DGTIC UNAM
DIRECCIÓN GENERAL DE CÓMPUTO Y DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN

REDEC UNAM

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
DIRECCIÓN GENERAL DE CÓMPUTO Y DE TECNOLOGÍAS
DE INFORMACIÓN Y COMUNICACIÓN

DIPLOMADO

Desarrollo de sistemas con tecnología Java

Módulo 11

Logging

Mtro. Alfonso Gregorio Rivero Duarte

1. Logging

1.1 Logging en las aplicaciones

¿Qué es Logging?

Hay que aprender a comprender el inicio de sesión en Spring Boot y cómo utilizar funciones de registro comunes mientras se desarrollan aplicaciones Spring Boot. Spring Boot simplifica enormemente las configuraciones de registro, lo que permite a los programadores centrarse en codificar la lógica empresarial.

Cuando se está preparando un programa para un entorno de producción tener un log donde se reporten los eventos y errores puede ser la diferencia entre pasar una semana tratando de replicar un error o solo leer un archivo y saber en que linea ocurrió el error y si bien hay todo un mundo de librerías y frameworks para este propósito no hay que olvidar que el propio Java ya contiene las clases para hacer esto y que nunca esta de mas ahorrarse dependencias.

¿Qué es Logging?

El LOG es un mensaje que le indica al desarrollador la ocurrencia de un evento que desea monitorear dentro de la aplicación.

Típicamente usamos salidas estándar en consola.

- `System.out.println()`
- `Console.WriteLine()`

Nuestras aplicaciones quedan con mensajes de todo tipo dentro del código, mensajes que en la salida a producción de los productos en algunos casos deben ser borrados.

El código desplegado en el servidor no habla?
En el servidor de producción no existe forma de depurar paso a paso?

¿Por qué no seguir usando salidas estándar a consola?

Consume recursos en el sistema operativo ya que las llamadas a **`System.out.println`** se agregan a un archivo Log que controla el servidor de aplicaciones.

Es difícil encontrar un error dentro de toda la aplicación.

En tiempo de ejecución al utilizar **`System.out.println()`** no es posible configurar el nivel de granularidad de los mensajes.

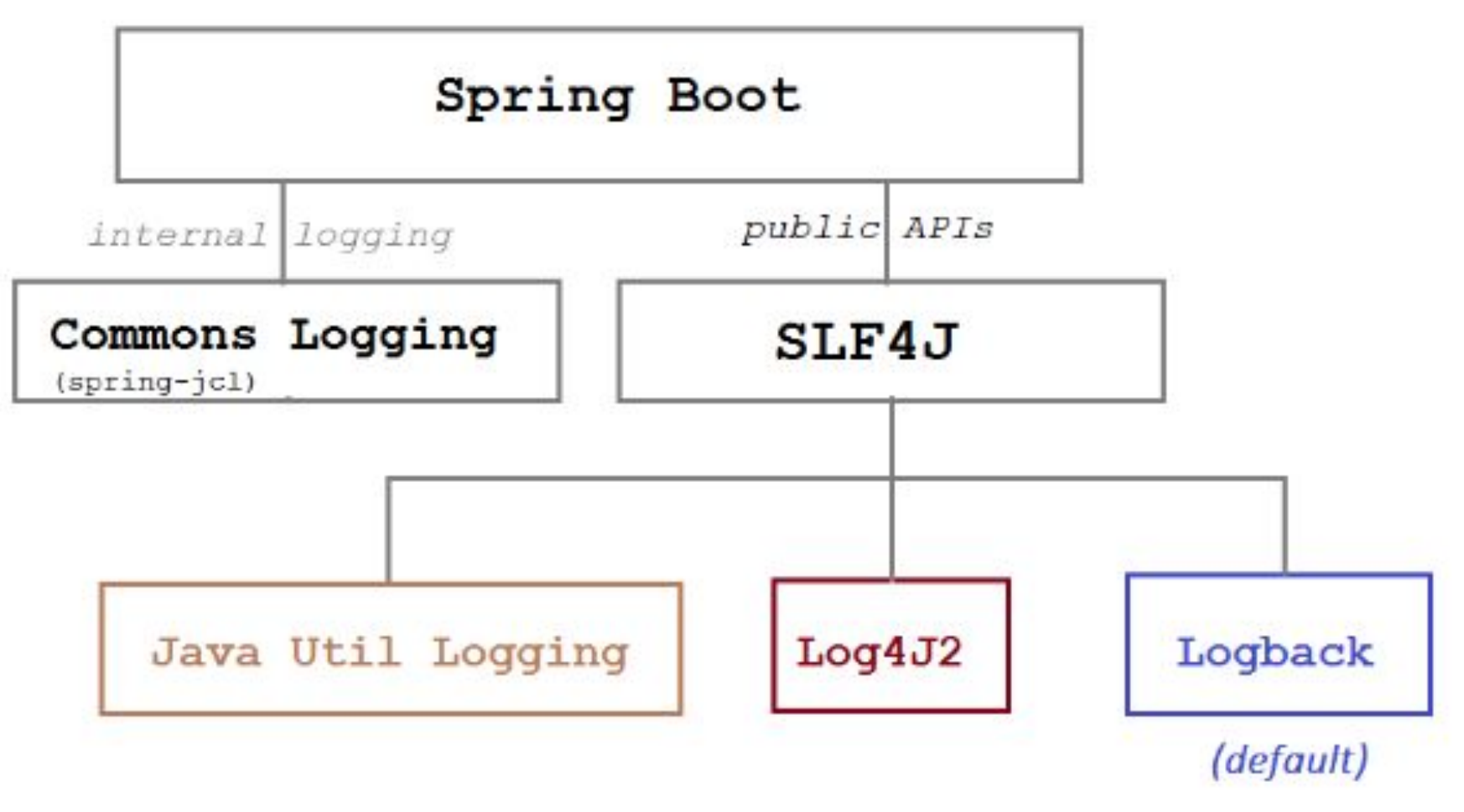
¿Por qué no seguir usando salidas estándar a consola?

