Features of the framework for building RESTful Service

Java Backend Developer I

Objetivos

- Comprender el concepto de Spring Framework, módulos y proyectos.
- Conocer los conceptos fundamentales de Spring core
- Conocer a fondo los conceptos relacionados a Spring MVC bajo el concepto de DispatcherServlet
- Conocer los tipos de comunicaciones entre cliente y servidor.

Agenda

- Revisión de Spring Framework, concepto, módulos y proyectos
- Revisión del Módulo Spring Core:
  - Características y beneficios.
  - Paradigma inversión de control e inyección de dependencia.
  - Concepto de beans y sus ámbitos
- Revisión del Modulo Spring MVC:
  - Concepto de DispatcherServlet
  - Concepto de Annotated Controllers
  - Concepto de Functional Endpoints
  - Concepto de URI Links
  - Concepto de HTTP Caching
  - Concepto de HTTP/2
  - Concepto de WebSockets

Spring Framework

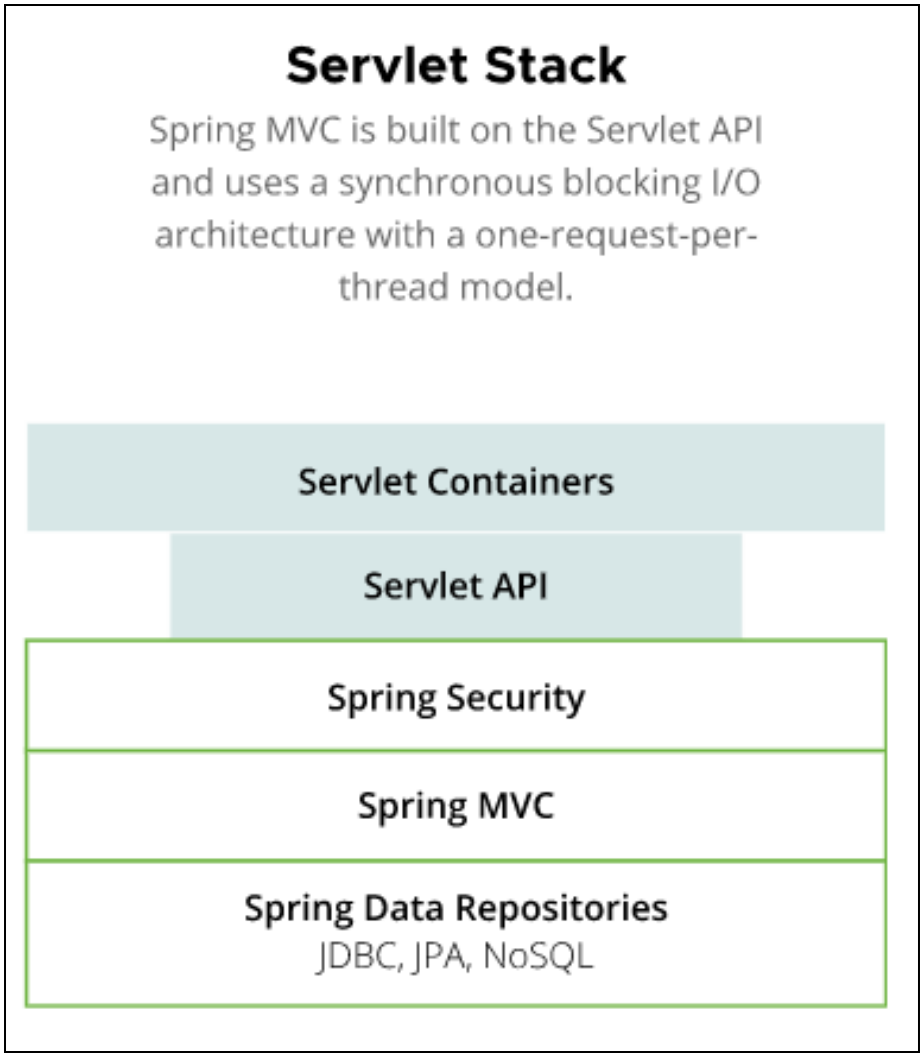
Concepto

Spring es un marco de trabajo que facilita el desarrollo de aplicaciones Java. Lo esencial de Spring es el manejo de la infraestructura de las aplicaciones y permitir al desarrollador que solo se encargue de programar la lógica de negocio usando objetos simples de Java, Groovy o Kotlin.

El término “Spring” puede utilizarse para referirse al proyecto Spring Framework, marco de trabajo que es un todo.

Spring Framework está dividido en módulos. Las aplicaciones pueden elegir los módulos que necesitan. Ellos incluyen un modelo de configuración y un mecanismo de inyección de dependencia. Con el tiempo, otros proyectos se han construido por encima del Framework Spring.

Spring MVC sobre Servlet



Spring Framework Proyectos

Microservices

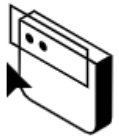
Quickly deliver production-grade features with independently evolvable microservices.

Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.

Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.

Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.

Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.

Batch

Automated tasks. Offline processing of data at a time to suit you.

Spring Framework

Características

A que da Soporte

Spring Framework proporciona soporte fundamental para diferentes arquitecturas de aplicaciones, incluyendo mensajería, datos transaccionales y persistencia, y web.

