МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГБОУ ВО
«Пермский государственный национальный исследовательский университет»

Механико-математический факультет
Кафедра информационной безопасности и систем связи

Отчёт

по лабораторной работе №3 «Разработка распределенного приложения, использующего технологию веб-служб»
по дисциплине «Технологии разработки распределенных приложений»

Работу выполнила
студентка гр. КМБ-16
Кузнецова Александра
Дмитриевна
«24» ноября 2020

Проверил
доцент кафедры
прикладной математики
и информатики,
к.ф.-м.н., доц.
Деменев Алексей
Геннадьевич

Пермь, 2020 г

Содержание

Постановка задачи

Цель: изучение возможностей технологии веб-служб для создания распределенных приложений.

Формируемые компетенции: способность применять на практике теоретические основы и общие принципы разработки распределенных систем; способность использовать на практике стандарты сетевого взаимодействия компонент распределенной системы.

Постановка задачи:

Необходимо реализовать веб-сервис и приложение, его использующее

1. Веб-сервис и приложение, его использующее, должны быть разработаны на разных объектно-ориентированных языках программирования.
2. Веб-сервис должен принимать параметры и передавать приложению результат.
3. В качестве параметров должны передаваться объекты классов, написанных самостоятельно, т.е. не должны передаваться строки, числа или другие простейшие типы.

1. Технологии и структура приложения

1.1 Описание предметной области

Предметная область – автоматический перевод текста. В приложении реализована функция перевода текста с определенного пользователем исходного языка на конечный язык, также определяемый пользователем. В качестве движка для перевода текста используется Google Neural Machine Translation [1], взаимодействие с которым в приложении организовано через библиотеку googletrans 3.0.0 [2].

1.2 Использованные технологии

Серверная часть приложения написана на языке Python 3.8 [3] и содержит логику обращения к Google Neural Machine Translation и получения ответа. Сервер взаимодействует с движком для перевода по протоколу SOAP [4], взаимодействие организовано с помощью библиотек spyne 2.13.16 [5] и lxml 4.6.1 [6]. Среда разработки – PyCharm Professional 2020.2 [7], предоставленная по индивидуальной студенческой лицензии JetBrains [8].

Клиентская часть приложения написана на языке Go 1.15.5 [9]. Для взаимодействие с сервером использовалась дополнительная библиотека gopkg.in/yaml.v2 [10] для работы с YAML. Для разработки клиентской части не использовалась специализированная IDE, код был написан в стандартном Notepad [11] и скомпилирован с помощью консольной утилиты go compiler [12].

1.3 Структура приложения, сборка и запуска

Библиотека spyne генерирует WSDL веб-сервиса автоматически. Работа происходит с объектом – исходный текст (строковый тип), язык исходного текста (строковый тип), итоговый язык перевода (строковый тип). От сервера ответ приходит в виде единственной строки. И входящие, и исходящие строки закодированы с помощью Unicode.

Определение входящих и исходящих типов данных для веб-сервиса осуществляется с помощью декоратора @rpc библиотеки spyne. Все значения, объявленные до параметра *_returns*, являются исходными и при генерации WSDL определяются как составные части объекта:

```
@rpc(Unicode, Unicode, Unicode, _returns=Unicode)
```

В итоге в WSDL входящие и исходящие типы определены как *сложные* (ComplexType):

```
<xs:complexType name="GoogleTranslate">
<xs:sequence>
<xs:element name="source_text" type="xs:string" minOccurs="0" nillabl
e="true"/>
```

```
<xs:element name="source_language" type="xs:string" minOccurs="0" nil
lable="true"/>
<xs:element name="destination_language" type="xs:string" minOccurs="0
" nillable="true"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="GoogleTranslateResponse">
<xs:sequence>
<xs:element name="GoogleTranslateResult" type="xs:string" minOccurs="
0" nillable="true"/>
</xs:sequence>
</xs:complexType>
```

