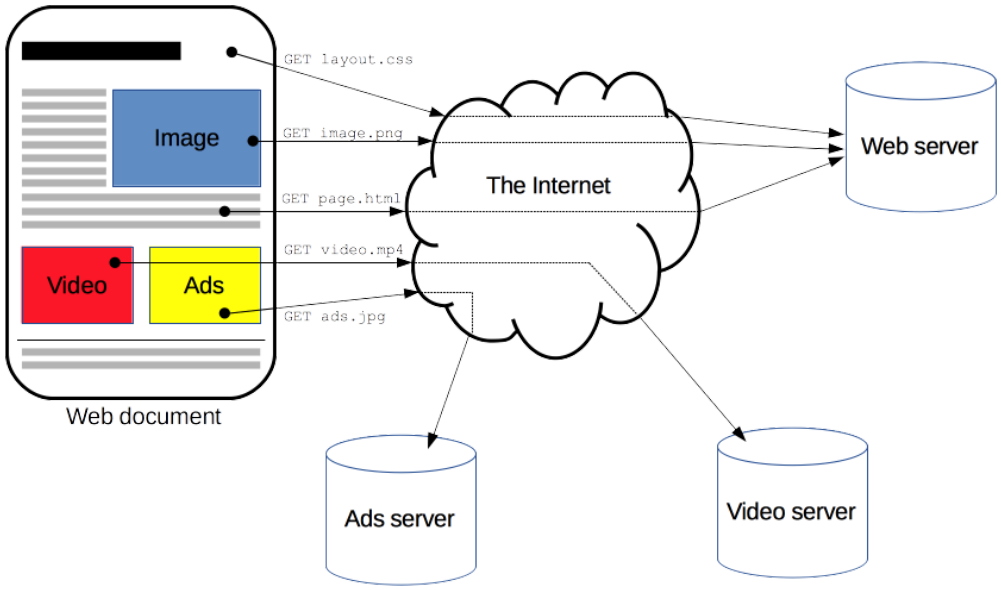
Spring Framework

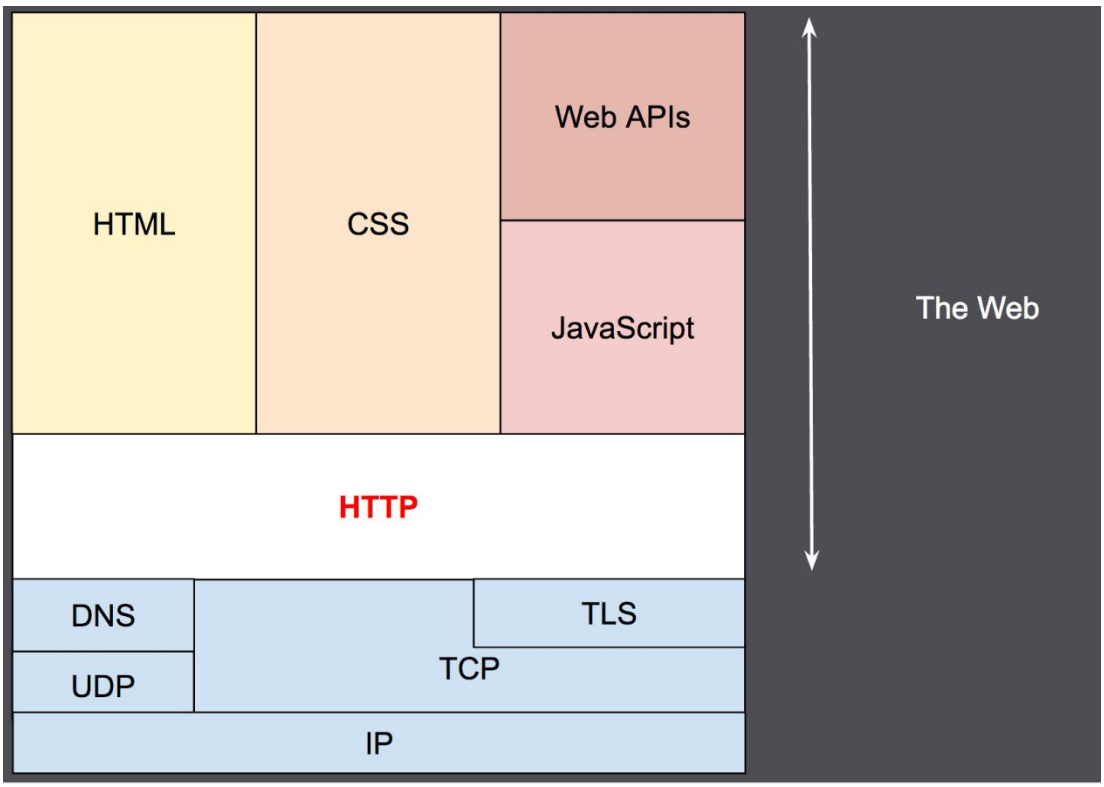
Entendiendo el protocolo HTTP

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual permite realizar una petición de datos o recursos, como pueden ser documentos HTML, Archivos PDF, imágenes, etc. Es la base de cualquier intercambio de datos en la Web y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada, por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así una página web completa, resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de la página web (CSS), el texto, las imágenes, vídeos, scripts.



Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que

utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman peticiones (Request), y los mensajes enviados por el servidor, se llaman respuestas. (Response)



Diseñado a principios de los 90, HTTP es un protocolo ampliable, que ha ido evolucionando con el tiempo. Es lo que se conoce como un protocolo de la capa de aplicación y se transmite sobre el protocolo TCP, o el protocolo encriptado TLS. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto (HTML), sino que además, se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. HTTP puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas Web en el acto.

Características clave del protocolo HTTP

HTTP es sencillo

Está pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más fácil la depuración de errores, y reduciendo la curva de aprendizaje para las personan que empieza a trabajar con él.

HTTP es extensible