Este Framework tiene un manejo de peticiones de forma bloqueante con el uso de servlets y no bloqueante con el uso de Spring WebFlux

Lenguajes

Spring facilita la creación de aplicaciones Java del tipo empresarial. Nos proporciona todo lo necesario más allá del propio lenguaje Java. También proporciona extensiones y soporte para aplicaciones creadas en Groovy y Kotlin.

Spring Framework

Características

Donde se ejecuta

En Servidor de Aplicaciones

Spring soporta una amplia gama de escenarios de aplicación. En una empresa grande, a menudo existen muchas aplicaciones durante mucho tiempo y tienen que ejecutarse en un JDK y en un servidor de aplicaciones.

En Servidor Embebido

Otras aplicaciones pueden correr como un solo *jar* con el servidor embebido, posiblemente en un ambiente de la nube. Sin embargo, otras pueden ser aplicaciones independientes (como lotes o cargas de trabajo de integración) que no necesitan un servidor.

Spring Framework

Características

Código Abierto que permite la evolución

Spring es de código abierto. Tiene una comunidad grande y activa que proporciona una retroalimentación continua basada en una amplia gama de casos de uso en el mundo real. Esto ha ayudado a Spring a evolucionar con éxito durante un tiempo muy largo.

Inyección de Dependencia y Anotaciones

Spring Framework también soporta las especificaciones de Inyección de dependencia y Anotaciones comunes, que los desarrolladores de aplicaciones pueden elegir utilizar en lugar de los mismos mecanismos proporcionados por Spring Framework.

Spring Framework

Características

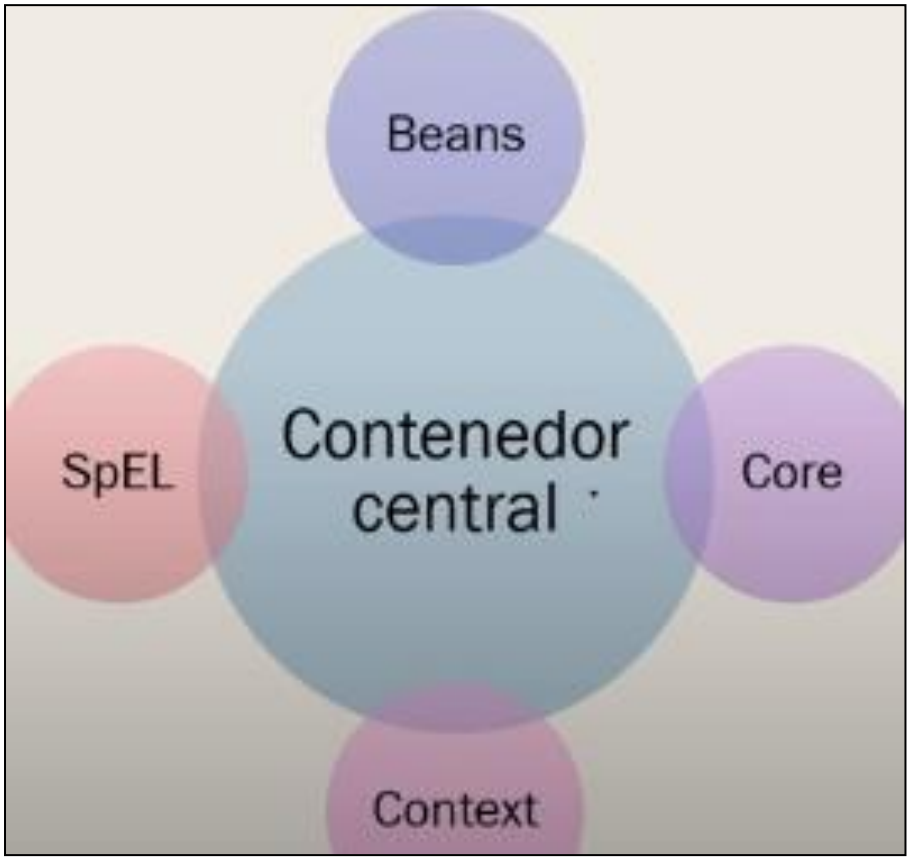
Evoluciona

En los primeros días de Java EE y Spring, se crearon aplicaciones para implementarlas en un servidor de aplicaciones. Hoy en día, con la ayuda de Spring Boot, las aplicaciones se crean de una forma amigable a los devops y la nube, con el contenedor de servlets embebido y fácil de cambiar.

Innovando y evolucionando en proyectos

Más allá del Spring Framework, hay otros proyectos, como Spring Boot, Spring Security, Spring Data, Spring Cloud, Spring Batch, entre otros. En spring.io/projects se puede ver la lista completa de proyectos Spring.

Módulo Spring Core



Permite

- Utiliza el paradigma de la inversión de control
- La creación de Beans a través de la inyección de dependencia.
- El Uso de contexto como almacén de los Bean en memoria.
- El uso de Spring Expresion Language (SpEL)

Spring Core

El core de **Spring** se basa en el concepto de **Inversión de Control** (**IoC** por sus siglas en inglés), más específicamente en la **Inyección de Dependencias** (**DI**).

Inversión de Control

Concepto Este es un proceso en el cual los objetos definen sus **dependencias** (o sea, los otros objetos con los que trabajan) solo a través de los argumentos de su constructor, argumentos a un método de factory, o métodos setter que son invocados después de que el objeto se ha construido.