Для веб-сервиса был создан отдельный класс SOAP с определенным в нем единственным методом GoogleTranslate. В методе происходит отправка элементов объекта *source_text*, *source_language*, *destination_language* в движок для перевода и получение ответа от него в *service_answer*.

```
class SOAP(ServiceBase):
    @rpc(Unicode, Unicode, Unicode, _returns=Unicode)
    def GoogleTranslate(ctx, source_text, source_language,
destination_language):
        translator = Translator()
        print(etree.tostring(ctx.in_document))
        service_answer = ""
        while service_answer == "":
            try:
                result = translator.translate(str(source_text),
src=source_language, dest=destination_language)
                service_answer = result.text
            except Exception:
                translator = Translator()

        return service_answer
```

Взаимодействие программы с удаленным движком организован через wsgi [13] – стандартный интерфейс Python. IP-адрес и порт определены в специальном конфигурационном файле *config.yml* и доступны для редактирования. В качестве протокола взаимодействия в обоих направлениях выбран SOAP v1.1:

```
app = Application([SOAP], tns='Translator',
                  # протокол для исходящих
                  in_protocol=Soap11(validator='lxml'),
                  # протокол для входящих
                  out_protocol=Soap11())
application = WsgiApplication(app)
if __name__ == '__main__':
```

```
    # загрузка конфигурационного файла
    config = load_config()
    # запуск через wsgi
    from wsgiref.simple_server import make_server
    server = make_server(config['soap_host'],
int(config['soap_port']), application)
    server.serve_forever()
```

Запуск веб-сервиса осуществляется с помощью команды `python3 main.py`.

Итоговый WSDL:

```
<wsdl:definitions xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:x
si="http://www.w3.org/2001/XMLSchema-
instance" xmlns:plink="http://schemas.xmlsoap.org/ws/2003/05/partner-
link/" xmlns:wsdlsoap11="http://schemas.xmlsoap.org/wsdl/soap/" xmlns
:wsdlsoap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:wsdl="htt
p://schemas.xmlsoap.org/wsdl/" xmlns:soap11enc="http://schemas.xmlsoa
p.org/soap/encoding/" xmlns:soap11env="http://schemas.xmlsoap.org/soa
p/envelope/" xmlns:soap12env="http://www.w3.org/2003/05/soap-
envelope" xmlns:soap12enc="http://www.w3.org/2003/05/soap-
encoding" xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing
" xmlns:xop="http://www.w3.org/2004/08/xop/include" xmlns:http="http:
//schemas.xmlsoap.org/wsdl/http/" xmlns:tns="Translator" targetNamesp
ace="Translator" name="Application">
<wsdl:types>
<xs:schema targetNamespace="Translator" elementFormDefault="qualified
">
<xs:complexType name="GoogleTranslate">
<xs:sequence>
<xs:element name="source_text" type="xs:string" minOccurs="0" nillabl
e="true"/>
<xs:element name="source_language" type="xs:string" minOccurs="0" nil
lable="true"/>
<xs:element name="destination_language" type="xs:string" minOccurs="0
" nillable="true"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="GoogleTranslateResponse">
<xs:sequence>
<xs:element name="GoogleTranslateResult" type="xs:string" minOccurs="
0" nillable="true"/>
</xs:sequence>
</xs:complexType>
<xs:element name="GoogleTranslate" type="tns:GoogleTranslate"/>
<xs:element name="GoogleTranslateResponse" type="tns:GoogleTranslateR
esponse"/>
</xs:schema>
</wsdl:types>
<wsdl:message name="GoogleTranslate">
```

```
<wsdl:part name="GoogleTranslate" element="tns:GoogleTranslate"/>
</wsdl:message>
<wsdl:message name="GoogleTranslateResponse">
<wsdl:part name="GoogleTranslateResponse" element="tns:GoogleTranslat
eResponse"/>
</wsdl:message>
<wsdl:service name="SOAP">
<wsdl:port name="Application" binding="tns:Application">
<wsdlsoap11:address location="http://localhost:8000/"/>
</wsdl:port>
</wsdl:service>
<wsdl:portType name="Application">
<wsdl:operation name="GoogleTranslate" parameterOrder="GoogleTranslat
e">
<wsdl:input name="GoogleTranslate" message="tns:GoogleTranslate"/>
<wsdl:output name="GoogleTranslateResponse" message="tns:GoogleTransl
ateResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="Application" type="tns:Application">
<wsdlsoap11:binding style="document" transport="http://schemas.xmlsoa
p.org/soap/http"/>
<wsdl:operation name="GoogleTranslate">
<wsdlsoap11:operation soapAction="GoogleTranslate" style="document"/>
<wsdl:input name="GoogleTranslate">
<wsdlsoap11:body use="literal"/>
</wsdl:input>
<wsdl:output name="GoogleTranslateResponse">
<wsdlsoap11:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
</wsdl:definitions>
```