Presentadas en la versión HTTP/1.0, las cabeceras de HTTP, han hecho que este protocolo sea fácil de ampliar y

de experimentar con él. Funcionalidades nuevas pueden desarrollarse, sin más que un cliente y su servidor, comprendan la misma semántica sobre las cabeceras de HTTP.

HTTP es un protocolo con sesiones, pero sin estados

HTTP es un protocolo sin estado, es decir: no guarda ningún dato entre dos peticiones en la misma sesión. Esto plante la problemática, en caso de que los usuarios requieran interactuar con determinadas páginas Web de forma ordenada y coherente, por ejemplo, para el uso de "cestas de la compra" en páginas que utilizan en comercio electrónico. Pero, mientras HTTP ciertamente es un protocolo sin estado, el uso de HTTP cookies, si permite guardar datos con respecto a la sesión de comunicación. Usando la capacidad de ampliación del protocolo HTTP, las cookies permiten crear un contexto común para cada sesión de comunicación.

HTTP y conexiones

Una conexión se gestiona al nivel de la capa de trasporte, y por tanto queda fuera del alcance del protocolo HTTP. Aún con este factor, HTTP no necesita que el protocolo que lo sustenta mantenga una conexión continua entre los participantes en la comunicación, solamente necesita que sea un protocolo fiable o que no pierda mensajes (como mínimo, en todo caso, un protocolo que sea capaz de detectar que se ha pedido un mensaje y reporte un error). De los dos protocolos más comunes en Internet, TCP es fiable, mientras que UDP, no lo es. Por lo tanto, HTTP se apoya en el uso del protocolo TPC, que está orientado a conexión, aunque una conexión continua no es necesaria siempre.