Spring Core

Inversión de Control

Beneficios

Nos proporciona los siguientes beneficios:

- Crea los objetos por nosotros
- Maneja nuestros objetos
- Ayuda a que nuestra aplicación sea configurable.
- Manejo de dependencias.

El paradigma de inversión de control se aplica en Spring Core a través de la Inyección de Dependencia.

Spring Core

Inyección de Dependencias

Concepto

Se enfoca en el acoplamiento débil: los componentes de nuestra aplicación deben estar lo más separado posible acerca de otros componentes.

La forma más fácil de lograr este bajo acoplamiento en Java es mediante el uso de Interfaces.

Como cada componente de la aplicación solo está consciente de la interface de otros componentes, podemos cambiar la implementación del componente sin afectar a los componentes que usan el nuevo componente.

Permite el desarrollo a base de Beans (POJOs)

Spring Core

Inyección de Dependencia en Spring

En lugar de que el mismo objeto sea quien se encargue de instanciar, o localizar, las dependencias con las que trabaja (usando directamente su constructor o un localizador de servicios), **es el contenedor el que inyecta estas dependencias cuando crea al bean**. Este proceso, como podemos observar, es el inverso al que normalmente se hace, y de ahí el nombre de Inversión de Control (es el framework el que hace el trabajo, no el programador).

Spring Core

Spring Beans

Concepto

Cualquier objeto en Spring framework que inicializamos a través del contenedor Spring se llama Spring Bean. Cualquier clase Java POJO normal puede ser un Spring Bean si está configurada para inicializarse a través del contenedor al proporcionar información de metadatos de configuración.

Ámbito

Hay cinco ámbitos definidos para Spring Beans.

1.- singleton: solo se creará una instancia del bean para cada contenedor. Este es el alcance predeterminado para los beans de spring. Al usar este alcance, asegúrese de que bean no tenga variables de instancia compartidas, de lo contrario, podría generar problemas de inconsistencia de datos.

Spring Core

Spring Beans

Ámbito

2.- prototype: se creará una nueva instancia cada vez que se solicite el bean.

3.- request: es igual que el alcance del prototipo, sin embargo, está destinado a ser utilizado para aplicaciones web. Se creará una nueva instancia del bean para cada solicitud HTTP.

4.- session: el contenedor creará un nuevo bean para cada sesión HTTP.

5.- global-session: se utiliza para crear beans de sesión global para aplicaciones de portlet.

Spring Core

Spring beans

Configuración

Spring Framework proporciona tres formas de configurar beans para usar en la aplicación.

1.- Basada en anotaciones: mediante el uso de anotaciones @Service o @Component. Los detalles del alcance se pueden proporcionar con la anotación @Scope.

2.- Basada en XML: al crear un archivo XML de configuración de Spring para configurar los beans. Si está utilizando Spring MVC Framework, la configuración basada en xml se puede cargar automáticamente en el archivo web.xml.

3.- Basada en Java: a partir de Spring 3.0, podemos configurar Spring Beans utilizando programas Java. Algunas anotaciones importantes utilizadas para la configuración basada en Java son @Configuration, @ComponentScan y @Bean.

Spring Web MVC

DispatcherServlet

Con Spring MVC es muy sencillo habilitar un back-end rest para nuestra aplicación web.

REST

Concepto Es un estilo de arquitectura basado en la representación. Todo lo que accedemos en rest es una representación y cada representación tiene un identificador de recurso (URI) único. Estos son dos conceptos importantes que debemos recordar en rest.

Recuerde siempre poner la anotación @ResponseBody en el método al que se accederá utilizando rest URI.

Con Spring podemos evitar esta anotación y agregar la anotación @RestController a la clase de controlador en sí. Es una buena práctica usar esta anotación si va a escribir un nuevo Spring rest controlador.

Spring Web MVC

RESTful Web Service

Concepto

REST (Transferencia de estado representacional) es un estilo arquitectónico con el que los Servicios web pueden.