После запуска WSDL доступен по адресу http://host:port/?wsdl.

Для тестирования работы приложения использовалась утилита SoapUI 5.6.0 [14]. С помощью утилиты производилось тестирование и отладка серверной части. Пример тестирования:

Рис. 1, пример тестирования приложения

Также на основе ответов, полученных с помощью SoapUI, было получено представление структуры ответа от веб-сервиса, использующееся далее в клиентской части.

Исходя из полученного WSDL исходящие SOAP-запросы имеют следующий вид:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tran="Translator">
   <soapenv:Body>
      <tran:GoogleTranslate>
         <!--Optional:-->
         <tran:source_text>STR</tran:source_text>
         <!--Optional:-->
         <tran:source_language>STR</tran:source_language>
         <!--Optional:-->
         <tran:destination_language>STR</tran:destination_language>
      </tran:GoogleTranslate>
   </soapenv:Body>
</soapenv:Envelope>
```

В клиентской части реализован ввод значений *source_text*, *source_language*, *destination_language* с консоли. IP-адрес и порт сервиса также загружаются из конфигурационного файла. Взаимодействие с сервисом организовано через отправку и получение XML:

```
type GoogleTranslate struct {
     XMLName xml.Name
     Body    struct {
          XMLName           xml.Name
          GoogleTranslateResponse struct {
               XMLName xml.Name
               Return  []string `xml:"GoogleTranslateResult"`
          } `xml:"GoogleTranslateResponse"`
     }
}

// wsdl service url
     url := fmt.Sprintf("%s%s%s%s%s",
          "http://",
          c.Host,
          ":",
          c.Port,
          "/?wsdl",
     )

     // payload
     payload := []byte(strings.TrimSpace(`
          <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tran="Translator">
              <soapenv:Body>
                 <tran:GoogleTranslate>
                    <!--Optional:-->
```

```
                    <tran:source_text>` + text + `</tran:source_text>
                    <!--Optional:-->
                    <tran:source_language>`    +    sourceLanguage    +
`</tran:source_language>
                    <!--Optional:-->
                    <tran:destination_language>`                        +
destinationLanguage + `</tran:destination_language>
                 </tran:GoogleTranslate>
              </soapenv:Body>
           </soapenv:Envelope>`,
     ))

     httpMethod := "POST"

     // soap action
     soapAction := "GoogleTranslate"

     // prepare the request
     req,      err      :=      http.NewRequest(httpMethod,      url,
bytes.NewReader(payload))
     if err != nil {
          log.Fatal("Error   on   creating   request   object.   ",
err.Error())
          return
     }

     // set the content type header, as well as the oter required
headers
     req.Header.Set("Content-type", "text/xml")
     req.Header.Set("SOAPAction", soapAction)

     // prepare the client request
     client := &http.Client{
          Transport: &http.Transport{
               TLSClientConfig: &tls.Config{
                    InsecureSkipVerify: true,
               },
          },
     }

     // dispatch the request
     res, err := client.Do(req)
     if err != nil {
          log.Fatal("Error on dispatching request. ", err.Error())
          return
     }

     // read and parse the response body
     result := new(GoogleTranslate)
     err = xml.NewDecoder(res.Body).Decode(result)
     if err != nil {
```

```
            log.Fatal("Error on unmarshaling xml. ", err.Error())
            return
      }