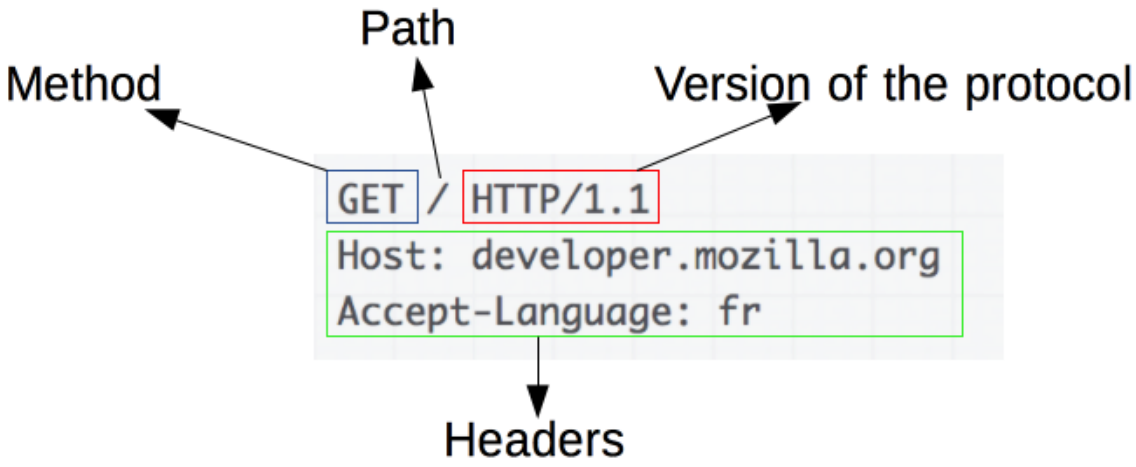
Mensajes HTTP

Los mensajes están estructurados en un formato binario y las tramas permiten la compresión de las cabeceras y su multiplexación. Así pues, incluso si solamente parte del mensaje original en HTTP se envía en este formato, la semítica de cada mensaje es la misma y el cliente puede formar el mensaje original en HTTP/1.1. Luego, es posible interpretar los mensajes HTTP/2 en el formato de HTTP/1.1.

Existen dos tipos de mensajes HTTP: peticiones y respuestas, cada uno sigue su propio formato.

Peticiones (Request)

Un ejemplo de petición HTTP:



Una petición de HTTP, está formada por los siguientes campos:

- Un método HTTP, normalmente pueden ser un verbo, como: GET, POST o un nombre como: OPTIONS o HEAD, que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o presentar un valor de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones.

- La dirección del recurso pedido; la URL del recurso, sin los elementos obvios por el contexto, como pueden ser: sin el protocolo (http://), el dominio (aquí developer.mozilla.org), o el puerto TCP (aquí el 80).

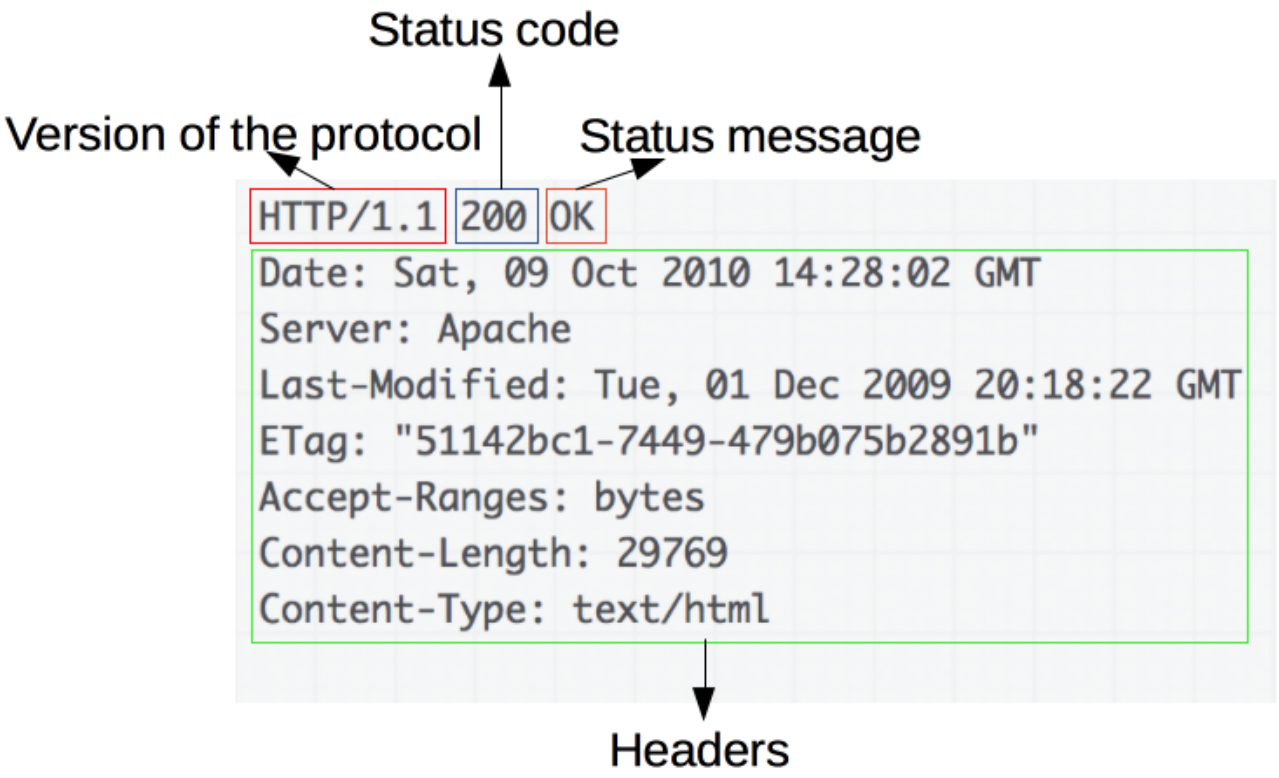
- La versión del protocolo HTTP.

- Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.

- un cuerpo de mensaje, en algún método, como puede ser POST, en el cual envía la información para el servidor.

Respuestas (Response)

Un ejemplo de respuesta:



Las respuestas están formadas por los siguientes campos:

- La versión del protocolo HTTP que están usando.
- Un código de estado, indicando si la petición ha sido exitosa, o no.
- Un mensaje de estado, una breve descripción del código de estado.
- Cabeceras HTTP, como las de las peticiones.
- Opcionalmente, el recurso que se ha pedido.

Spring MVC

Configuración

Spring tiene sus propios módulos que conforman un framework para el desarrollo de aplicaciones Java basadas en web, Este framework sigue el patrón de diseño MVC. A este conjunto de módulos se le conoce con el nombre de Spring MVC.

Al igual que Struts 2, Spring MVC es un framework de capa de presentación, basado o dirigido por peticiones.

Esto quiere decir que los programadores estructuramos el diseño de la aplicación web en términos de métodos que manejan peticiones HTTP.

El framework define una serie de interfaces que siguen el patrón de diseño Strategy para todas las responsabilidades que deben ser manejadas por el framework. El objetivo de cada interface es ser simple y clara, para que sea fácil para los usuarios de Spring MVC crear nuestras propias implementaciones.

La parte central de Spring MVC es un componente llamado el "DispatcherServlet", el cual sigue el patrón de diseño front controller, este envía las peticiones a los componentes designados para manejarlas.

Las interfaces más importantes definidas por Spring MVC son:

- Controller: Se encuentra entre el Modelo y la Vista, maneja las peticiones que entran y redirige a las respuestas apropiadas.

- HandlerAdapter: Es quien realiza la invocación del manejador de la petición (el DispatcherServlet delega esa tarea a este componente), esto incluye inyectar los parámetros adecuados usando reflexión.

- HandlerInterceptor: Intercepta las peticiones de entrada. Es similar, pero no igual, a los filtros en el API de Servlets.

- HandlerMapping: Selecciona objetos que manejan las peticiones de entrada basado en cualquier atributo o condición interna o externa a esos atributos.

- LocaleResolver: Resuelve y opcionalmente guarda el Locale de un usuario individual.

- MultipartResolver: Facilita trabajar con la carga de archivos, envolviendo las peticiones de entrada.

- View: Responsable de regresar una respuesta al cliente. Algunas peticiones pueden ir directo a la vista sin pasar por el modelo.

- ViewResolver: Selecciona una vista basado en un nombre lógico para la vista.

Cada una de las interfaces anteriores tiene una responsabilidad importante dentro del framework y la abstracción ofrecida por estas interfaces permite tener variaciones en sus implementaciones. Spring MVC contiene implementaciones de casi todas estas, las cuales están construidas sobre el API de Servlets.

Un principio clave en Spring MVC es el principio de diseño "Abierto / Cerrado" o, por su nombre en inglés, "Open for extension, closed for modification". Algunos métodos en las clases core de Spring MVC están marcados como final

para que los desarrolladores no puedan sobre-escribirlos y proporcionar un nuevo comportamiento, de esta forma el framework garantiza que su comportamiento básico no puede ser modificado.

