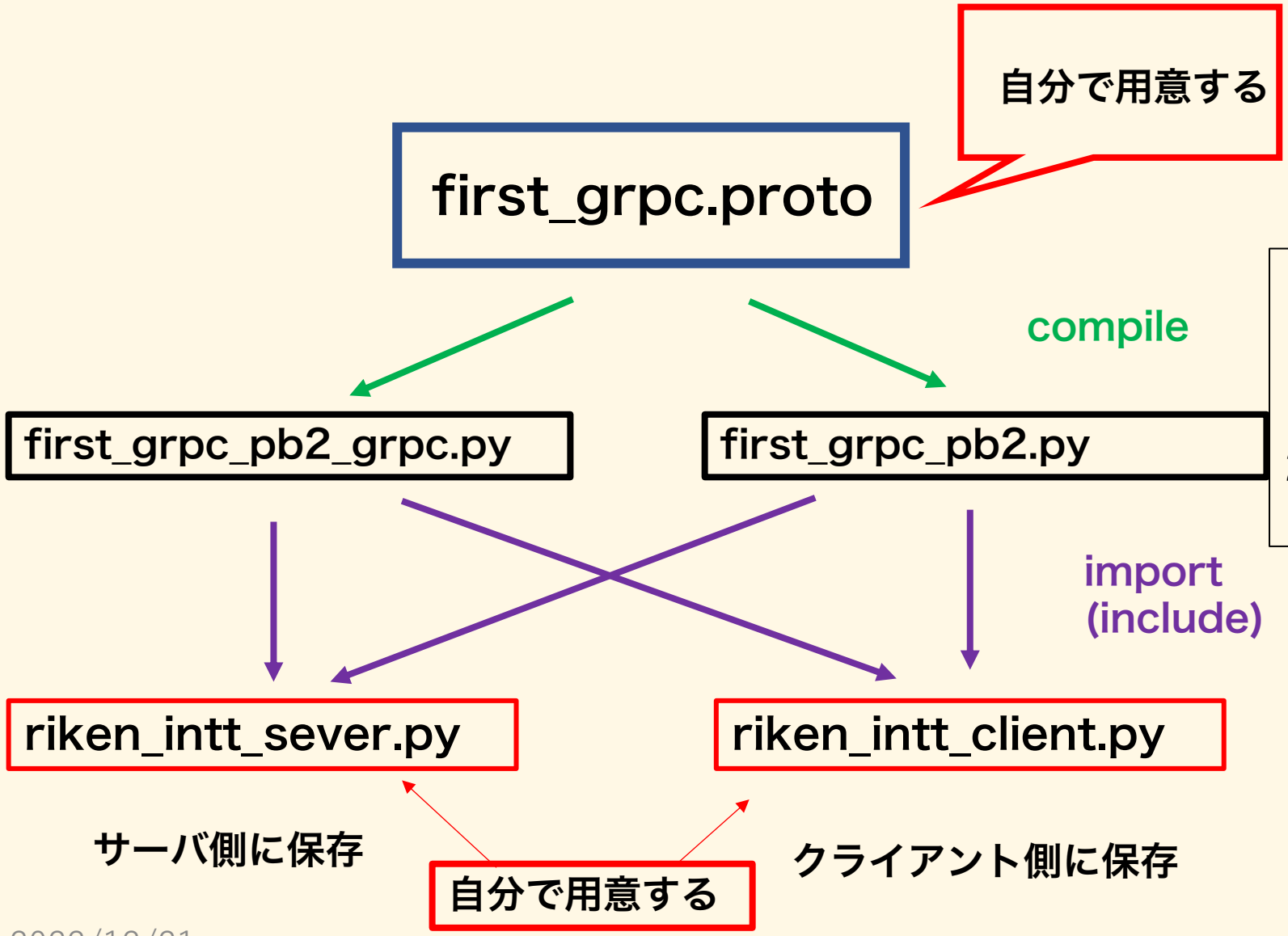
- Define a service in a .proto file.
- Generate server and client code using the protocol buffer compiler.
- Use the Python gRPC API to write a simple client and server for your service.

It assumes that you have read the Introduction to gRPC and are familiar with protocol buffers. You can find out more in the proto3 language guide and Python generated code guide.

Used by

Square NETFLIX CoreOS Cockroach LABS CISCO JUNIPER NETWORKS

gRPCを使うための手順　フロー



1. .protoファイルで
   通信のインターフェースを定義する。

2. 自動生成されたファイルを用いて通信
   を実装する。何をやりたいか実装する。

first_grpc.proto

first_grpc.proto

Users > hikaru > Desktop > INTT > test_bench > 2022_1213_gRPC > helloworld > first_grpc.proto

```
1
2
3   syntax = "proto3";
4
5   service riken_intt{
6
7   rpc Check_communication( CheckRequest ) returns ( CheckReply ){}
8   rpc fitting_for_cosmic_ray ( Fitting_Request ) returns ( Fitting_Reply) {}
9
10  }
11
12  message CheckRequest{
13
14      string request_communication_status = 1;
15
16  }
17
18  message CheckReply{
19
20      string reply_communication_status = 1;
21
22  }
23
24
25  message Fitting_Request{
26
27      string input = 1;
28  }
29
30
31  message Fitting_Reply{
32
33      float mean         = 1;
34      float mean_error   = 2;
35      float reduced_chi2 = 3;
36
37  }
```

