Servicios REST

Antonio Espín Herranz

Contenidos

- Introducción servicios REST.
- SOAP vs REST
- Verbos REST: GET, PUT, DELETE, POST
- Clientes REST

Introducción

- El estilo REST *es una forma ligera de crear Servicios* Web.
- Se basan en las URLs.
- Proporcionan acceso a URLs para obtener información o realizar alguna operación.
- Son interesante para utilizar con **peticiones** de tipo **AJAX** y para acceder con **dispositivos con poco recursos**.

Características

- Sistema cliente / servidor.
- No hay estado → sin sesión.
- Soporta un sistema de caché
- ***Cada recurso tendrá una única dirección de red***.
- Sistema por capas.
- Variedad de formatos:
  - XML, HTML, text plain, JSON, etc.

Recursos

- Un recurso REST es cualquier cosa que sea direccionable a través de la Web.

- Algunos ejemplos de recursos REST son:
  - Una noticia de un periódico
  - La temperatura de Alicante a las 4:00pm

Algunos formatos soportados

Formato	Tipo MIME
Texto plano	`text/plain`
HTML	`text/html`
XML	`application/xml`
JSON	`application/json`

URI

- Una URI, o **Uniform Resource Identifier, en un servicio web RESTful es un** hiper-enlace a un recurso, y es la única forma de intercambiar representaciones entre clientes y servidores.

- Un servicio web RESTful expone un conjunto de recursos que identifican los objetivos de la interacción con sus clientes.

Formato de las peticiones

- La peticiones REST tienen un formato con este:
- http://localhost:8080/app/trabajadores/101
- **trabajadores: representa un recurso.**
- 101:El identificador del Trabajador, es el equivalente a …/trabajadores?id=101
- La URL de REST está orientada a recursos y localiza un recurso.

Verbos REST

- Los verbos nos permiten llevar a cabo acciones con los recursos.

- Se asocian con las operaciones **CRUD.**
  - **GET:** Obtener información sobre un recurso. El recurso queda identificado por su URL. **Operación read**.
  - **POST:** Publica información sobre un recurso. **Operación create.**
  - **PUT:** Incluye información sobre recursos en el Servidor. **Operación update.**
  - **DELETE:** Elimina un recurso en el Servidor. **Operación delete.**

REST vs SOAP

	REST	SOAP
Características	Las operaciones se definen en los mensajes. Una dirección única para cada instancia del proceso. Cada objeto soporta las operaciones estándares definidas. Componentes débilmente acoplados.	Las operaciones son definidas como puertos WSDL. Dirección única para todas las operaciones. Múltiple instancias del proceso comparten la misma operación. Componentes fuertemente acoplados.
Ventajas declaradas	Bajo consumo de recursos. Las instancias del proceso son creadas explícitamente. El cliente no necesita información de enrutamiento a partir de la URI inicial. Los clientes pueden tener una interfaz "listener" (escuchadora) genérica para las notificaciones. Generalmente fácil de construir y adoptar.	Fácil (generalmente) de utilizar. La depuración es posible. Las operaciones complejas pueden ser escondidas detrás de una fachada. Envolver APIs existentes es sencillo Incrementa la privacidad. Herramientas de desarrollo.
Posibles desventajas	Gran número de objetos. Manejar el espacio de nombres (URIs) puede ser engorroso. La descripción sintáctica/semántica muy informal (orientada al usuario). Pocas herramientas de desarrollo.	Los clientes necesitan saber las operaciones y su semántica antes del uso. Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones. Las instancias del proceso son creadas implícitamente.

¿Dónde es útil REST?

- El servicio Web no tiene estado.
- Tanto el productor como el consumidor del servicio conocen el contexto y contenido que va a ser comunicado
- El ancho de banda es importante y necesita ser limitado.
  - REST es particularmente útil en dispositivos con escasos recursos como PDAs o teléfonos móviles
- Los desarrolladores pueden utilizar tecnologías como **AJAX**

¿Dónde es útil SOAP?

- Se establece un contrato formal para la descripción de la interfaz que el servicio ofrece → WSDL.
- La arquitectura necesita manejar procesado asíncrono e invocación.