El "DispatcherServlet" es la pieza central de Spring MVC. Spring MVC como muchos otros frameworks orientado a peticiones, está diseñado alrededor de un Servlet central que despacha las peticiones a los controladores y ofrece otra funcionalidad que facilita el desarrollo de aplicaciones web. Sin embargo, el "DispatcherServlet" hace más que sólo eso, se encuentra completamente integrado con el contenedor de IoC de Spring, lo cual nos permite usar el resto de características de Spring.

Flujo de navegación del "DispatcherServlet"

1. El cliente hace una petición a la aplicación web, esta petición llega al DispatcherServlet.

2. El DispatcherServlet determina qué componente debe atender la petición y la envía a este.

3. El Controller implementa la lógica específica para responder la petición, para lo que puede hacer uso de cualquier recurso que esté al alcance de cualquier aplicación Java (incluyendo conexiones con servicios web o con base de datos).

4. Una vez que el Controller termina su proceso, regresa la petición al DispatcherServlet, estableciendo los datos adecuados del modelo e indicando el nombre lógico de la vista que debe regresarse al cliente y un modelo lleno con los nombres y valores de los atributos que se usarán para generar la vista final.

5. En base al nombre lógico regresado por el Controller, el DispatcherServlet usa un ViewResolver para determinar qué recurso debe utilizar para generar la vista final que se mostrará al usuario, este recurso puede ser una JSP, una página HTML, un template de Velocity, un archivo de Excel, un PDF, etc.

6. El DispatcherServlet obtiene la vista que será regresada al cliente.

7. El DispatcherServlet finalmente regresa la vista adecuada al cliente.

A partir del paso 5 se indica qué componente y de qué forma se resolverá la vista (en este caso una página) debe regresarse en respuesta a la petición que haya hecho el usuario. Para esto Spring tiene un componente llamado "ViewResolver", el cual se encarga de, en base a un nombre lógico, identificar el recurso (o página) al cual se está haciendo referencia.

La resolución de vistas en Spring es muy flexible. Un Controller es típicamente responsable de preparar el modelo con datos y seleccionar el nombre de la vista que será regresada al usuario, sin embargo, también podría escribir la respuesta directamente en el stream de respuesta. La resolución del nombre de la vista es altamente configurable y se puede integrar con representación basada en tecnologías de templates como JSP, Velocity y Freemarker, o generar directamente XML, JSON, Atom, y muchos otros tipos de contenido.

Spring MVC tiene varias implementaciones de ViewResolver, como las que se muestran a continuación:

ViewResolver	Descripción
`AbstractCachingViewResolver`	Es un `ViewResolver` abstracto que coloca vistas en caché. Algunas veces las vistas necesitan cierta preparación antes de que puedan ser usadas; extendiendo esta clase proporciona ese caché.
`XmlViewResolver`	Implementción de `ViewResolver` que acepta un archivo de configuración en `XML` con los mismos `DTDs` que los bean factories de `Spring`. El archivo de configuración por default es "`/WEB-INF/views.xml`"
`ResourceBundleViewResolver`	Implementación de `ViewResolver` que usa definiciones de beans en un `ResourceBundle`, especificado por el nombre base del bundle. Tipicamente definimos el bundle en un archivo de propiedades, localizado en el `classpath`. El nombre por default es "`views.properties`"
`UrlBasedViewResolver`	Implementación simple de `ViewResolver` que realiza la resolución directa de nombres de vistas lógicas a `URLs`, sin una definición explícita de mapeo. Es apropiado si nuestros nombres lógicos coinciden con los nombres de los recursos de vista de forma directa, sin la necesidad de mapeos adicionales.
`InternalResourceViewResolver`	Subclase de `UrlBasedViewResolver` que soporta vistas basadas en `JSPs`. Tiene otras subclases útiles como `JstlView` y `TilesView`.
`VelocityViewResolver / FreeMarkerViewResolver`	Subclase de `UrlBasedViewResolver` soporta vistas creadas usando templates creados usando `Velocity` o `Freemarker`, respectivamente, y sublclases propias de estos.
`ContentNegotiatingViewResolver`	Implementación de `ViewResolver` que resuelve una vista basada en la petición para determinar el nombre del archivo de la vista, o en la cabecera "`Accept`".