Comunicación HTTP

La generación de servicios basados en HTTP tiene dos perspectivas: **Cliente** y **Servidor**. Nosotros vamos a ver únicamente la parte del Cliente, es decir, la que nos permite generar aplicaciones capaces de conectarse con servicios HTTP.

Peticiones basadas en HttpURLConnection

Hasta la **versión 1.8 de Java**, la clase HttpURLConnection ha sido la base de la programación de aplicaciones capaces de acceder a recursos en la web.

Los pasos para crear una petición HTTP sencilla son:

1) Crear la URL del recurso objetivo
2) Abrir la conexión
3) Configurar la conexión
    a. El método HTTP que usamos: Get, Post, Put, Delete
    b. Tipo de contenido
    c. Sistema de codificación
    d. Agente de usuario a usar
4) Obtención y evaluación del **código de respuesta HTTP**.
    a. Si es 200, obtener InputStream para leer y el OutputStream para escribir. Desconectar.
    b. Si no es 200, tratar el código de respuesta (que es un error).

Tienes un ejemplo en la clase **AccesoRAE**

Peticiones basadas en java.net.Http

A partir de **Java 11**, el paquete http ofrece una forma más potente, sencilla y actualizada de realizar peticiones HTTP.

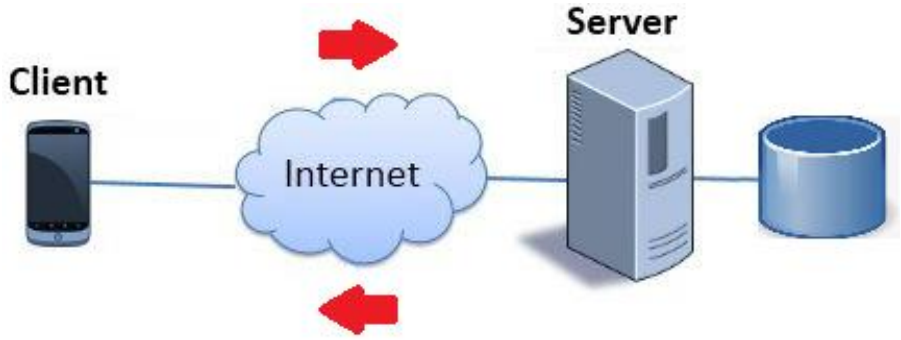
Los pasos para crear una petición HTTP son:

1) Crear un HttpClient
2) Crear un HttpRequest indicando la URI y otros parámetros de cabecera de la request
3) Realiza la petición mediante send () y obtener la respuesta.
4) Procesar la respuesta.

Tienes un ejemplo en la clase **AccesoRAEHttp**

Seamos prácticos: HTTP y REST

REST significa **Transferencia de Estado Representacional.** Hablar de REST es hablar de un tipo de **Arquitectura** en concreto. Actualmente, REST en un sentido muy amplio se refiere a tener un Servidor que es capaz de realizar operaciones sobre una **Base de Datos** (normalmente), usando **HTTP** para la comunicación y pasando información entre Cliente / Servidor mediante **XML, JSON**, etc.



Una explicación elemental del funcionamiento de un servicio REST sería la siguiente:

- El **Servidor** tiene una serie de métodos expuestos al exterior. Esto significa que un **Cliente** puede ejecutar esos métodos como si fuese su propio código.
- Estos métodos están mapeados, de forma que cada uno se corresponde con una Request o petición HTTP diferente. Por ejemplo, esto sería una invocación a uno de esos métodos:

  http://localhost:8080/PruebasRestFull/rest/hello/javatpoint

- Cuando un **Cliente** lanza esta Request (petición HTTP) el método asociado en el lado del servidor se ejecuta, genera un resultado y lo devuelve al cliente.
- Los mensajes que viajan por Internet son texto plano (XML, JSON), por tanto, será necesario hacer conversiones de datos.
- Las **respuestas** del Servidor son **códigos de respuesta HTTP**.

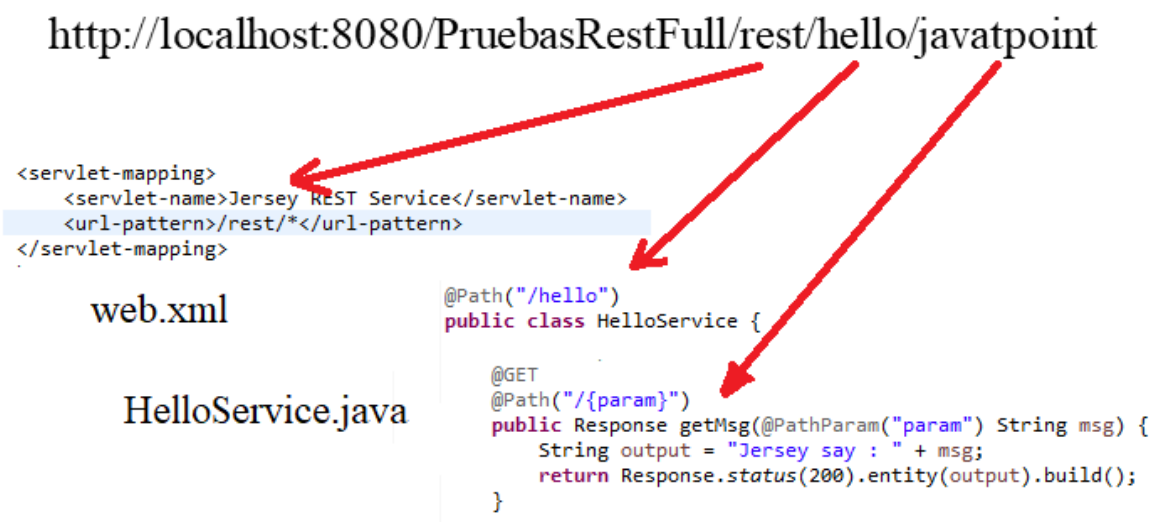
Vamos a ver cómo hace una **request** (petición) para encontrar el método del Servidor. Tomemos por ejemplo la siguiente URL:

http://localhost:8080/PruebasRestFull/rest/hello/javatpoint

Si la desglosamos, tenemos lo siguiente:

- **Http://** → Protocolo de comunicación que se está usando.
- **Localhost:8080** → La IP y el Puerto del Servidor destino.
- **PruebasRestFull** → El proyecto, el **recurso**, el nombre del servicio que queremos invocar. Esto le permite al **Servidor** diferenciar el recurso al que nos estamos refiriendo. Los Servidores normalmente tienen docenas de estos servicios.
- **Rest** → Mapeo del Servlet. Reconoce lo que viene detrás como llamadas a métodos REST
- **Hello** → El path de la clase. Esto es configurable en cada proyecto, clase o método mediante el uso de Anotaciones.
- **Javapoint** → Es un atributo que le pasamos al Server, en este caso un String.
- NO puede verse simplemente con la URL, pero esta Request se corresponde a un **GET**. Lanzamos una petición sobre un servicio que ejecuta una Select sobre Base de Datos, y nos retorna el resultado.

Esta URL en el lado **Servidor** se desgrana de la siguiente forma:



El Protocolo HTTP

HTTP es un protocolo orientado a transacciones que sigue el esquema petición-respuesta entre un Cliente y un Servidor. El **Cliente** realiza una petición enviando un **Mensaje** con cierto formato al **Servidor**, que envía siempre un **mensaje de respuesta**. Estos **mensajes HTTP** son texto plano. En cada una de las peticiones se incluye también la acción requerida por el servidor, la URL del recurso y la versión del HTTP. En las respuestas se incluye la versión del HTTP, un código de respuesta, y la propia respuesta del servidor. Como vemos, HTTP es un protocolo bastante simple.

Cada **Mensaje** internamente está dividido en dos partes:

- Una **cabecera** con metadatos.
- Un **cuerpo** del mensaje, que es opcional. Típicamente tiene los datos que se intercambian entre el Cliente y Servidor.

Las acciones que tenemos disponibles son varias. Las más conocidas, sobre todo si has desarrollado web, son:

- **GET:** Se usan los GET para solicitar el Servicio especificado. Esto quiere decir que un GET se usa para solicitar o recuperar datos de un Servidor (su BBDD).

- **POST:** Envía datos a un Servicio. Esto puede resultar en la creación de un nuevo recurso o en la actualización de uno existente en el Servidor (su BBDD).

Pero no son las únicas acciones, como por ejemplo DELETE, PUT, HEAD, LOCK…

Existe una equivalencia entre las acciones y las operaciones que pueden hacerse sobre una BBDD. Por ejemplo, un GET equivale a una SELECT, un POST equivale a una INSERT, un DELETE es un DELETE, etc. Aunque realmente puedes no seguir esta norma, dado que tú diseñas la comunicación como prefieras, es recomendable que no reinventes la rueda. Así que no, que le llegue un GET a un Servidor no significa que vaya a ejecutarse un SELECT automáticamente. Eso **lo configuras** tú.

Por defecto, se toma:

HTTP	**SQL**
GET	Select
POST	Insert
PUT	Update
DELETE	Delete

No todas las operaciones HTTP permiten el envío de respuestas completas del Servidor.

Respuestas HTTP

Las **respuestas** de un Servidor se hacen siempre mediante códigos de respuesta. Esencialmente, son números que indican lo que ha pasado con la petición.

- **Códigos con formato 1xx**: Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- **Códigos con formato 2xx**: Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- **Códigos con formato 3xx**: Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- **Códigos con formato 4xx**: Errores causados por el cliente. Indica que ha habido un error en el procesado de la petición a causa de que el cliente ha hecho algo mal.
- **Códigos con formato 5xx**: Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

Corresponde a los programadores del lado Servidor decidir cuándo se envía cada código. De la misma forma, corresponde a los programadores del lado Cliente decidir cómo se trata cada uno de esos códigos HTTP.

Varios de los posibles códigos de respuesta más utilizados son:

100 Continue. Todo hasta ahora está bien y el cliente debe continuar con la solicitud o ignorarla

200 Ok. La solicitud ha tenido éxito. Si era por ejemplo un:

GET: El recurso se ha obtenido y se transmite en el cuerpo del mensaje

PUT o POST: El recurso que describe el resultado de la acción se transmite en el cuerpo del mensaje.

201 Created. Se ha creado un nuevo recurso. Respuesta típica de PUT

301 Moved Permanently. La URI del recurso ha cambiado. La nueva URI se devolverá en la respuesta si se conoce.

400 Bad Request. El server no entiende, sintaxis inválida

401 Unauthorized. Tienes que autenticarte.

403 Forbidden. No tienes permiso de acceso

404 Not Found. No podemos encontrar el recurso.

Mensajes HTTP

Un Mensaje HTTP puede ‘verse’ circulando por la red de la siguiente manera:

- Mensaje del Cliente

```
GET /index.html HTTP/1.1
Host: www.example.com
Referer: www.google.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Connection: keep-alive
[Línea en blanco]
```

- Mensaje de respuesta del server (en este caso un HTML)

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221

<html lang="eo">
<head>
<meta charset="utf-8">
<title>Título del sitio</title>
</head>
<body>
<h1>Página principal de tuHost</h1>
(Contenido)
  .
  .
  .
```