No se pueden deshabilitar los mensajes, ya que están en código duro.

El manejo y control de mensajes en tiempo de ejecución es nulo.

No es posible enviar los mensajes a otros sistemas diferentes de la consola ejemplo JMS, JDBC, SMTP, entre otros.

Comprender la arquitectura de logging de Spring Boot



Comprender la arquitectura de logging de Spring Boot

Para las API internas, Spring Boot utiliza Java Commons Logging (JCL). Para las API públicas que están expuestas a los programadores de aplicaciones, Spring Boot utiliza SLF4J (Simple Logging Façade para Java), que permite a los programadores elegir diferentes marcos de registro.

Actualmente, Spring Boot viene con bibliotecas para Java Util Logging (JUL), Log4J2 y Logback. Y Logback se utiliza como marco de registro predeterminado. Eso significa que cuando se utiliza Spring Boot, no es necesario especificar ninguna dependencia del marco de registro, ya que están incluidas de forma predeterminada.

Incluso, al usar Lombok, ésta librería trae anotaciones para ejecución de logs de manera más sencilla

Escribir mensajes de log en Spring Boot

Dado que Spring Boot usa SLF4J para las API públicas, podemos declarar un log usando la API SLF4J. Por ejemplo:

```
private static final Logger LOGGER = LoggerFactory.getLogger(AppController.class);
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Escribir mensajes de log en Spring Boot

Y luego puede escribir mensajes de registro para diferentes niveles de registro de la siguiente manera:

```
LOGGER.trace("for tracing purpose");
LOGGER.debug("for debugging purpose");
LOGGER.info("for informational purpose");
LOGGER.warn("for warning purpose");
LOGGER.error("for logging errors");
```

Escribir mensajes de log en Spring Boot

Si usamos Lombok, sería mucho más fácil.

Ya no sería necesario declarar:

```
private static final Logger LOGGER = LoggerFactory.getLogger(AppController.class);
```

Sería tan sencillo como anotar **`@Slf4j`** a nivel de clase, y listo!

Eso es todo. Al utilizar las API SLF4J, su código de registro no depende de ningún marco de registro. Y puede optar por utilizar un marco de registro específico proporcionando su archivo de configuración en el classpath.

Nivel de Prioridad de Traza

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

■ Nivel básico

- ALL
- OFF

■ Nivel extra

DEBUG

- Se utiliza para escribir mensajes de depuración, este log no debe estar activado cuando la aplicación se encuentre en producción.

INFO

- Se utiliza para mensajes similares al modo “verbose” en otras aplicaciones.

WARN

- Se utiliza para mensajes de alerta sobre eventos que se desea mantener constancia, pero que no afectan el correcto funcionamiento del programa.

ERROR

- Se utiliza en mensajes de error de la aplicación que se desea guardar, estos eventos afectan al programa pero lo dejan seguir funcionando, como por ejemplo, algún parámetro de configuración no es correcto y se carga el parámetro por defecto.

FATAL

- Se utiliza para mensajes críticos del sistema, generalmente luego de guardar el mensaje el programa abortará.

ALL

- Este nivel es el más bajo posible, habilita todos los logs de la aplicación

OFF

- Este es el nivel más alto posible, se deshabilitan todos los logs

Configuración en Spring

- Dependiendo del nivel que se requiere, se puede configurar en el archivo properties:

```
# To set logs level as per your need.
logging.level.root = DEBUG
```

Contacto

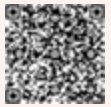
Mtro. Alfonso Gregorio Rivero Duarte

Senior Data Manager - CBRE

devil861109@gmail.com

Tels: (+52) 55 289970 69

Redes sociales:



https://www.linkedin.com/in/alfonso-gregorio-rivero-duarte-139a9225/