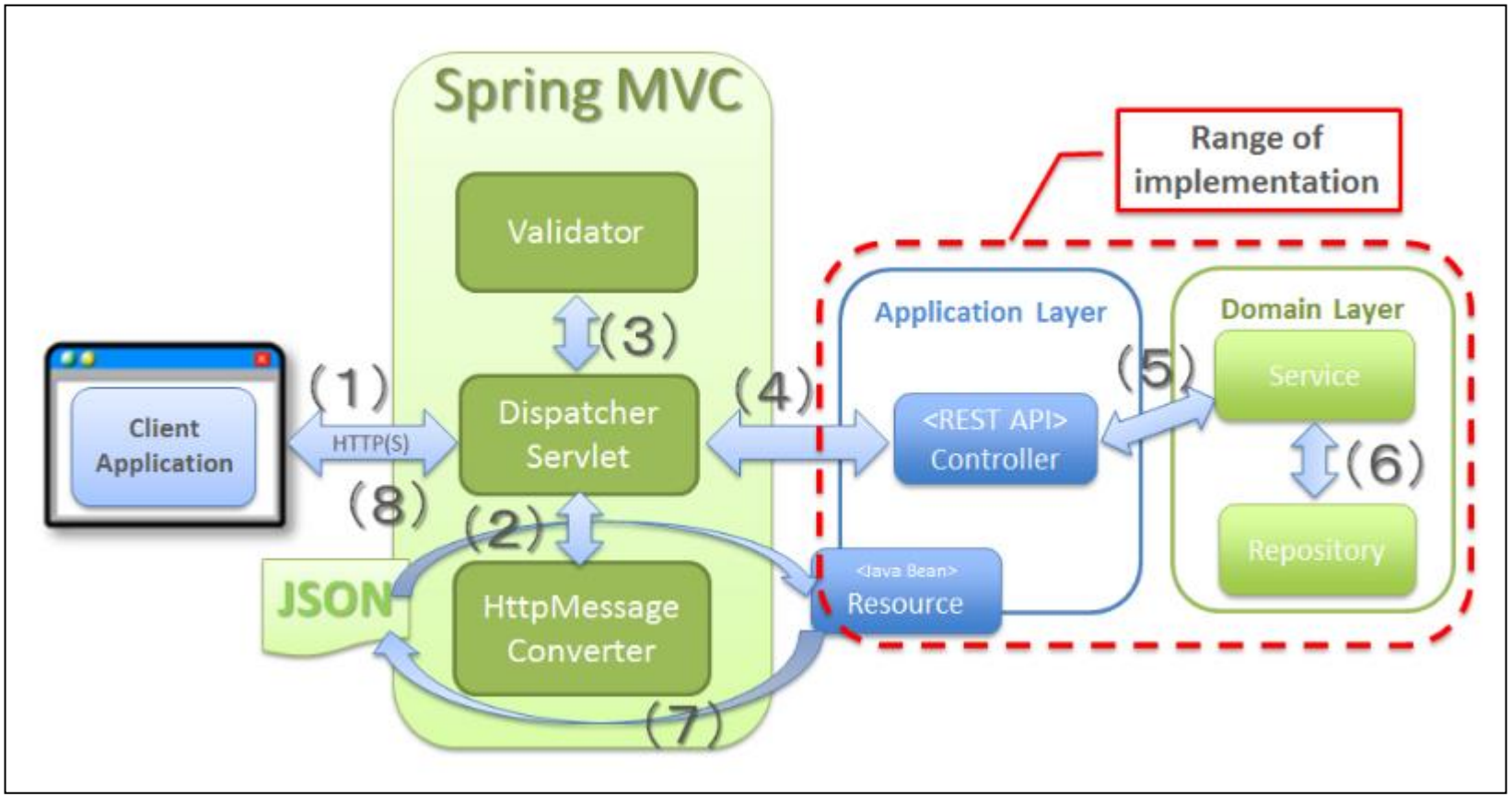
Un servicio web es una unidad de código administrado, que se puede invocar mediante solicitudes HTTP.

De manera simple para aquellos que son nuevos en el servicio web. Desarrolla la funcionalidad principal de su aplicación, la implementa en un servidor y la expone a la red. Una vez que se expone, se puede acceder mediante URI a través de solicitudes HTTP de una variedad de aplicaciones cliente. En lugar de repetir la misma funcionalidad en múltiples aplicaciones cliente (web, de escritorio y móviles), la escribe una vez y accede a ella en todas las aplicaciones.

Spring Web MVC

RESTful

Cuando el Servicio web RESTful se desarrolla utilizando Spring MVC, la aplicación se configura de la siguiente manera. La implementación es necesaria para la parte marcada con marco rojo.



Spring Web MVC

RESTful PASO A PASO

Spring MVC:

Paso1: Spring MVC recibe una solicitud del cliente y determina la API REST (método del controlador del controlador) que se llamará.

Paso2: Spring MVC convierte el mensaje en formato JSON especificado en la solicitud BODY en un objeto Resource mediante HttpMessageConverter.

Paso3: Spring MVC realiza la validación de entrada para el valor almacenado en el objeto Resource utilizando Validator.

Paso4: Spring MVC llama a la API REST. Aquí, el recurso que se ha convertido de JSON y para el cual se lleva a cabo la validación de entrada, se entrega a REST API.

Spring Web MVC

RESTful PASO A PASO

REST API:

Paso5: La API REST llama al método de servicio y realiza el proceso para DomainObject como Entity, etc.

Paso6: El método de servicio llama al método Repository y realiza el proceso CRUD para el DomainObject, como Entity, etc.

Paso7: Spring MVC convierte el objeto Resource devuelto por REST API en un mensaje en formato JSON, utilizando HttpMessageConverter.

Paso8: Spring MVC establece el mensaje en formato JSON en respuesta BODY y responde al cliente.

Spring Web MVC

Annotated Controllers

El controlador predeterminado se basa en las anotaciones @RestController y @RequestMapping, @GetMapping que ofrecen una amplia gama de métodos de manejo flexibles.

Ejemplo:

```
@RestController
public class EmployeeRestController {

        @Autowired
        private EmployeeRepository repository;

        @GetMapping("/rest/employee/get/{id}")
        public Employee getEmployeeByID(@PathVariable("id") int id) {
                return repository.retrieve(id);
        }
```

Spring Web MVC

Annotated Controllers

@RestController

Anotación que registra el controlador para Spring MVC. Agregada desde Spring Framework 4.0.

Debido a la anotación @RestController, no es necesario asignar la anotación @ResponseBody a cada método del Controlador.