      // print the users data
      users := result.Body.GoogleTranslateResponse.Return
      fmt.Println("\nTranslated text:")
      fmt.Println(strings.Join(users, ", "))
```

Запуск клиентского приложения выполняется с помощью `go run soap_client.go`.

Пример работы:

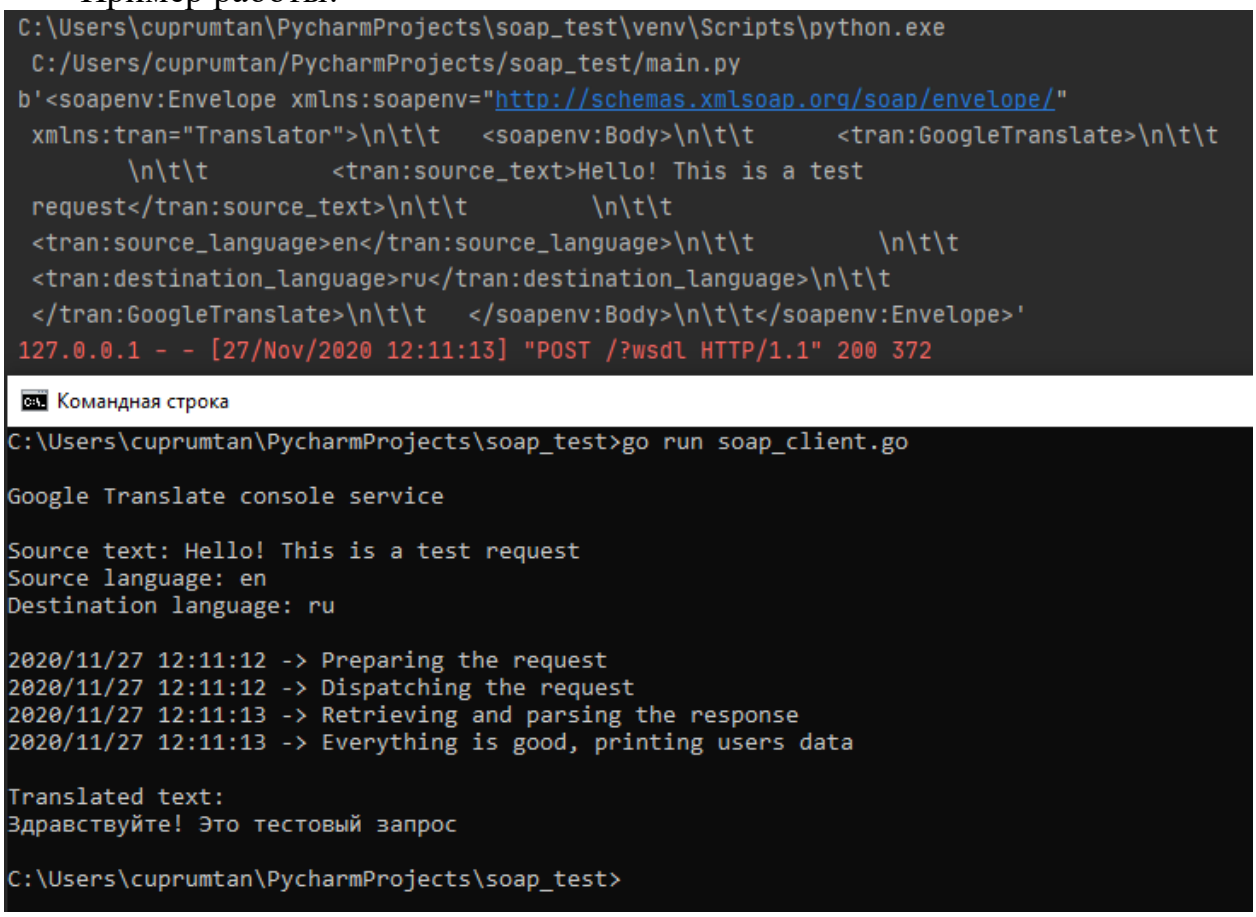

Рис. 2, пример работы приложения

2. Требования к приложению и их выполнение

2.1 Приложение, написанное студентом, работает в сети Интернет без сбоев

Требование выполнено, так как для приложения создан конфигурационный файл, в котором задаются IP-адрес и порт для работы веб-сервиса.

2.2 Веб-сервис и приложение, его использующее, написаны на разных языках программирования

Требование выполнено: серверная часть приложения написана на Python, клиентская на Go.

2.3 Веб-сервис принимает от приложения исходные данные и возвращает результат

Требование выполнено. В клиентской части приложения организован ввод исходных данных с консоли и последующая передача этих данных веб-сервису посредством XML-запроса в виде ComplexType. Веб-сервис определяет полученные данные корректно, проводит необходимые с ними манипуляции и возвращает ответ также в виде ComplexType, который успешно обрабатывается клиентом.

2.4 В качестве параметров веб-сервис принимает объекты классов

Требование выполнено. Приложение работает с объектом, содержащим три элемента - *source_text (string), source_language (string), destination_language (string).* В WSDL объект определен как ComplexType:

```
<xs:complexType name="GoogleTranslate">
<xs:sequence>
<xs:element name="source_text" type="xs:string" minOccurs="0" nillabl
e="true"/>
<xs:element name="source_language" type="xs:string" minOccurs="0" nil
lable="true"/>
<xs:element name="destination_language" type="xs:string" minOccurs="0
" nillable="true"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="GoogleTranslateResponse">
<xs:sequence>
<xs:element name="GoogleTranslateResult" type="xs:string" minOccurs="
0" nillable="true"/>
</xs:sequence>
</xs:complexType>
```

Список источников

1. Yonghui Wu, Mike Schuster, Zhifeng Chen. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. [Электронный ресурс] URL: https://arxiv.org/abs/1609.08144 (дата обращения 24.11.2020).
2. PyPI: googletrans 3.0.0. [Электронный ресурс] URL: https://pypi.org/project/googletrans/3.0.0/ (дата обращения 24.11.2020).
3. Python: Release 3.8. [Электронный ресурс] URL: https://www.python.org/downloads/release/python-386/ (дата обращения 24.11.2020).
4. W3C: SOAP specification. [Электронный ресурс] URL: https://www.w3.org/TR/soap/ (дата обращения 24.11.2020).
5. PyPI: spyne 2.13.16. [Электронный ресурс] URL: https://pypi.org/project/spyne/2.13.16/ (дата обращения 24.11.2020).
6. PyPl: lxml 4.6.1. [Электронный ресурс] URL: https://pypi.org/project/lxml/4.6.1/ (дата обращения 24.11.2020).
7. JetBrains Confluence: PyCharm 2020.2. [Электронный ресурс] URL: https://confluence.jetbrains.com/display/PYH/PyCharm+2020.2+Release+Notes (дата обращения 01.10.2020).
8. JetBrains Students License. [Электронный ресурс] URL: https://www.jetbrains.com/ru-ru/community/education/#students (дата обращения 01.10.2020).
9. Go: Go 1.15 Release Notes. [Электронный ресурс] URL: https://golang.org/doc/go1.15 (дата обращения 24.11.2020).
10. gopkg.in/yaml.v2 release page. [Электронный ресурс] URL: https://gopkg.in/yaml.v2 (дата обращения 24.11.2020).
11. Microsoft Notepad. [Электронный ресурс] URL: https://www.microsoft.com/ru-ru/p/windows-notepad (дата обращения 24.11.2020).
12. Go: Go code [Электронный ресурс] URL: https://golang.org/doc/install#code (дата обращения 24.11.2020).
13. WSGI wiki: what is WSGI. [Электронный ресурс] URL: https://wsgi.readthedocs.io/en/latest/what.html (дата обращения 24.11.2020).

14. SoapUI Open Source Release 5.6.0. [Электронный ресурс] URL: https://www.soapui.org/downloads/latest-release/release-notes/ (дата обращения 24.11.2020).