

Seminario 3: Servicios WEB y REST API en Python

Sistemas Distribuidos

Gabriel Guerrero Contreras

Departamento de Ingeniería Informática
Universidad de Cádiz



Curso 2019 – 2020

Indice

- 1 Componentes de un Sistema Distribuido
- 2 Servicios Web
- 3 REST API en Python
- 4 Bottle Framework
- 5 Bibliografía

Sección 1 | Componentes de un Sistema Distribuido

Componentes de un Sistema Distribuido

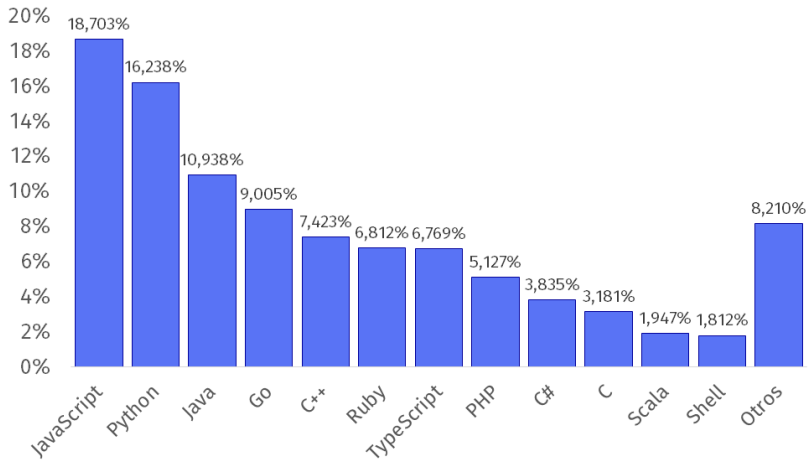
- Un sistema distribuido se compone de diferentes partes o componentes
- Cada uno de esos componentes puede estar implementado con un framework o lenguaje de programación diferente
- Cada lenguaje está especializado en un tipo de aplicación

Tipos de Lenguajes

- De bajo, medio o alto nivel
- Multiplataforma (Java)
- Paralelismo y concurrencia
- Plataforma web (PHP o Javascript)
- Tiempo real (Ada)
- Lenguajes "pegamento"(Perl)
- Calculo científico (Scala)

Tipos de Lenguajes

Popularidad de los Lenguajes de Programación según los proyectos de GitHub



Tipos de Lenguajes

Usar el mismo lenguaje

- Debe ser multiplataforma
- Debe poseer un extenso conjunto de librerías

Usar distintos lenguajes

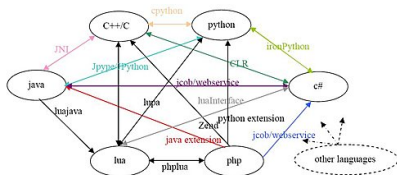
- ¿Comunicación entre componentes?
 - Bases de datos
 - Conexiones
 - Sistema de llamadas remotas

Tipos de Lenguajes

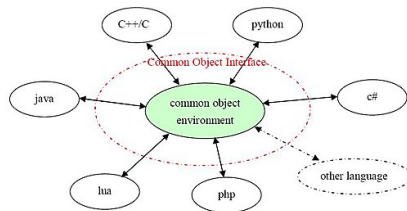
¿Tiene sentido utilizar el lenguaje más popular para todos los componentes de nuestro sistema distribuido?

Programación Multilenguaje

Librerías específicas



Interfaz de objeto común



Sección 2 | Servicios Web

Servicios Web (WS)

Definición

Sistema que permite la comunicación y el intercambio de datos entre aplicaciones y sistemas heterogéneos en entornos distribuidos expuestos en una intranet o a través de Internet.

- Estandarizado por el W3C: <https://www.w3.org/TR/ws-arch/>
- Ofrece un enfoque que permite **interoperar** a diferentes aplicaciones, sobre diferentes plataformas y/o frameworks

Tipos de Servicios

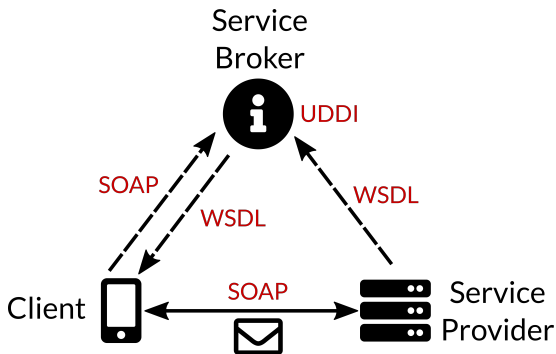
Servicios Web SOAP

- Exponen la funcionalidad como procedimientos y ejecutables remotos
- Las especificaciones están dictadas por los estándares SOAP y WSDL
- Tienen el objetivo de solucionar los problemas de integración heredados las tecnologías anteriores (COM, CORBA o RMI) y lograr su interoperatividad

Servicios Web REST

- Basados en la arquitectura web y en su estándar de base: **HTTP**
- Exponen completamente su funcionalidad como un conjunto coordinado de URIs
- Se diseño para abordar los problemas de SOAP
- Permite diferentes formatos de mensajes, como HTML, **JSON**, XML, y texto plano.

