Acceso a datos

Introducción a la API de **log4j2**

¿Qué es la API de log4j2?

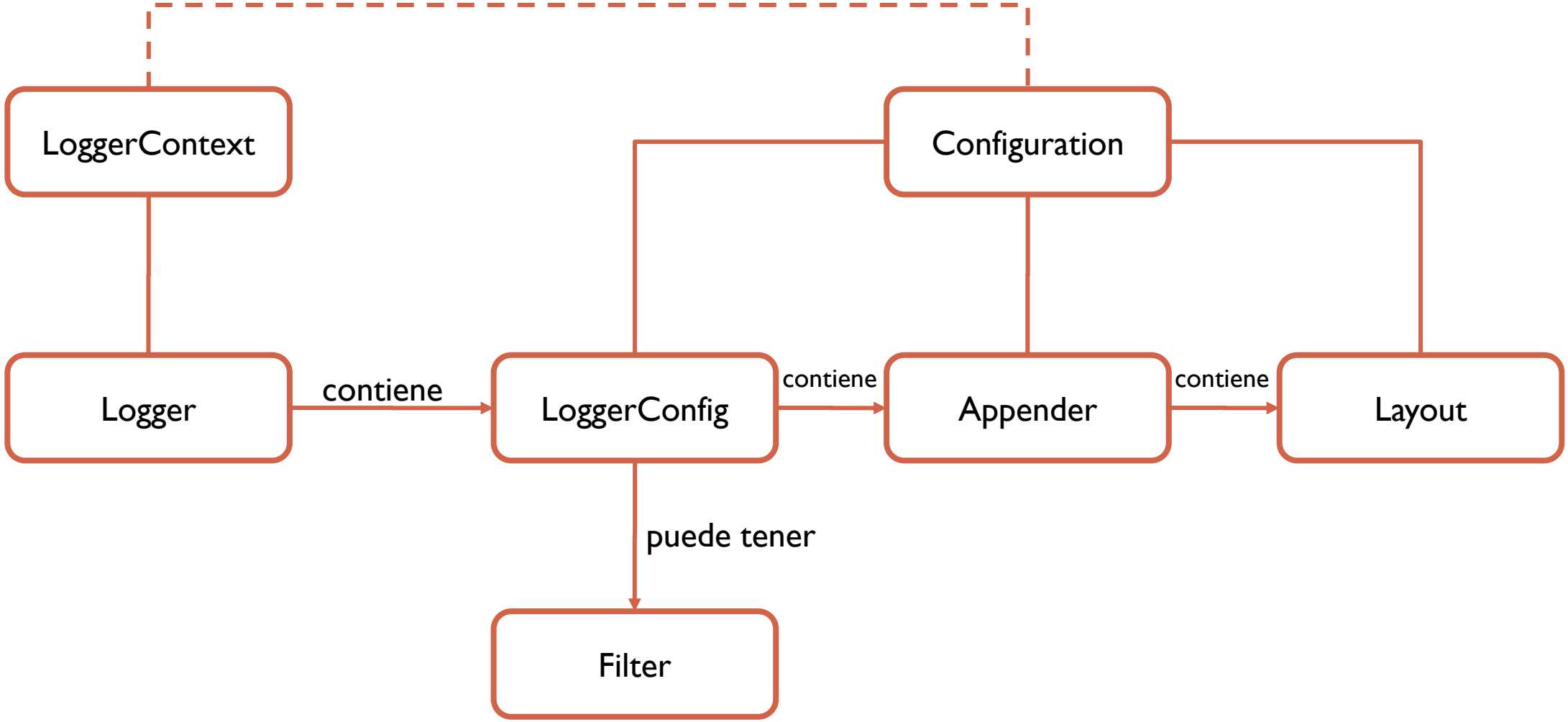
- Indicar qué hace un programa en un entorno empresarial mediante ***System.out*** no es la mejor idea.
  - Además, sobrecarga la consola de salida y se mezclan todos los mensajes de aviso de los diferentes programas en ejecución.
- **log4j2** es una API muy utilizada que nos permite:
  - Usar diferentes niveles de mensajería preestablecidos.
  - Centralizar los mensajes de aviso (y su configuración) en un único lugar.
- La estructura que ofrece es muy versátil, y se puede configurar para una aplicación completa, por paquetes o por clases.

Niveles pre-configurados y propios

Nivel	Valor
OFF	0
FATAL	100
ERROR	200
WARN	300
INFO	400
DEBUG	500
TRACE	600
ALL	MAX_VALUE

- Los niveles van de más especiales a menos especiales y más comunes de utilizar.
- Van de 100 en 100 para poder incluir nosotros nuestros propios niveles entre medias.
- Cada uno de esos niveles se puede configurar para que haga una cosa concreta.

Arquitectura



¿Qué es cada cosa?

