Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

Кафедра ВПМ

Отчет о лабораторной работе № 1

«Сервис-ориентированная архитектура вычислений

на основе протокола XML-RPC»

по дисциплине

«Распределенные системы обработки информации»

Выполнил:

Студент группы 943

Серов А.П.

Проверил:

доц. Князьков П.А.

Рязань 2023

# Цель работы

Получение практических знаний о сервис-ориентированной архитектуре вычислений на основе протокола XML-RPC.

# Задание

1. Загрузить дистрибутив Anaconda3 2022/Python 3.9 version;
2. Изучить программы файлов тетрадей клиента xmlrpc\_client.ipynb   
   и сервера xmlrpc\_server.ipynb;
3. Переписать код тетрадей клиента xmlrpc\_client.ipynb и сервера xmlrpc\_server.ipynb с Python 2 на Python 3. Для примера работы с Python 3 и изображением изучить код тетради «gdal\_tiff\_mask\_2022\_PIL.ipynb»;
4. Дополнить функционал сервера функцией проверки на наличие человека в черном списке с проверкой по ФИО и дате рождения;
5. Модифицировать функцию сервера send\_back\_inversion на корректную работу также с монохромными изображениями, т.е. входными данными могут быть размерности (M, N, 3) и (M, N, 1). Для инверсии изображения запрещено использовать библиотечные функции;
6. Дополнить функционал сервера функцией бинаризации изображения по задаваемому порогу 1-255 для цветных и монохромных изображений. Для бинаризации изображений запрещено использовать библиотечные функции;
7. Дополнить функционал сервера функцией разворота изображения относительно вертикали для цветных и монохромных изображений. Для поворота изображения запрещено использовать библиотечные функции.
8. Дополнить функционал клиента проверкой функций сервера по пунктам 4-7 задания.
9. Оформить отчет по результатам выполнения ЛР.

Листинг **xmlrpc\_server.ipynb**

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

import xmlrpc.client

import datetime

import pandas as pd

import numpy as np

import pickle

class RequestHandler(SimpleXMLRPCRequestHandler):

rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(("localhost", 8008),

requestHandler=RequestHandler)

# Тест

def ping():

return True

server.register\_function(ping, 'ping')

# Время сервера

def now():

return datetime.datetime.now()

server.register\_function(now, 'now')

# Отображение строкового вида, типа и значений

def show\_type(arg):

return (str(arg), str(type(arg)), arg)

server.register\_function(show\_type, 'type')

# Сумма

def test\_sum(a, b):

return a + b

server.register\_function(test\_sum, 'sum')

# Степень

def test\_pow(a, b):

return a\*\*b

server.register\_function(test\_pow, 'pow')

# Проверка нахождения клиента в черном списке c использованием Pandas Data Frame

def black\_list\_check(sname):

frame = pd.read\_csv('bad\_boys2.csv', header=0, sep=',', encoding='utf8')

exist = any(frame['Surname'] == sname)

if (exist == True):

return sname + ": "+ "bad\_boy"

else:

return sname + ": "+ "good\_boy"

server.register\_function(black\_list\_check, 'black\_list\_check')

# Проверка нахождения клиента в черном списке по фио и дате рождения c использованием Pandas Data Frame

def black\_list\_check\_by\_name\_and\_birth(surname,name,patronym,birth):

frame = pd.read\_csv('bad\_boys2.csv', header=0, sep=',', encoding='utf8')

# processed\_frame = frame.query("Surname == @surname and Name == @name and Patronym = patronym and Birth = birth")

# exist = len(processed\_frame) != 0

exist = False

for index, el in frame.iterrows():

if el['Surname'] == surname and el['Name'] == name and el['Patronym'] == patronym and el['Birth'] == birth:

exist = True

break

return surname + " " + name + " " + patronym + " " + birth+ ": "+ ("bad\_boy" if exist else "good\_boy")

server.register\_function(black\_list\_check\_by\_name\_and\_birth, 'black\_list\_check\_by\_name\_and\_birth')