riken_intt_sever.py

```
class riken_intt(first_grpc_pb2_grpc.riken_inttServicer):

    def Check_communication(self, request, context):
        print(request.request_communication_status)
        return first_grpc_pb2.CheckReply(reply_communication_status= " GOOD !!")
    def fitting_for_cosmic_ray(self, request, context):

        ROOT.gStyle.SetOptFit(1111)
        datadir = "/mnt/c/Users/HikaruImai/Desktop/INTT/cosmic_ray/data/cosmic_ray_per_30min/"
        flist_name = "LoopDataSet_DACscan_1129_pro.txt"
        print(request.input)
        number_of_muon_list =  call_count_cosmic_ray(datadir , flist_name )
        print( "muon = ",number_of_muon_list)
        graph  = ROOT.TGraphErrors()
        graph.SetMarkerStyle(20)

        c = ROOT.TCanvas("cc","cc",1000,1000)
        index = 0
        for num in number_of_muon_list:
            print("num = " , num)
            graph.SetPoint(index, index + 1 , num   )
            graph.SetPointError( index , 0, math.sqrt( num )     )
            index = index + 1

        pol0 = ROOT.TF1("pol0","pol0",0,1)
        pol0.SetParameters(0,600)
        graph.Fit( "pol0","Q")

        mean       = pol0.GetParameter(0)
        mean_error = pol0.GetParError(0)
        chi2       = pol0.GetChisquare()
        dof        = pol0.GetNDF()
        graph.Draw("AP")
        print(mean)
        print(mean_error)

        reduced_chi2 = chi2/dof
        print(reduced_chi2)
        c.Print("fit.pdf")
        return first_grpc_pb2.Fitting_Reply(mean = mean , mean_error=mean_error,reduced_chi2=reduced_chi2 )
```

```
4
5   service riken_intt{
6
7   rpc Check_communication( CheckRequest ) returns ( CheckReply ){}
8   rpc fitting_for_cosmic_ray ( Fitting_Request ) returns ( Fitting_Reply) {}
9
10  }
```

.proto

```
def serve():

    port = '8000'
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))

    first_grpc_pb2_grpc.add_riken_inttServicer_to_server(riken_intt(), server)
    server.add_insecure_port('[::]:'+port)
    server.start()
    print("server started")
    server.wait_for_termination()


if __name__ == '__main__':

    serve()
```

riken_intt_sever.pyで

関数の実装
サーバーを立てる？ことをする。

riken_intt_client.py

```python
import grpc
import first_grpc_pb2
import first_grpc_pb2_grpc

def run():

    server_IP_address = "172.27.217.44" # RIKEN testbench PC address
    port = "8000"
    with grpc.insecure_channel(server_IP_address+port) as channel :
        stub =  first_grpc_pb2_grpc.riken_inttStub(channel)
        response = stub.Check_communication( first_grpc_pb2.CheckRequest (request_communication_status="Hello!") )
        fitting_result = stub.fitting_for_cosmic_ray(first_grpc_pb2.Fitting_Request(input="Fitting by pol0"))
        print(response)
        print(fitting_result.mean,fitting_result.mean_error,fitting_result.reduced_chi2)

if __name__ == '__main__':
    run()

```

クライアントはサーバに問い合わせることをする

結果

```
python3 riken_intt_server.py
```

```
riken_intt@DESKTOP-5QU8C9R:/mnt/c/Users/RIKEN_INTT/Desktop/Hikaru/2022_1213/2022_1213_gRPC/helloworld$ python3 riken_intt_server.py
E1213 15:48:22.937369900     438 socket_utils_common_posix.cc:223]     check for SO_REUSEPORT: UNKNOWN:Protocol not available {syscall:"getsockopt(SO_REUSEPORT)", os_error:"Protocol not available", e
rrno:92, created_time:"2022-12-13T15:48:22.9371869+09:00"}
server started
Hello!, I am from Hikaru PC
Fitting by pol0
 ---Fitting result---
mean (server side) --> 631.4122293950709
mean error (server side) --> 4.835869945081584
reduced chi2 (server side) --> 1.2257617820416864
Info in <TCanvas::Print>: pdf file fit.pdf has been created
```

```
[hikaru@hikaru helloworld]$ python3 riken_intt_client.py
reply_communication_status: " GOOD !!"

 Mean = 631.4122314453125  +-  4.835869789123535
reduced chi2 =  1.2257617712020874
[hikaru@hikaru helloworld]$ python3 riken_intt_client.py
reply_communication_status: " GOOD !!"

 Mean = 631.4122314453125  +-  4.835869789123535
reduced chi2 =  1.2257617712020874
```