Servicios Web REST

Tópicos Selectos de Computación I

Introducción

- REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.
- Creado en por Roy Fielding en 2000.
- REST: Representacional State Transfer.

Arquitectura

- Diseños fundamentales:
  - Se utiliza un protocolo cliente/servidor sin estado.
  - Peticiones y respuestas a través de operaciones bien definidas (POST, GET, PUT, DELETE).
  - Representación de recursos con una sintaxis universal. Los recursos son direccionables por un URIs.
  - El uso de hipermedios para la información de la aplicación como para las transacciones de estados de la aplicación.

Recursos en REST

- Llamados también recursos de información. Son accedidos a través de un identificador global (URI).
- Para la manipulación de los recursos clientes y servidores se comunican a través de un interfaz estándar (HTTP) e intercambian representaciones de estos recursos (archivos descargados/enviados).
- La información es transmitida por cualquier numero de conectores (p.e. clientes, servidores, túneles, etc.)

Recursos en REST

- Los verbos HTTP no proporcionan ningún recurso estándar para descubrir recursos.

- No hay ninguna operación LIST o FIND en HTTP, que se corresponderían con las operaciones list*() y find*() en el ejemplo RPC.

REST - Representación

- La iniciativa OpenSearch intenta estandarizar las búsquedas usando REST estableciendo especificaciones para descubrir recursos y un formato genérico para utilizar con sistemas basados en REST, incluyendo RDF, Atom, RSS y XML con XLink para gestionar los enlaces.

REST - Protocolos

- Los recursos tienen su propio identificador:
  - http://www.example.org/locations/mx/ver/xalapa_city.
- Los clientes trabajarían con estos recursos a través de las operaciones estándar de HTTP, como GET para descargar una copia del recurso.
- Cada objeto tiene su propia URL y puede ser manejado en cache. También puede ser copiado y guardado como marcador.
- POST se utiliza por lo general para acciones con efectos laterales, como enviar una orden de compra o añadir ciertos datos a una colección.

REST - Operaciones

- En RPC, se pone el énfasis en la diversidad de operaciones del protocolo, o verbos; por ejemplo una aplicación RPC podría definir operaciones como:
  - sumar()
  - saludar()

REST - Operaciones

- Una aplicación REST podría definir los siguientes tipos de recursos:
  - Usuario {}
  - Localización {}

REST - Ejemplos

- Ejemplo: World Wide Web.
- GET
  - https://www.googleapis.com/buzz/v1/activities/userId/@self/activityId
  - http://www.example.org/locations/us/ny/new_york_city
  - http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=facultad%20de%20estadistica%20e%20informatica
- Por ejemplo, para actualizar la localización de un usuario, un cliente REST podría descargar primero el registro XML usando GET y después modificar el archivo para cambiar la localización y subirlo al servidor utilizando el método HTTP PUT.
  - <usuario>
    - <nombre>María Juana</nombre>
    - <sexo>mujer</sexo>
    - <localización href="http://www.example.org/locations/us/ny/new_york_city">
      - Nueva York, NY, US
    - </localización>
  - </usuario>

REST - Ejemplos

- Los grandes proveedores de Web 2.0 están migrando a esta tecnología, p.e.:
  - Yahoo, Google y Facebook,
- Tales proveedores marcaron como obsoletos sus servicios SOAP y WSDL argumentando que utilizan un modelo orientado a los recursos más sencillo de utilizar.

REST - Ejemplos

- No consistente, poco elegante:

```
GET /agregarUsuario?nombre=Juan HTTP/1.1
Host: 127.0.0.1
```

- Petición REST:

```
POST /usuarios HTTP/1.1
Host: 127.0.0.1
Content-type: application/xml

<usuario>
    <nombre>Juan</nombre>
</usuario>
```

REST - Ejemplos

- Petición REST para obtener un recurso:

```
GET /usuarios/Juan HTTP/1.1
Host: 127.0.0.1
```

- No adecuado:

```
GET bucarUsuario?nombre=Juan HTTP/1.1
Host: 127.0.0.1
```

REST - Ejemplos

- Petición REST para actualizar un recurso:

```
PUT /usuarios/Juan HTTP/1.1
Host: 127.0.0.1
Content Type: application/xml

<usuario>
    <nombre>Ana</nombre>
</usuario>
```

- No adecuado:

```
GET actualizarUsuario?nombre=Juan&nuevo=Ana HTTP/1.1
```

REST - Ejemplos

- Los *Servidores Web* están diseñados para responder a peticiones HTTP GET con la búsquedas de recursos a través de una petición.

- Con HTTP GET se devuelve una representación de respuesta y no debería añadir un registro a la base de datos.

REST - Ventajas

- Los servidores pueden ser más sencillos debido a que la comunicación no tiene estados.
- Mayor eficiencia gracias a mensajes menores y al uso de caches.
- Cualquier navegador sirve como cliente.
- Uso de interfaces uniformes.

REST - Ventajas

- Mayor compatibilidad a largo plazo.
- No requiere un mecanismo especial para el descubrimiento.

REST - Desventajas

- El desarrollador tiene un cambio de perspectiva.
  - Se utilizan recursos en vez de llamadas a métodos.