# Бинарная передача данных

def send\_back\_binary(bin\_data):

data = bin\_data.data

return xmlrpc.client.Binary(data)

server.register\_function(send\_back\_binary, 'send\_back\_binary')

# Инверсия цвета

# На вход изображение RGB размерности (M, N, 3) и (M, N, 1) со значениями 0-255

def send\_back\_inversion(bin\_data):

img\_arr = pickle.loads(bin\_data.data)

# img\_arr=255-img\_arr

inverse = lambda x: 255 - x

height = img\_arr.shape[0]

weight = img\_arr.shape[1]

is\_monochrome = len(img\_arr.shape)==2

if is\_monochrome:

for i in range(height):

for j in range(weight):

img\_arr[i][j] = inverse(img\_arr[i][j])

else:

colors = img\_arr.shape[2]

for i in range(height):

for j in range(weight):

for c in range(colors):

img\_arr[i][j][c]=inverse(img\_arr[i][j][c])

pimg = pickle.dumps(img\_arr)

return xmlrpc.client.Binary(pimg)

server.register\_function(send\_back\_inversion, 'color\_inversion')

# Отзеркаливает изображение по вертикали

def rotate\_vertical(bin\_data):

img\_arr = pickle.loads(bin\_data.data)

# img\_arr=np.flip(img\_arr, axis = 1)

height = img\_arr.shape[0]

half\_weight = img\_arr.shape[1]//2

is\_monochrome = len(img\_arr.shape)==2

if is\_monochrome:

for i in range(height):

for j in range(half\_weight):

img\_arr[i][j],img\_arr[i][-j-1] = img\_arr[i][-j-1], img\_arr[i][j]

else:

colors = img\_arr.shape[2]

for i in range(height):

for j in range(half\_weight):

for c in range(colors):

img\_arr[i][j][c],img\_arr[i][-j-1][c] = img\_arr[i][-j-1][c], img\_arr[i][j][c]

pimg = pickle.dumps(img\_arr)

return xmlrpc.client.Binary(pimg)

server.register\_function(rotate\_vertical, 'rotate\_vertical')

# Бинаризация по порогу

def threshold\_img(bin\_data,limit):

img\_arr = pickle.loads(bin\_data.data)

# img\_arr = (img\_arr > limit)\*255

threshold\_val\_fun = lambda x: 255 if x>limit else 0

height = img\_arr.shape[0]

weight = img\_arr.shape[1]

is\_monochrome = len(img\_arr.shape)==2

if is\_monochrome:

for i in range(height):

for j in range(weight):

img\_arr[i][j] = threshold\_val\_fun(img\_arr[i][j])

else:

colors = img\_arr.shape[2]

for i in range(height):

for j in range(weight):

for c in range(colors):

img\_arr[i][j][c]=threshold\_val\_fun(img\_arr[i][j][c])

pimg = pickle.dumps(img\_arr)

return xmlrpc.client.Binary(pimg)

server.register\_function(threshold\_img, 'threshold\_img')

print("Listening on port 8008...")

server.serve\_forever()

Листинг **xmlrpc\_client.ipynb**

import xmlrpc.client

import pickle

server = xmlrpc.client.ServerProxy("http://localhost:8008")

print ('Ping:', server.ping())

print ('Server datetime:', server.now())

print ('View, type, value:', server.type(2))

print ('View, type, value:', server.type(2.))

