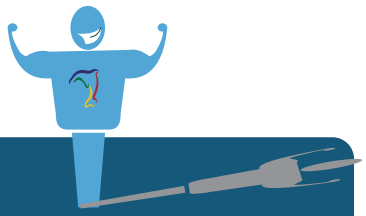
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

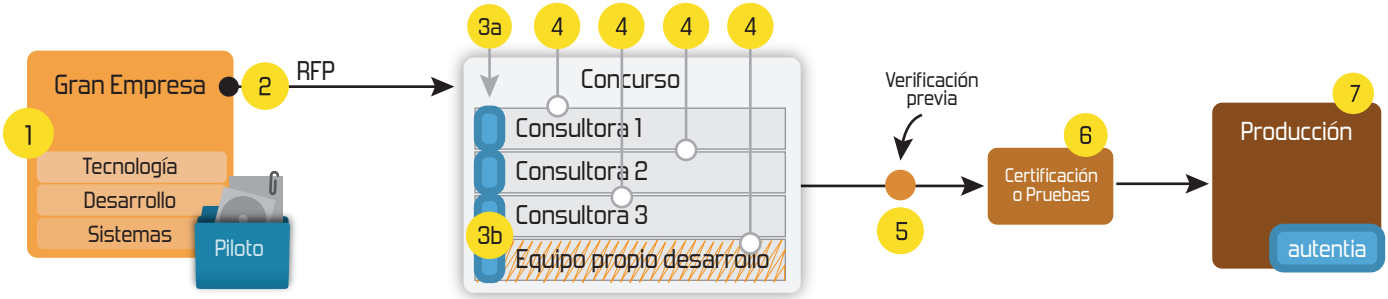
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

AdictosAlTrabajo
Final de Terrakas ¡¡No te lo pierdas!!
terrakas.com
autentia Soporte a desarrollo informático
Hosting patrocinado por enredados
Entra en Adictos a través de
Entrar
Deseo registrarme
Olvidé mi contraseña
Inicio Quiénes somos Formación Comparador de salarios Nuestros libros Más

Jose Manuel Sánchez Suárez

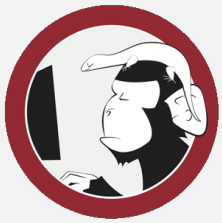
Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Catálogo de servicios Autentia



Síguenos a través de:

Fecha de publicación del tutorial: 2013-09-10 Tutorial visitado 0 veces Descargar en PDF

Introducción a SLF4J: Simple Logging Facade for Java.

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. Beneficios de usar SLF4J frente a una implementación de logging concreta.
- 4. Creación de una implementación propia.
- 5. Referencias.
- 6. Conclusiones.

1. Introducción

1.1. Aviso para navegantes.

Lo normal a veces es la excepción, pero trabajando en proyectos empresariales "normalmente" el equipo de desarrollo no tiene acceso a los entornos de estabilidad de un sistema horizontal, véase, preproducción, QA o, muchos menos, producción.

¿Qué ocurre si despúes de pedir un pase entre entornos se produce un error o excepción en un entorno al que el equipo de desarrollo no tiene acceso?, pues que debemos de disponer de los mecanismos necesarios para recabar toda la información posible del error en el entorno en el que se reproduce, aquello de "acceder o pedir los logs". Debemos tener en cuenta que cuando hablamos de un pase entre entornos, puede traducirse en horas o días, ya no solo depende de tu prioridad sino de la prioridad del equipo de sistemas que te da soporte. Esperar 3 días para un pase y terminar con "la primera en la frente", hace que debamos ser precabidos y, siguiendo las recomendaciones del framework corporativo, hagamos uso del sistema de logging que nos permita tener acceso a la información para detectar el origen del error.

A nivel de arquitectura, podemos elegir, e imponer el uso a nivel de desarrollo, de un sistema de logging que le permita:

- al equipo de desarrollo la trazabilidad de las operaciones que se realizan en el código, con distintos niveles de trazas, y.
- al equipo de sistemas:
  - proporcionar una vía para que el equipo de desarrollo tenga acceso a las trazas (teniendo en cuenta si son o no aplicaciones distribuidas) o remitirlas fácilmente frente a una petición, y
  - poder modificar el nivel de trazabilidad en caliente, para que se tracen ciertas operaciones en una situación excepcional, aquello de bajar el nivel de log para detectar un error.

En este tutorial, en el entorno de aplicaciones Java/JEE, vamos a exponer el uso de SLF4J sobre cualquier otro sistema de logging e intentaremos justificar su elección.

1.2. Simple Logging Facade For Java.

Simple Logging Facade for Java (SLF4J) es una abstracción de varios frameworks de logging que permite elegir el framework concreto en tiempo de despliegue. Como su propio nombre indica, no es más que una fachada.

