Министерство образования и науки Российской Федерации Федеральное агентство по образованию

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**Использование протокола XML-RPC**

Лабораторная работа №8

по курсу «Разработка программных систем»

Выполнил студент группы ИВТб-31 /Категов А. Д./ Проверил преподаватель /Чистяков Г.А./

Киров 2024

1. Цель

Целью работы является знакомство с протоколом XML-RPC, а также получение навыков применения этого протокола для организации серверного взаимодействия.

1. Задание

В соответствии с выбранной тематикой разработать клиентское приложение, делегирующее выполнение вычислительных действий серверному приложению, и серверное приложение, выполняющее вычисления клиента. Для организации взаимосвязи между клиентом и сервером использовать протокол XML-RPC.

Для выполнения лабораторной работы необходимо решить следующие задачи:

* Согласовать тематику разработки с преподавателем
* Разработать структуры клиентского и серверного приложений
* Реализовать приложения
* Продемонстрировать работу приложения.

1. Листинг программы

Листинг программной реализации приведен в приложении А.

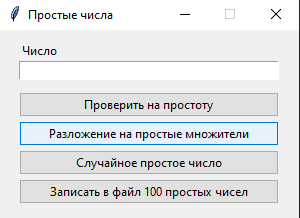


Рисунок 1 – Пользовательский интерфейс

**Вывод**: в ходе выполнения лабораторной работы были изучены основные возможности протокола XML-RPC. В процессе работы было разработано серверное приложение, реализующее функции для работы с простыми числами и клиентское приложение, делегирующее выполнение вычислительных действий серверному приложению. Взаимосвязь между клиентом и сервером организована с помощью протокола XML-RPC. На основании всего сказанного выше можно сделать вывод о том, что цель достигнута, необходимые знания получены, задание выполнено верно, что доказывается полученными результатами.

Приложение А

(обязательное)

Листинг программы

**model.py**

import xmlrpc.client

import socket

class Model:

def \_\_init\_\_(self):

self.server = None

def connect(self):

connected, self.server = self.\_get\_rpc()

return connected

def \_get\_rpc(self):

"""checking the connection to the server"""

serv = xmlrpc.client.ServerProxy("http://localhost:8888/")

try:

serv.\_() # call a fictive method.

except xmlrpc.client.Fault:

pass # connected to the server and the method doesn't exist which is expected.

except ConnectionRefusedError:

return False, None

else:

return False, None

return True, serv

# Methods for working with the server

is\_prime = lambda self, x: self.server.is\_prime(str(x))

prime\_factorization = lambda self, n: self.server.prime\_factorization(str(n))

get\_random\_prime = lambda self: self.server.get\_random\_prime()

generate\_primes = lambda self, n: self.server.generate\_primes(str(n))

**controller.py**

class Controller:

    def \_\_init\_\_(self, model, view): # constructor

        self.model = model

        self.view = view

    def int\_conversion(self, value):

        """conversion to int"""

        try:

            v = int(value)

            return v

        except ValueError:

            return None

    def is\_prime(self, value):

        """checking for a prime"""

        v = self.int\_conversion(value)

        if (v is None) or (v < 0):

            self.view.show\_result(1, "Введите неотрицательное целое число")

            return

        if v == 1 or v == 0:

            self.view.show\_result(0, "Число не является ни простым ни составным")

        else:

            try:

                self.view.show\_result(0, "Число простое" if self.model.is\_prime(v) else "Число составное")

            except Exception:

                self.view.show\_result(0, "Вы ввели слишком большое число")

    def prime\_factorization(self, value):

        v = self.int\_conversion(value)

        if (v is None) or (v < 0):

            self.view.show\_result(1, "Введите неотрицательное целое число")

            return

        if v == 1 or v == 0:

            self.view.show\_result(0, "Число не является ни простым ни составным")

        else:

            try:

                div\_str = self.model.prime\_factorization(v)

                r = list(int(x) for x in div\_str.split())

                self.view.show\_result(0, f"Простые множители: {\*r,}" if len(r) > 1 else "Это простое число")

            except Exception:

                self.view.show\_result(0, "Вы ввели слишком большое число")

    def get\_random\_prime(self):

        self.view.set\_input\_value(self.model.get\_random\_prime())

    def file\_name\_valid(self, value):

        """determines whether the file name is valid"""

        try:

            invalid\_chars = ['/', '\\', '|', ':', '\*', '?', '"', '<', '>']

            for char in invalid\_chars:

                if char in value:

                    return None

            if (value == '') or (len(value) > 50):

                return None

            return value

        except ValueError:

            return None

    def generate\_primes(self, filename):

        """save primes to file"""

        file = self.file\_name\_valid(filename)

        if (file is None):

            self.view.show\_result(1, "Недопустимое название файла")

            return

        primes\_str = self.model.generate\_primes(100)

        primes = list(int(x) for x in primes\_str.split())

        with open(file + '.txt', 'w') as file:

            for prime in primes:

                file.write(f"{prime}\n")