Librerías python para REST

- Instalar la librería: **bottle**
  - **pip install bottle**

  import bottle

  - Viene con un servidor para poder publicar los servicios.
  - También suministra clientes para consumir los servicios.
  - Es un **framework** para implementar **APIs de REST**, **ligero** y **eficiente**.
  - Cumple con el **estándar WSGI**, se puede utilizar con servidores **compatibles** con WSGI, por ejemplo, **Apache**.
  - Independiente de la BD.

Servidor

```
from bottle import route, run

if __name__=='__main__':
    @route('/hello/:name', method='GET')
    def get_set_of_results(name):
        return {'message':'Hello %s' % name}

    run()
```

Cliente

```
from http.client import HTTPConnection

conn = HTTPConnection('localhost:8080')
print('Ha creado la conexion ...')

conn.request('GET', '/hello/ana')
print('Hace la peticion GET ...')

response = conn.getresponse()
print('Obtiene la respuesta ...')


print(response.status)
print(response.reason)
print(response.read())
```

Métodos

- El método **run**() que lanza el script servidor, pone en marcha el servidor.

- Por defecto, tiene dos parámetros:
  - **host** = 127.0.0.1
  - **port** = 8080

- Se pueden hacer recargas según se modifica el código:

```
from bottle import run
run(reloader=True, debug=True)
```

- **@route**('/hello/:name', method='GET')
  - Es un decorador para crear la url, con los : delante de un identificador se declaran parámetros que luego tendrá que recibir la función que se está decorando.

Parámetros

- Los parámetros se indican en la URI, mediante los símbolos **<**nombre_param**>** y la función manejadora recibirá tantos parámetros como se hayan indicado en la URI.

- Ejemplo:

```
@get('/<param1>/<param2>')
def handler(param1, param2):
    pass
```

- Se pueden **crear rutas en cascada con los decoradores** para utilizar **parámetros opcionales** (es decir, la función se puede decorar más de una vez, pero tiene que ser el mismo decorador con distinta URI):

```
@get('/<param1>')
@get('/<param1>/<param2>')
def handler(param1, param2 = None)
    pass
```

Filtros en el enrutamiento

- **int**

```
@get('/<param:int>')
def handler(param):
    Pass
```

- **float**

```
@get('/<param:float>')
def handler(param):
    pass
```

- **re (expresiones regulares)**

```
@get('/<param:re:^[a-z]+$>')
def handler(param):
    pass
```

API REST

```
from bottle import request, response
from bottle import post, get, put, delete

_names = set() # the set of names

@post('/names')
def creation_handler():
    '''Handles name creation'''
    pass

@get('/names')
def listing_handler():
    '''Handles name listing'''
    pass

@put('/names/<name>')
def update_handler(name):
    '''Handles name updates'''
    pass

@delete('/names/<name>')
def delete_handler(name):
    '''Handles name deletions'''
    pass
```

El enrutamiento en bottle se realiza
Mediante decoradores.
El decorador route es equivalente a estos 4
Pero estos ya son específicos de cada operación

EL API SE IMPLEMENTA EN UN SCRIPT SEPARADO.

Y EN OTRO SCRIPT SE PONE EN MARCHA LA APP.

Decoradores

- @get Listar Recursos
- @post Crear Nuevos Recursos
- @put Actualizar Recursos
- @delete Borrar Recursos

- **Ver ejemplo3 de rest**

Generación de Errores

- 404: Recurso no encontrado.
- Una función del servicio puede generar un error Http con la función:

  return HTTPError(404, “mensaje”)

- 500: Error interno en el servidor.

  return HTTPError(500, “mensaje”)

  – Por ejemplo, se produce un error, al borrar un registro, y lanzamos el error 500.

Estructura App

script_principal.py

api <DIR>

names.py (***Se implementa el API***)

- **En el script principal:**

```
import bottle
from api import names

app = application = bottle.default_app()

if __name__ == '__main__':
    bottle.run(host = '127.0.0.1', port = 8000)
```

Enlaces

- Documentación **bottle**:
  http://bottlepy.org/docs/dev/

- **Ejemplos**:
  - https://www.toptal.com/bottle/building-a-rest-api-with-bottle-framework

  - https://www.simplifiedpython.net/python-rest-api-example/

- Para probar el API se puede utilizar:
  - **postman**
    - Instalar en el equipo.
    - Para entrar podemos utilizar una cuenta de gmail.