Por lo tanto, es posible crear Controller para REST API de una manera simple.

```
@RequestMapping("members") // (1)
@RestController // (2)
public class MemberRestController {

    // omitted ...

}
```

Spring Web MVC

Annotated Controllers

@RequestMapping

Se usa para asignar URL como '/editPet' en una clase completa o un método en particular. Normalmente, la anotación asigna una ruta de solicitud específica (o patrón de ruta) en un controlador, con anotaciones de nivel de método adicionales que "reducen" la asignación primaria para un método de solicitud de método HTTP específico ("GET" / "POST") o Parámetros específicos de solicitud HTTP.

```
@RequestMapping("members") // (1)
@RestController // (2)
public class MemberRestController {

    // omitted ...

}
```

Spring Web MVC

Functional Endpoints

Concepto

Una solicitud HTTP se maneja con una HandlerFunction que toma ServerRequest y devuelve Mono <ServerResponse>, los cuales son contratos inmutables que ofrecen acceso amigable JDK-8 a la solicitud y respuesta HTTP. HandlerFunction es el equivalente de un método @RequestMapping en el modelo de programación basado en anotaciones.

Spring Web MVC

Functional Endpoints

Las solicitudes se enrutan a una HandlerFunction con una RouterFunction que toma ServerRequest y devuelve Mono <HandlerFunction>. Cuando una solicitud coincide con una ruta en particular, se utiliza la función Handler asignada a la ruta. RouterFunction es el equivalente de una anotación @RequestMapping.

```
import static org.springframework.http.MediaType.APPLICATION_JSON;
import static org.springframework.web.reactive.function.server.RequestPredicates.*;
import static org.springframework.web.reactive.function.server.RouterFunctions.route;

PersonRepository repository = ...
PersonHandler handler = new PersonHandler(repository);

RouterFunction<ServerResponse> route =
    route(GET("/person/{id}").and(accept(APPLICATION_JSON)), handler::getPerson)
        .andRoute(GET("/person").and(accept(APPLICATION_JSON)), handler::listPeople)
        .andRoute(POST("/person"), handler::createPerson);


public class PersonHandler {

    // ...

    public Mono<ServerResponse> listPeople(ServerRequest request) {
        // ...
```

Spring Web MVC

URLs Links

UriComponents

UriComponentsBuilder ayuda a crear URI a partir de plantillas de URI con variables, como muestra el siguiente ejemplo:.

Java | Kotlin

```
UriComponents uriComponents = UriComponentsBuilder
        .fromUriString("https://example.com/hotels/{hotel}") 1
        .queryParam("q", "{q}") 2
        .encode() 3
        .build(); 4

URI uri = uriComponents.expand("Westin", "123").toUri(); 5
```

1. Static factory method with a URI template.
2. Add or replace URI components.
3. Request to have the URI template and URI variables encoded.
4. Build a `UriComponents`.
5. Expand variables and obtain the `URI`.

Spring Web MVC

URLs Links

UriBuilder

UriComponentsBuilder implementa UriBuilder. Puede crear un UriBuilder, a su vez, con un UriBuilderFactory. Juntos, UriBuilderFactory y UriBuilder proporcionan un mecanismo conectable para construir URI a partir de plantillas de URI, basadas en una configuración compartida, como una URL base, preferencias de codificación y otros detalles.

Puede configurar RestTemplate y WebClient con UriBuilderFactory para personalizar la preparación de URI. DefaultUriBuilderFactory es una implementación predeterminada de UriBuilderFactory que usa UriComponentsBuilder internamente y expone las opciones de configuración compartidas.

El siguiente ejemplo muestra cómo configurar un RestTemplate:

Spring Web MVC

URLs Links

UriBuilder El siguiente ejemplo muestra cómo configurar un RestTemplate:

Java Kotlin

```
// import org.springframework.web.util.DefaultUriBuilderFactory.EncodingMode;

String baseUrl = "https://example.org";
DefaultUriBuilderFactory factory = new DefaultUriBuilderFactory(baseUrl);
factory.setEncodingMode(EncodingMode.TEMPLATE_AND_VALUES);

RestTemplate restTemplate = new RestTemplate();
restTemplate.setUriTemplateHandler(factory);
```

The following example configures a `WebClient`:

Java Kotlin

```
// import org.springframework.web.util.DefaultUriBuilderFactory.EncodingMode;

String baseUrl = "https://example.org";
DefaultUriBuilderFactory factory = new DefaultUriBuilderFactory(baseUrl);
factory.setEncodingMode(EncodingMode.TEMPLATE_AND_VALUES);

WebClient client = WebClient.builder().uriBuilderFactory(factory).build();
```

Spring Web MVC

HTTP Catching

Concepto

El almacenamiento en caché es la capacidad de almacenar copias de datos de acceso frecuente en varios lugares a lo largo de la ruta de solicitud-respuesta.

Cuando un consumidor solicita una representación de recursos, la solicitud pasa por un caché o una serie de cachés (caché local, caché de proxy o proxy inverso) hacia el servicio que aloja el recurso. Si alguna de las cachés a lo largo de la ruta de solicitud tiene una copia nueva de la representación solicitada, utiliza esa copia para satisfacer la solicitud. Si ninguna de las memorias caché puede satisfacer la solicitud, la solicitud viaja hasta el servicio (o el servidor de origen como se conoce formalmente).

Spring Web MVC

HTTP Catching

Beneficio

El almacenamiento en caché de HTTP puede mejorar significativamente el rendimiento de una aplicación web.