# print ('View, type, value:', server.type('My string'))

# print ('View, type, value:', server.type("My string"))

# print ('View, type, value:', server.type([1,2,3]))

# print ('View, type, value:', server.type(["one", "two", "three"]))

# print ('View, type, value:', server.type((1,2,"3")))

# print ('Sum 2 + 3 :', server.sum(2, 3))

# print ('Pow 2^3: ', server.pow(2, 3))

from PIL import Image # Работа с изображением

from pylab import \* # Отображение

import xmlrpc.client

# Режим отображения внутри ноутбука

%matplotlib inline

gray()

img = Image.open('Jellyfish.jpg')

img\_arr = array(img)

# Исходное изображение

imshow(img\_arr)

# Тест бинарной передачи данных

pimg = pickle.dumps(img\_arr) # универсально

img\_bin = xmlrpc.client.Binary(pimg)

img\_bin2 = server.send\_back\_binary(img\_bin)

img\_arr2 = pickle.loads(img\_bin2.data) # универсально

# Изображение после возрата с сервера

imshow(img\_arr2)

# Тест вращения по вертикали

def send\_to\_server\_aplly\_func\_get\_res(img\_arr\_in, func, second\_arg = None):

pimg = pickle.dumps(img\_arr\_in)

img\_bin = xmlrpc.client.Binary(pimg)

if second\_arg is None:

img\_bin2 = func(img\_bin)

else:

img\_bin2 = func(img\_bin, second\_arg)

img\_arr\_out = pickle.loads(img\_bin2.data)

return img\_arr\_out

img\_arr\_rotate = send\_to\_server\_aplly\_func\_get\_res(img\_arr, server.rotate\_vertical)

# Изображение после возрата с сервера

imshow(img\_arr\_rotate)

# Изображение с инверсией цвета

img\_arr\_inv = send\_to\_server\_aplly\_func\_get\_res(img\_arr, server.color\_inversion)

imshow(img\_arr\_inv)

# Бинаризация по порогу

img\_arr\_inv = send\_to\_server\_aplly\_func\_get\_res(img\_arr, server.threshold\_img, 128)

imshow(img\_arr\_inv)

print(server.black\_list\_check\_by\_name\_and\_birth('Иванов','Иван','Иванович','22.03.1989'))

print(server.black\_list\_check\_by\_name\_and\_birth('Светлый','Иван','Иванович','22.03.1989'))

**Выполнение заданий 4 – 7**

4. Дополнить функционал сервера функцией проверки на наличие человека в черном списке с проверкой по ФИО и дате рождения.

На рисунке 1 представлен код сервера, реализующий задание 4.

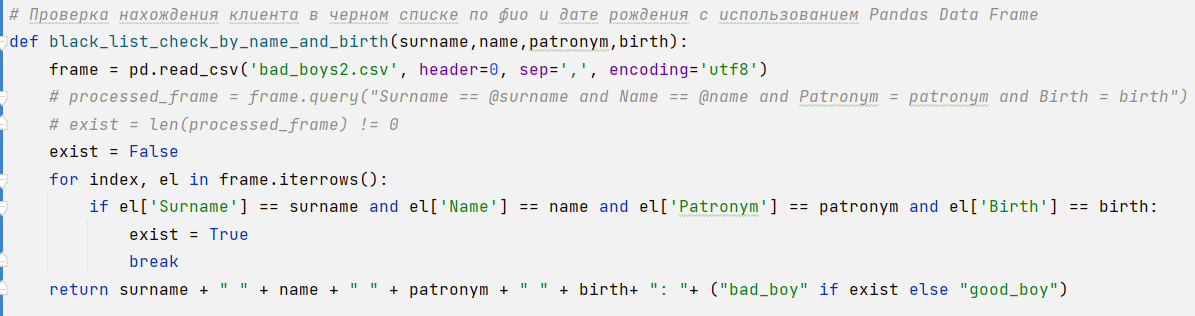


Рисунок 1 - функция проверки на наличие человека в черном списке с проверкой по ФИО и дате рождения

Вызов этой функции на клиенте представлен на рисунке 2.