**view.py**

from tkinter import \*

import tkinter.messagebox as messagebox

import tkinter.ttk as ttk

class View(ttk.Frame):

    def \_\_init\_\_(self, parent): #constructor

        super().\_\_init\_\_(parent)

        self.controller = None

        self.number\_input = None

        self.grid\_columnconfigure(0, weight=1)

    def set\_controller(self, controller):

        self.controller = controller

    def create\_controls(self):

        """creates a label, entry, buttons and assigns a click response to them"""

        label = Label(self, text="Число")

        label.grid(row=0, column=0, padx=10, sticky="w")

        self.number\_input = Entry(self)

        self.number\_input.grid(row=1, column=0, padx=10, pady=(0, 10), sticky="we")

        check\_button = ttk.Button(self, text='Проверить на простоту', command=self.check)

        check\_button.grid(row=2, column=0, padx=10, pady=2, sticky="we")

        factorize\_button = ttk.Button(self, text='Разложение на простые множители', command=self.factorize)

        factorize\_button.grid(row=3, column=0, padx=10, pady=2, sticky="we")

        random\_prime\_button = ttk.Button(self, text='Случайное простое число', command=self.random\_prime)

        random\_prime\_button.grid(row=4, column=0, padx=10, pady=2, sticky="we")

        next\_prime\_button = ttk.Button(self, text='Записать в файл 100 простых чисел', command=self.generate\_primes)

        next\_prime\_button.grid(row=5, column=0, padx=10, pady=2, sticky="we")

        self.grid(row=0, column=0, pady=10, padx=10,  sticky="we")

    def check(self):

        if self.controller:

            if self.number\_input.get():

                self.controller.is\_prime(self.number\_input.get())

    def factorize(self):

        if self.controller:

            if self.number\_input.get():

                self.controller.prime\_factorization(self.number\_input.get())

    def random\_prime(self):

        if self.controller:

            self.controller.get\_random\_prime()

    def generate\_primes(self):

        if self.controller:

            if self.number\_input.get():

                self.controller.generate\_primes(self.number\_input.get())

    def show\_result(self, mtype, msg):

        if mtype == 0:

            messagebox.showinfo("Результат", msg)

        elif mtype == 1:

            messagebox.showerror("Ошибка", msg)

    def set\_input\_value(self, v):

        self.number\_input.delete(0, END)

        self.number\_input.insert(0, v)

**client.py**

from tkinter import \*

from client\_stuff.view import View

from client\_stuff.controller import Controller

from client\_stuff.model import Model

class App(Tk):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        model = Model()

        connected = model.connect()

        if not connected:

            print("Server is not connected")

            self.destroy()

            return

        self.title('Простые числа')

        self.geometry('300x190')

        self.resizable(False, False)

        self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

        view = View(self)

        self.controller = Controller(model, view)

        view.set\_controller(self.controller)

        view.create\_controls()

        self.grid\_columnconfigure(0, weight=1)

    def on\_closing(self):

        self.destroy()

if \_\_name\_\_ == '\_\_main\_\_':

    app = App()

    app.mainloop()

**server.py**

from xmlrpc.server import SimpleXMLRPCServer

import server\_stuff.prime as prime

def main():

    server = SimpleXMLRPCServer(("localhost", 8888))

    print("Listening on port 8888...")

    server.register\_function(prime.is\_prime, "is\_prime")

    server.register\_function(prime.prime\_factorization, "prime\_factorization")

    server.register\_function(prime.get\_random\_prime, "get\_random\_prime")

    server.register\_function(prime.generate\_primes, "generate\_primes")

    server.serve\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**prime.py**

import random

import sys

sys.setrecursionlimit(2000)

MAX\_PRIME = 999999937

def is\_prime(n): # optimized trial division with 6k optimization

    n = int(n)

    if n <= 1: # numbers less than or equal to 1 are not prime

        return False

    if n <= 3:                   # 2 and 3 are prime

        return True

    if n % 2 == 0 or n % 3 == 0: # exclude even numbers and numbers divisible by 3

        return False

    if n > MAX\_PRIME:

        raise Exception('obtained a number exceeding the MAX\_PRIME')

    i = 5 # check the divisors from 5 to sqrt(n) in increments of 6

    while i \* i <= n:

        if n % i == 0 or n % (i + 2) == 0:

            return False

        i += 6

    return True

def prime\_factorization\_int(num):

    factors = []

    k = 2

    while num > 1:

        while num % k == 0:

            factors.append(k)

            num //= k

        k += 1

    return factors

#prime\_factorization\_int = lambda num, k = 2: [] if num == 1 else [num] if is\_prime(num) else [k] + prime\_factorization(num // k, k) if num % k == 0 else prime\_factorization(num, k + 1)

def prime\_factorization(n):

    n = int(n)

    return ' '.join(str(x) for x in prime\_factorization\_int(n))

def next\_prime(i):  # find the next prime number

    candidate = i + 1

    while not is\_prime(candidate):

        candidate += 1

    return candidate

get\_random\_prime = lambda: str(next\_prime(random.randint(2, MAX\_PRIME - 1)))

def generate\_primes(size): # generating an array of prime numbers

    s = int(size)

    primes = []

    while len(primes) < s:

        prime = get\_random\_prime()

        primes.append(prime)

    return ' '.join(str(x) for x in primes)