Interacción WS SOAP



Estándares Empleados

UDDI (Universal Description Discovery and Integration)

- Registro público diseñado para almacenar de forma estructurada información sobre Servicios Web y facilitar su descubrimiento

WSDL (Web Services Description Language)

- Protocolo estándar definido por el W3C para describir un servicio Web (**contrato**)
- Describe la interfaz pública de los servicios web:
 - Operaciones
 - Formatos de mensajes
 - Requisitos del protocolo
- Lo suelen construir automáticamente las herramientas de desarrollo

Estándares Empleados

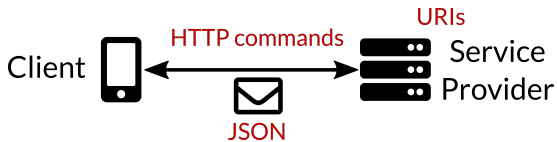
SOAP (Simple Object Access Protocol)

- Protocolo de comunicación de servicios y aplicaciones web
- Establece el formato para enviar y recibir mensajes
- Basado en **XML** e independiente de la plataforma
- Un mensaje SOAP contiene los siguientes elementos:
 - **Envelope** (obligatorio): identifica el documento XML como un mensaje SOAP
 - **Header** (opcional): permite extender un mensaje SOAP de forma modular y descentralizada
 - **Body** (obligatorio): contiene la información a transmitir
 - **Fault** (opcional): contiene la información sobre errores y estado

Mensaje SOAP

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

Interacción WS REST



Estándares Empleados

URI (Uniform Resource Identifier)

- Identifica un recurso por su nombre, por su ubicación o por ambos
- Comprende la URL y/o el URN:
 - URN: identifica de forma unívoca los recursos electrónicos por un nombre
 - [urn:isbn:0451450523](#)
 - URL: indica un recurso en Internet para poder localizarlo
 - [www.uca.es](#)
- [esquema://máquina/directorio/archivo](#)

Estándares Empleados

Comandos HTTP

- **GET**: Solicita el recurso ubicado en la URL especificada
- **HEAD**: Solicita el encabezado del recurso ubicado en la URL especificada
- **POST**: Envía datos al programa ubicado en la URL especificada
- **PUT**: Envía datos a la URL especificada
- **DELETE**: Borra el recurso ubicado en la URL especificada

Diferencias entre:

- **PUT** pone un recurso en la dirección especificada en la URL y es idempotente
 - **PUT** a la URL: `myServer.com/user/1134`
- **POST** envía datos a una URL para que el recurso en esa URI los maneje y **no** es idempotente
 - **POST** a la URL: `myServer.com/user`

Estándares Empleados

JSON (JavaScript Object Notation)

- Formato de intercambio de mensajes
- Completamente independiente del lenguaje de programación
- Consiste en colecciones de pares nombre/valor y listas ordenadas de valores

Ejemplo de información codificada en JSON

```
{
  "username" : "my_username",
  "password" : "my_password",
  "validation-factors" : {
    "validationFactors" : [
      {
        "name" : "remote_address",
        "value" : "127.0.0.1"
      }
    ]
  }
}
```

Misma información en XML

```
<authentication-context>
  <username>my_username</username>
  <password>my_password</password>
  <validation-factors>
    <validation-factor>
      <name>remote_address</name>
      <value>127.0.0.1</value>
    </validation-factor>
  </validation-factors>
</authentication-context>
```

SOAP vs REST

	SOAP	REST
Diseño	Estandarizado	Pautas y recomendaciones flexibles
Seguridad	Soporte SSL	HTTPS y SSL
Rendimiento	Requiere más recursos	Requiere menos recursos
Mensajes	XML	Texto plano, HTML, XML, JSON, YAML, y otros
Protocolos de Transferencia	HTTP, SMTP, UDP, y otros	HTTP
Recomendado para	Aplicaciones de alta seguridad, servicios financieros, pasarelas de pago	APIs públicas, servicios móviles, redes sociales
Ventajas	Seguridad y extensibilidad	Flexibilidad, escalabilidad y rendimiento

Sección 3 | REST API en Python

Cliente

- Utiliza peticiones HTTP
- Sencillo de implementar mediante la librería `requests`