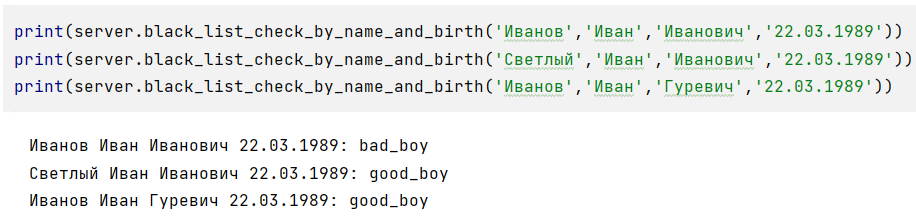


Рисунок 2 – вызов функции проверки на наличие человека в черном списке с проверкой по ФИО и дате рождения

5. Модифицировать функцию сервера send\_back\_inversion на корректную работу также с монохромными изображениями, т.е. входными данными могут быть размерности (M, N, 3) и (M, N, 1). Для инверсии изображения запрещено использовать библиотечные функции.

На рисунке 3 представлен код сервера, реализующий задание 5.

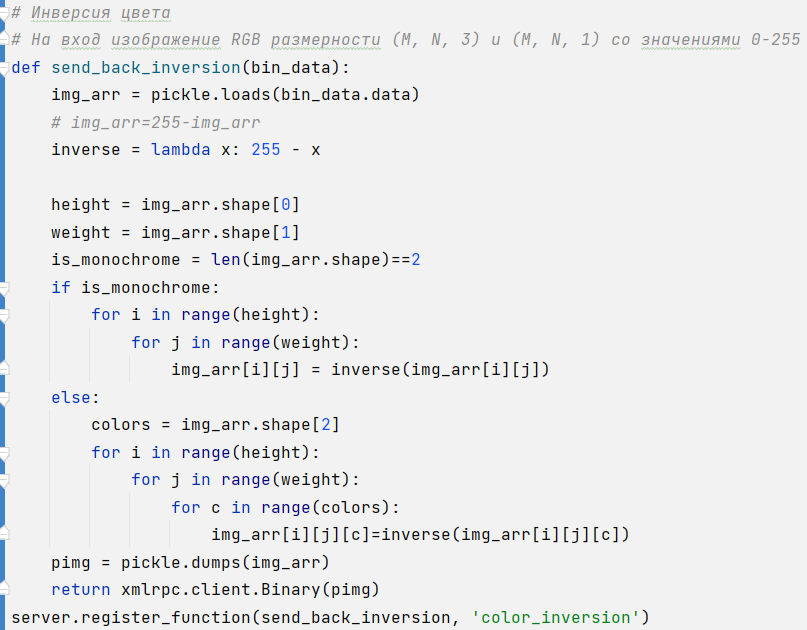


Рисунок 3 - функция инвертирующая цвет изображения

Для уменьшения дублирования кода при вызове этой функции на клиенте и функций по следующим заданиям была создана функция, которая отправляет бинарные данные на сервер, применяет функцию из параметра и возвращает распакованные данные обратно на клиент. Ее листинг представлен на рисунке 4.

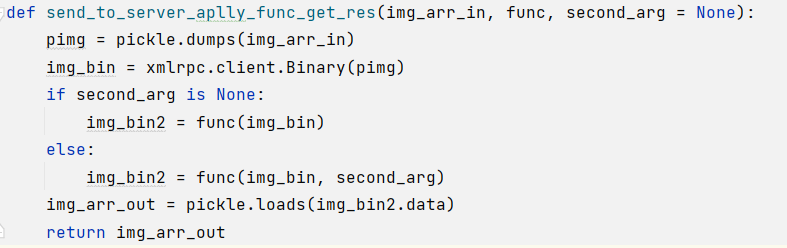


Рисунок 4 - вспомогательная функция для упаковки данных в бинарный формат, применения функции и возврата распакованных данных на клиент.

Вызов функции инвертирования изображения на клиенте с результатом применения к цветному изображению представлен на рисунке 5.