El almacenamiento en caché HTTP gira en torno al encabezado de respuesta de Control de caché y, posteriormente, los encabezados de solicitud condicional (como Last-Modified y ETag). Cache-Control aconseja cachés privados (por ejemplo, navegador) y públicos (por ejemplo, proxy) sobre cómo almacenar en caché y reutilizar las respuestas.

Un encabezado ETag se usa para hacer una solicitud condicional que puede resultar en un 304 (NO MODIFICADO) sin cuerpo, si el contenido no ha cambiado. ETag puede verse como un sucesor más sofisticado del encabezado Last-Modified.

Spring Web MVC

HTTP/2

Concepto

Los contenedores Servlet 4 son necesarios para admitir HTTP / 2, y Spring Framework 5 es compatible con Servlet API 4. Desde la perspectiva del modelo de programación, no hay nada específico que las aplicaciones tengan que hacer. Sin embargo, hay consideraciones relacionadas con la configuración del servidor. Para más detalles, vea la página wiki HTTP / 2.

La API de Servlet expone una construcción relacionada con HTTP / 2. Puede usar javax.servlet.http.PushBuilder para enviar recursos de manera proactiva a los clientes, y se admite como argumento de método para los métodos @RequestMapping.

Spring Web MVC

WebSocket

Concepto

El protocolo WebSocket RFC 6455 define una nueva capacidad importante para las aplicaciones web: comunicación bidireccional full-duplex entre el cliente y el servidor. Es una nueva y emocionante capacidad que sigue los pasos de una larga historia de técnicas para hacer que la web sea más interactiva, incluidos Applets Java, XMLHttpRequest, Adobe Flash, ActiveXObject, varias técnicas Comet, eventos enviados por el servidor y otros.

Una introducción adecuada al protocolo WebSocket está más allá del alcance de este documento. Sin embargo, como mínimo, es importante comprender que HTTP se usa solo para el protocolo de enlace inicial, que se basa en un mecanismo integrado en HTTP para solicitar una actualización de protocolo (o en este caso un cambio de protocolo) al que el servidor puede responder con el estado HTTP 101 (cambio de protocolos) si está de acuerdo.

Spring Web MVC

WebSocket

Concepto

Suponiendo que el protocolo de enlace tenga éxito, el socket TCP subyacente a la solicitud de actualización HTTP permanece abierto y tanto el cliente como el servidor pueden usarlo para enviarse mensajes entre sí.

Además de los desafíos de adopción a corto y mediano plazo, el uso de WebSocket presenta importantes consideraciones de diseño que es importante reconocer desde el principio, especialmente en contraste con lo que sabemos sobre la creación de aplicaciones web en la actualidad.

Hoy, REST es una arquitectura ampliamente aceptada, comprendida y compatible para crear aplicaciones web. Es una arquitectura que se basa en tener muchas URL (sustantivos), un puñado de métodos HTTP (verbos) y otros principios como el uso de hipermedia (enlaces), permanecer sin estado, etc.