Ejemplo petición GET

```
import requests

URL = "http://maps.googleapis.com/maps/api/geocode/json"
location = "delhi technological university"
PARAMS = {'address': location}
r = requests.get(url = URL, params = PARAMS)
data = r.json()

latitude = data['results'][0]['geometry']['location']['lat']
longitude = data['results'][0]['geometry']['location']['lng']
formatted_address = data['results'][0]['formatted_address']

print("Latitude:%s\nLongitude:%s\nFormatted Address:%s"
      %(latitude, longitude, formatted_address))
```

Ejemplo petición POST

```
import requests
```

```
API_ENDPOINT = "http://pastebin.com/api/api_post.php"  
API_KEY = "XXXXXXXXXXXXXXXXXXXX"
```

```
source_code = '''  
print("Hello, world!")  
a = 1  
b = 2  
print(a + b)  
'''
```

```
data = {'api_dev_key':API_KEY,  
        'api_option':'paste',  
        'api_paste_code':source_code,  
        'api_paste_format':'python'}  
r = requests.post(url = API_ENDPOINT, data = data)
```

```
pastebin_url = r.text  
print("The pastebin URL is:%s"%pastebin_url)
```

Servidor

- Complejo codificarlo a mano mediante `sockets`:
 - Gestionar peticiones y salida
 - Gestionar el paralelismo
 - Múltiples peticiones

Solución: Framework de desarrollo web

Frameworks Web Populares para Python



Pyramid

- Flexible, minimalista, rápido y fiable
- Primeros frameworks web que fue compatible con Python 3
- Ideal para desarrollo de aplicaciones web grandes

Frameworks Web Populares para Python

The Django logo is displayed in a large, dark green, lowercase font. The letters are bold and have a slightly irregular, hand-drawn style.

Django

- Mayor framework web basado en Python
- Comunidad grande y activa

Frameworks Web Populares para Python



Flask

- Microframework minimalista de solo un único archivo
- Varias extensiones disponibles

Frameworks Web Populares para Python



Bottle

- Microframework muy simple que proporciona un mínimo de herramientas al desarrollador
- Ideal para crear una API web realmente simple

Sección 4 | Bottle Framework

Instalación

- Recomendada: `$ sudo pip install bottle`
- Más información:
`https://bottlepy.org/docs/dev/tutorial.html`

Hola Mundo

```
from bottle import Bottle, run

app = Bottle()

@route('/hello/<name>')
def index(name):
    return template('<b>Hello {{name}}</b>!', name=name)

run(app, host='localhost', port=8080)
```

Ejecuta este script y luego dirígete con tu navegador a <http://localhost:8080/hello/world>

Route

- Nos permite asociar (`bind`) rutas a las funciones
- Podemos asociar más de una ruta a una función
- Se pueden definir parámetros entre `<>`
- Se puede indicar el tipo del parámetro (`:int` o `:float`)
- Con `:re` se pueden incluir expresiones regulares

Route

```
@route('/')
@route('/hello/<name>')
def greet(name='Stranger'):
    return template('Hello {{name}}, how are you?', name=name)
```

Route

```
@route('/object/<id:int>')
def callback(id):
    assert isinstance(id, int)

@route('/show/<name:re:[a-z]+>')
def callback(name):
    assert name.isalpha()
```

Request (GET)

```
from bottle import get, request # o route

@get('/cars') # o @route('/cars')
def getcars():
    cars = [ {'name': 'Audi', 'price': 52642},
              {'name': 'Mercedes', 'price': 57127},
              {'name': 'Skoda', 'price': 9000},
              {'name': 'Volvo', 'price': 29000},
              {'name': 'Bentley', 'price': 350000},
              {'name': 'Citroen', 'price': 21000},
              {'name': 'Hummer', 'price': 41400},
              {'name': 'Volkswagen', 'price': 21600} ]

    return dict(data=cars)
```

Request (POST)

```
from bottle import post, request # o route

@post('/login') # o @route('/login', method='POST')
def do_login():
    try:
        data = request.json()
    except:
        raise ValueError
    if data is None:
        raise ValueError

    username = data['username']
    password = data['password']
    if check_login(username, password):
        return "<p>Your login information was correct.</p>"
    else:
        return "<p>Login failed.</p>"
```

Request (PUT)

```
from bottle import put, request # o route

# o @route('/names/<oldname>', method='PUT')
@put('/names/<oldname>')
def update_handler(name):
    try:
        data = json.load(utf8reader(request.body))
    except:
        raise ValueError

    newname = data['name']
    _names.remove(oldname)
    _names.add(newname)

    # return 200 Success
    response.headers['Content-Type'] = 'application/json'
    return json.dumps({'name': newname})
```

Sección 5 | Bibliografía

Bibliografía

- <https://bottlepy.org/docs/dev/tutorial.html>
- <https://bottlepy.org/docs/dev/>
- <https://www.toptal.com/bottle/building-a-rest-api-with-bottle-framework>