- LoggerContext
  - Es la unidad central de todo el sistema de log.
  - Mantiene todos los Logger existentes y que se pueden solicitar en la aplicación.
  - Mantiene una referencia a la configuración general.
- Configuration
  - Contiene y describe la configuración de todo el sistema de log.
  - Describe todos los Logger, Appender, Filter, etcétera, que pueden usarse en toda la aplicación.
  - Se puede hacer en un archivo de Properties, XML, JSON o YAML.
  - También se puede configurar mediante programación, aunque no es nada usual.
  - Se puede recargar su configuración sin tener que reiniciar toda la aplicación.

- Logger
  - Es el componente principal para hacer un registro de un evento.
  - Los Logger se obtienen mediante

    `LogManager.getLogger()`
  - Los mensajes del log se llaman posteriormente sobre un Logger concreto, para indicar el nivel de seriedad de dicho evento.
- LoggerConfig
  - Mediante LoggerConfig describimos cómo se va a comportar un Logger concreto.
  - Permite también indicar diferentes acciones a realizar para cada uno de los niveles, así como configurar los Appender y Filter

- Appender
  - Describe y configura cómo y dónde se va almacenar un mensaje de log.
  - Un Logger puede tener múltiples Appender, y se mandará el mensaje de log a todos y cada uno de ellos.
  - Log4j2 tiene varios Appender preconfigurados y dispuestos para su uso: ConsoleAppender, FileAppender, HTTPAppender, JDBCAppender, NoSQLAppender, y un largo etcétera.
- Filter
  - Se pueden agregar filtros para seleccionar qué mensajes son realmente importantes e interesantes, de forma centralizada. Pueden aplicarse a Logger o a Appender.
- Layout
  - Indica cómo se va a mostrar cada mensaje, y se asocia a un Appender concreto.

Log4j2 vs Log4j

- Plugins
  - Permite la extensibilidad mediante plugins, ahorrando mucha codificación.
  - Para ello, se usará la anotación @Plugin en nuestra clase.
- Recolector de basura
  - Se mejora la eficiencia del Garbage Collector, usando muchísima menos memoria que en la versión previa.
  - Se mejora el trabajo con *Threads* gracias al uso de Context Map Lookup.
- Logger Asíncronos (Asynchronous Loggers)
  - Descargan bastante trabajo gracias a los Asynchronous Loger, que se ejecutan en hilos separados de la aplicación principal.
- Soporte de expresiones lambda.
  - Gracias a ellas ahorramos muchísimas líneas de código, deja un código más limpio.

Agregar Log4j2 al proyecto

Usando Apache Maven

Configuración de un proyecto Java

Tipos de proyectos

- Un proyecto Java puede tener múltiples estructuras, siendo las más conocidas:
  - Ant
  - Maven
  - Gradle
- Cada estructura nos define una serie de carpetas, así como una forma de trabajo interna.

Maven

- En **Maven** utilizaremos **Actifacts**, que es un lugar centralizado donde se almacenan todas las versiones de las librerías que soporta.
  - La mayoría de Actifacts son POM y JAR, pero en realidad puede ser cualquier cosa.
- Utilizaremos el archivo **pom.xml** para agregar nuestras dependencias.

te vas a api y buscas log4j api

Agregar las dependencias (de la y API y del CORE)

- Se agrega al archivo pom.xml las dependencias del **core** y de la **api**

```
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.20.0</version>
</dependency>
```

O también…

- Si queremos mantener la misma versión (y hacer el mantenimiento futuro más sencillo) a través de varios Artifacts que están relacionados entre sí, tenemos que utilizar el BOM (Bill of Material), que se encarga de sincronizar las versiones.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-bom</artifactId>
      <version>2.20.0</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- Una vez definido el BOM, las versiones no son necesarias:

```
<dependencies>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
  </dependency>
</dependencies>
```

En otro tipo de proyectos

- Si trabajas en un proyecto Gradle
  - Hay que tocar el archivo de dependencias, build.gradle, y agregar:

```
dependencies {
    implementation 'org.apache.logging.log4j:log4j-api:2.20.0'
    implementation 'org.apache.logging.log4j:log4j-core:2.20.0'
}
```

- Si trabajamos en un proyecto Ant
  - Bajamos sencillamente los archivos .jar y los agregamos en nuestras dependencias.

Archivo de configuración

- Se puede pasar por parámetro del programa, como argumento, dando el valor a la propiedad log4j.configurationFile.
- Si no lo encuentra, se pone a buscar en el siguiente orden:
  - log4j2.properties
  - log4j2.yml o log4j2.yaml
  - log4j2.jsn o log4j2.json
  - log4j2.xml
- Si no encuentra ninguno de los archivos, entra a funcionar la DefaultConfiguration:
  - Nivel Root, con nivel por defecto de ERROR
  - Los mensajes se sacan solo por consola
  - Se utiliza un PatternLayout por defecto muy concreto

Layout

- Mediante el *Layout* elegimos qué información se va a mostrar.
- Se puede configurar para mostrar una cierta información.
- Hay diferentes tipos de *Layout*, pero los más comunes son:
  - Simple
  - Pattern
  - XML
  - JSON
  - CSV
  - HTML