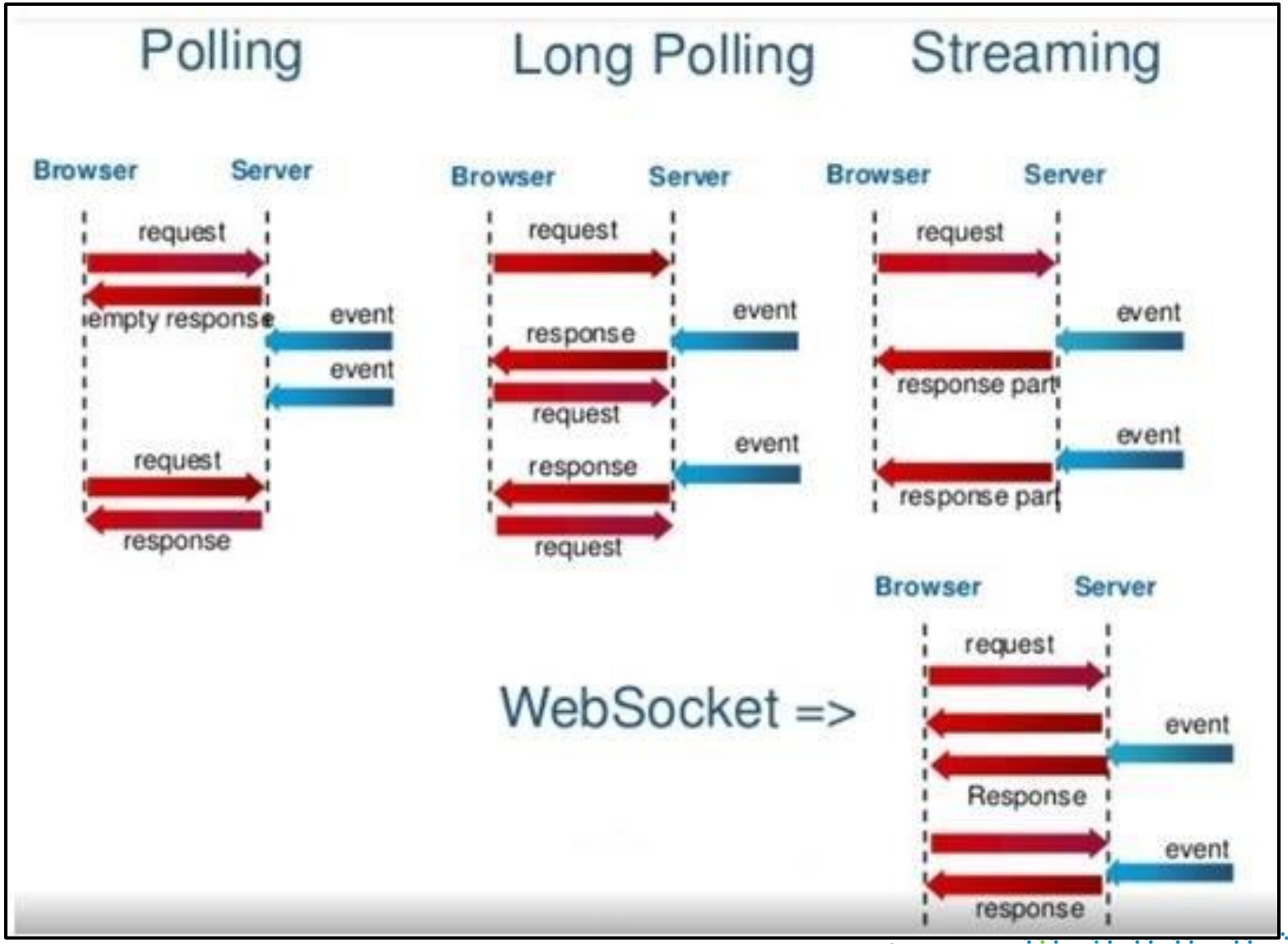
Spring Web MVC

WebSocket

Por el contrario, una aplicación WebSocket puede usar una única URL solo para el protocolo de enlace HTTP inicial. Todos los mensajes posteriores comparten y fluyen en la misma conexión TCP. Esto apunta a una arquitectura de mensajería completamente diferente, asíncrona, dirigida por eventos. Una que está mucho más cerca de las aplicaciones de mensajería tradicionales (por ejemplo, JMS, AMQP).

Spring Framework 4 incluye un nuevo módulo de mensajes de Spring con abstracciones clave del proyecto Spring Integration, como Message, MessageChannel, MessageHandler y otros que pueden servir como base para dicha arquitectura de mensajes. El módulo también incluye un conjunto de anotaciones para asignar mensajes a métodos, similar al modelo de programación basado en anotaciones Spring MVC.

Tecnología de comunicación sobre HTTP



Tecnología de comunicación sobre HTTP

Antecedentes

Originariamente, las aplicaciones web se construían sobre un modelo cliente-servidor, en el que quien iniciaba todas las transacciones era siempre el primero. Si bien es sencillo de implementar, este mecanismo corre con una debilidad: no le permite al servidor enviar información por voluntad propia.

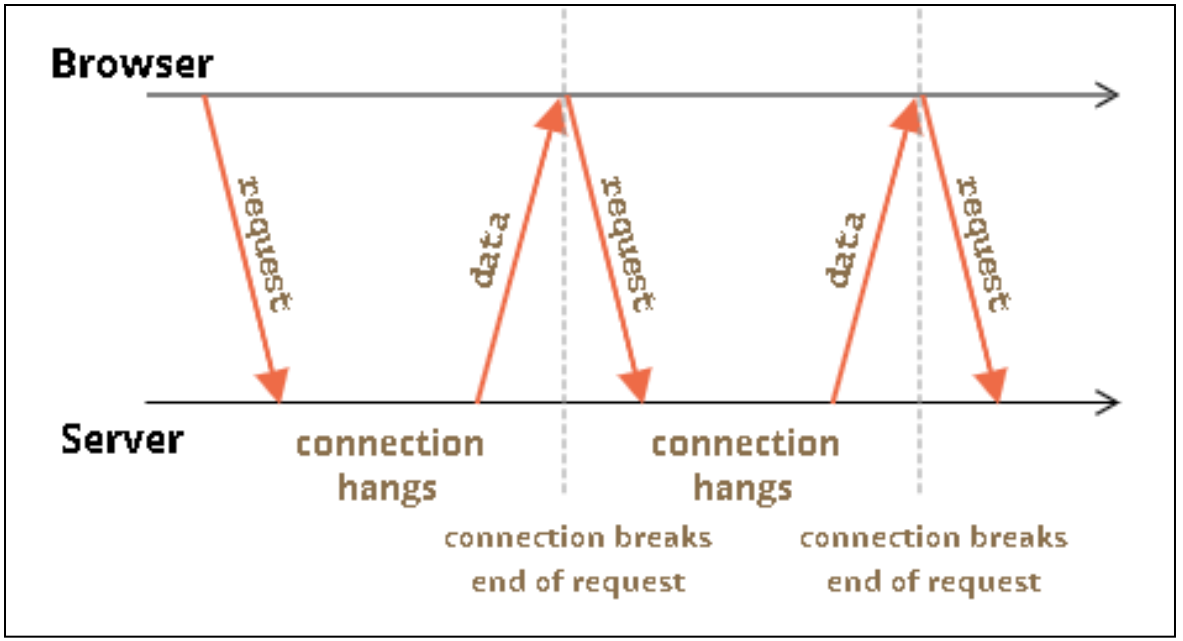
Ahora pensemos en aplicaciones con información en tiempo real (se podra verificar que el antiguo método no se adaptaría a esta lógica) es por ello que aparece las comunicaciones del tipo Long Polling o WebSockets, las cuales son asíncronas.