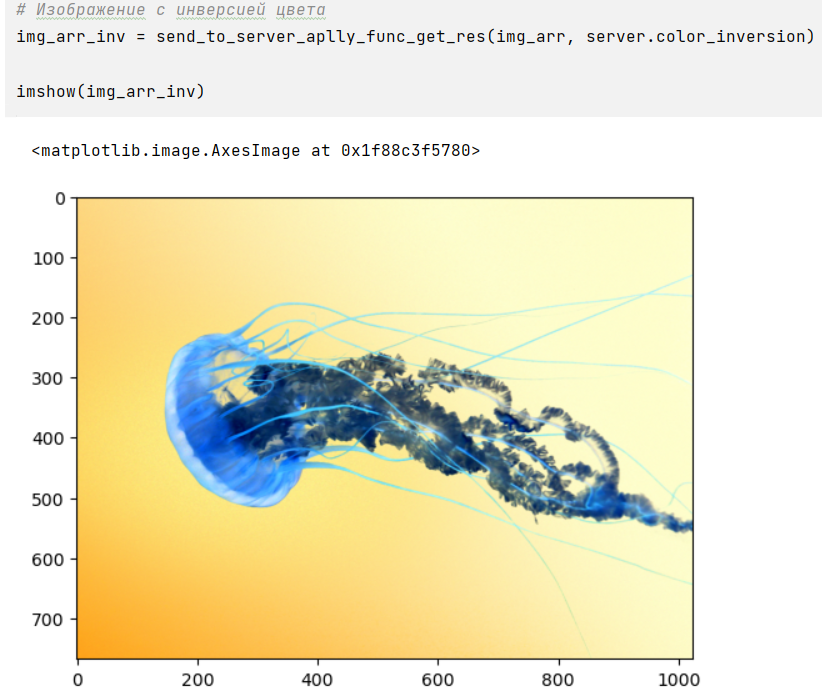


Рисунок 5 – Вызов функции инвертирования изображения на клиенте с результатом применения к цветному изображению

Инверсия монохромного изображения представлена на рисунке 6.



Рисунок 6 – Инверсия монохромного изображения

6. Дополнить функционал сервера функцией бинаризации изображения по задаваемому порогу 1-255 для цветных и монохромных изображений. Для бинаризации изображений запрещено использовать библиотечные функции;

На рисунке 7 представлен код сервера, реализующий задание 6.



Рисунок 7 – Функция бинаризации изображения по задаваемому порогу

Вызов функции бинаризации изображения на клиенте с результатом применения к цветному изображению представлен на рисунке 8.