Tecnología de comunicación sobre HTTP

Long Polling

Concepto

Es la forma más simple de tener una conexión persistente con el servidor, que no utiliza ningún protocolo específico como WebSocket o Server Side Events.



Tecnología de comunicación sobre HTTP

Long Polling

Funcionamiento

1. Se envía una solicitud al servidor.
2. El servidor no cierra la conexión hasta que tiene un mensaje para enviar.
3. Cuando aparece un mensaje, el servidor responde a la solicitud con él.
4. El navegador realiza una nueva solicitud de inmediato.

Es decir La técnica de **Long Polling** nos permite emular una funcionalidad de tipo “server push” sobre HTTP en la que el servidor, a diferencia de cómo era en el antiguo método,“inicia” los eventos.

Tecnología de comunicación sobre HTTP

Streaming

Concepto

La tecnología HTTP Streaming es similar al long-polling excepto que la conexión no se cierra con la existencia de nuevos datos disponibles o en un intervalo determinado. En cambio, estos datos se envían a la conexión existente que permanece abierta, esto significa que el servidor podrá seguir enviando fragmentos de datos al cliente.

Es importante que la conexión permanezca abierta, ya que, el protocolo HTTP solo permite realizar peticiones desde el lado del cliente. Por lo tanto, cuando se produce un cambio de estado en la información (dato), no hay forma de que el servidor pueda abrir las conexiones necesarias para los clientes interesados.

Tecnología de comunicación sobre HTTP

Streaming

Funcionamiento

Existen dos categorías: Streaming Bajo Demanda (On-Demand) y Video Streaming (Live Streaming).

Streaming Bajo Demanda (On-Demand)

Este se tipo de Streaming se caracteriza porque el usuario descarga el archivo de video completo y luego reproduce. Sin embargo, la transferencia completa de archivos en el modo de descarga generalmente sufre un tiempo de transferencia largo y quizás inaceptable.

Tecnología de comunicación sobre HTTP

Streaming

Funcionamiento

Video Streaming (Live Streaming).

En Live Streaming no es necesario la descarga del contenido de video en su totalidad. Por el contrario, este se va reproduciendo mientras se recibe y decodifican las partes del contenido. Una analogía que se emplea para comprender esta categoría de Streaming es el servicio por televisión, donde en ella se puede consumir el recurso mientras se transmite y al finalizar el programa televisivo este ya no se encontrará disponible para el usuario.

Aplicabilidad:

Transmisión de Video

Tecnología de comunicación sobre HTTP

WebSocket

Concepto

Es un protocolo que provee canales de comunicación *full-duplex* sobre una única conexión TCP.

Un canal de comunicación es *full-duplex* cuando los datos van en ambos sentidos de manera simultánea.

En una comunicación cliente-servidor no se da de esta manera, ya que el cliente hace un pedido y espera la respuesta del servidor, con websocket se envían datos entre cliente y servidor sin esperar respuesta alguna

Tecnología de comunicación sobre HTTP

WebSocket

Funcionamiento

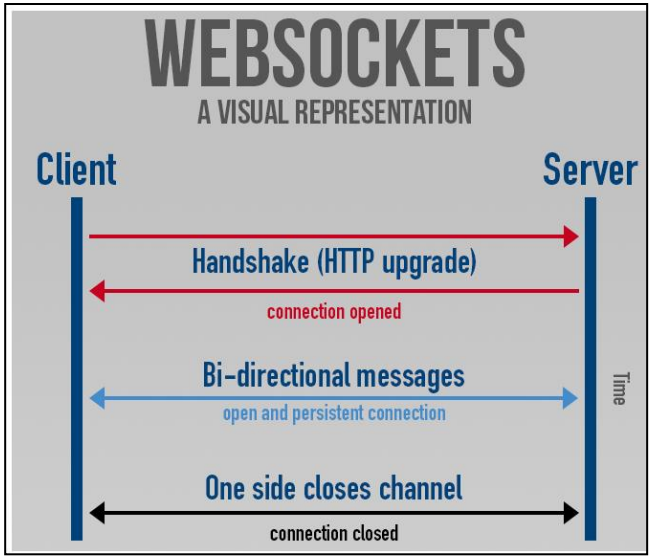
El cliente establece una conexión TCP con el servidor mediante un “handshake”. Tanto el servidor como cliente pueden enviarse mensajes entre sí en el momento en que deseen, así, la conexión permanece viva hasta que cualquiera de los dos extremos la cierre explícitamente.

El “handshake” (proceso de negociación de los parámetros del canal de comunicación entre un cliente y un servidor) al que se hacía referencia arriba se da de este modo

Tecnología de comunicación sobre HTTP

WebSocket

Diagrama



Aplicabilidad

Son muchas las aplicaciones que usan WebSockets: juegos, redes sociales o chats. Podemos decir, entonces, que se trata de una tecnología cuyo uso seguramente se incrementará en los próximos años.

Lecturas adicionales

Para obtener información adicional, puede consultar los siguientes enlaces:

https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html

http://www.http2demo.io/

Resumen

En este capítulo, usted aprendió:

- Spring Framework, concepto, módulos y proyectos
- Spring Core: Conceptos fundamentales.
- Spring Web MVC y la relación con RESTful
- Tecnología de comunicación sobre HTTP