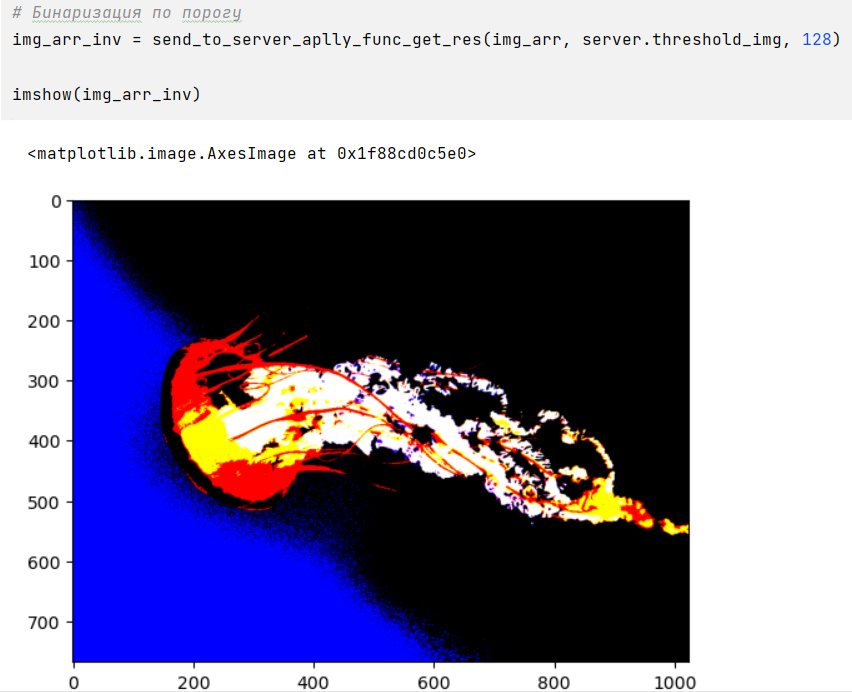


Рисунок 8 – Функция бинаризации цветного изображения по порогу 128

Бинаризация монохромного изображения по порогу 64 представлена на рисунке 9.



Рисунок 9 – Бинаризация цветного изображения по порогу 64

7. Дополнить функционал сервера функцией разворота изображения относительно вертикали для цветных и монохромных изображений. Для поворота изображения запрещено использовать библиотечные функции.

На рисунке 10 представлен код сервера, реализующий задание 7. 

Рисунок 10 – Функция разворота изображения по вертикали

Вызов функции для цветного изображения представлен на рисунке 11.

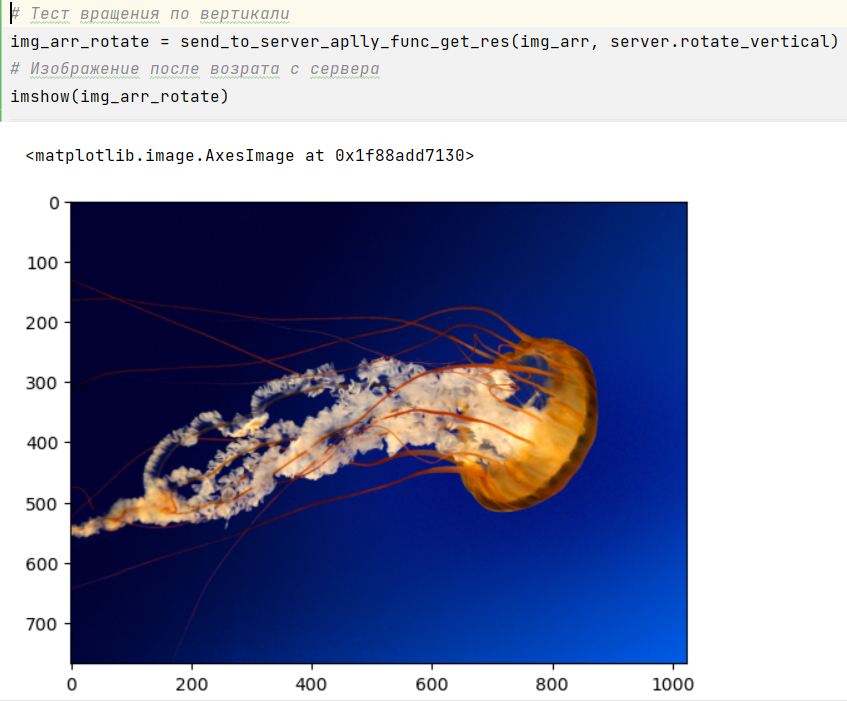


Рисунок 11 – Разворот цветного изображения по вертикали

Вызов функции для монохромного изображения представлен на рисунке 12.



Рисунок 12 – Разворот монохромного изображения по вертикали

Выводы

В ходе выполнения данной работы были изучены методы фильтрации значений по DataFrame, разработаны функции для инверсии цвета изображения, бинаризации по порогу, разворота по вертикали.

Сервис ориентированную архитектуру целесообразно использовать, если приложение может быть разделено на несколько автономных серверов. Сервисы с такой архитектурой не зависят друг от друга и могут использоваться разными приложениями одновременно. Также такие сервисы могут повторно переиспользоваться в новых проектах, что облегчает и ускоряет разработку новых приложений. Легче создавать, обновлять и отлаживать небольшие сервисы, чем большие блоки кода в монолитных приложениях.