"Por debajo", podemos elegir usar entre:

- log4j
- java.util.logging
- Jakarta Commons Logging
- logback
- una implementación por defecto que traza vía syserr u otra que silencia los logs,

Últimas Noticias

» Técnicas de división de historias de usuario
» Dolomitas on Giro
» Comentando el libro: Agile Management de Angel Medinilla
» Final de temporada de Terrakas Autentia freaklances
» Atención, APLAZADO Estreno último capítulo de Terrakas
Histórico de noticias

Últimos Tutoriales

» Comentando Management 3.0, de Jurgen Appelo
» Un vistazo a Django Web Framework
» Introducción a XQuery con ejemplos
» Contando mi experiencia en los talleres gratuitos de Apple
» Comentando el libro: Disciplined Agile Delivery

- o crear nuestra propia implementación para SLF4J, haciendo uso de una implementación de otro framework de logging ya existente o no.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.4 GHz Intel Core i7, 8GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Lion 10.7.4
- slf4j 1.7.5
- Java(TM) SE Runtime Environment (build 1.7.0_11-b21)

3. Beneficios de usar SLF4J frente a una implementación concreta.

Si SLF4J no es más que una fachada y, al final, terminamos usando java.util.logging, ¿por qué no usar directamente java.util.logging?, en este punto intentaremos justificarlo:

- bajo acoplamiento, si usamos SLF4J, no acomplamos nuestras clases a una implementación de logging concreta; aún haciendo uso de la implementación estándar de la propia JDK, usando SLF4J podríamos seguir haciendo uso de la misma y, si decidimos en un futuro intercambiarla por otra las aplicaciones no sufrirían ninguna modificación. Pero con SLF4J podemos ir un paso más alla y, puesto que la implementación concreta de logging se resuelve en tiempo de despliegue podríamos tener no ya solo una configuración distinta sino una implementación distinta por entorno; bastaría con proporcionar una implementación de SLF4J a nivel de classloader del propio servidor de aplicaciones,
  - en un entorno de tests, por ejemplo con el soporte de junit, no necesitaríamos incluir ninguna implementación, simplemente no tendríamos trazas con el sistema de logging, no lo necesitamos puesto que estamos en el entorno de desarrollo; a las malas, si necesitamos trazar ya tenemos el debugger del IDE con el que trabajemos o podemos incluir en el classloader del test la implementación por defecto de slf4j-simple.
- tiene ciertas características a la hora de escribir las líneas de log que lo hacen más atractivo que usar una implementación concreta, básicamente:
  - no es necesario comprobar el nivel de trazabilidad antes de escribir una línea de log y
  - evita la concatenación de cadenas ya que se pasan como argumentos variables al método que traza.
- las dos características anteriores, hacen que no solo sea más atractivo, sino que es más eficiente, puesto que:
  - si al desarrollador se le olvida comprobar el nivel de trazabilidad antes de trazar con java.util.logging, lo hace internamente, con lo que no hay posibilidad de error, y
  - consume menos recursos, imaginad una traza con una concatenación de cadenas dentro de un bucle.
- las aplicaciones solo tendrán una dependencia a slf4j-api-X_X_X.jar, no a la implementación concreta.
- no se basa en ninguna configuración especial o modificación del classloader.

A continuación algunos ejemplos de uso:

```
1  import org.slf4j.Logger;
2  import org.slf4j.LoggerFactory;
3  // no importamos ningún paquete específico de la implementación de logging que usaremos
4
5  public class HelloWorld {
6    private static final Logger LOGGER = LoggerFactory.getLogger(HelloWorld.class);
7
8    private static final String userName = "psanchez";
9    public void doNothing() {
10
11     LOGGER.info("Hello World");
12
13     LOGGER.trace("Hello {}", userName);
14
15     try {
16         // any exception
17     } catch (Exception e){
18         LOGGER.error("Exception happens", e);
19     }
20
21     // no longer necessary
22     // if(LOGGER.isTraceEnalbed()){
23     //  LOGGER.trace("Hello: " + userName);
24     // }
25   }
26 }
```

4. Creación de una implementación propia.

Imaginemos que ya disponemos de un sistema de trazabilidad propio que, por las características de nuestros sistemas horizontales, nos obligan a realizar diferentes tareas previas o posteriores a generar una traza en los distintos entornos. Desde el punto de vista de arquitectura podríamos optar por crear una implementación propia de un appender de log4j o logback, o implementar nuestro propio sistema de escritura de trazas basado en una implementación de java.util.logging o apache commons logging.

En cualquier caso, podríamos optar por crearnos nuestra propia implementación de SLF4J, como capa de recubriento, para:

- no afectar a los desarrollos actuales, si ya disponen del soporte de SLF4J o
- no afectar a los desarrollos futuros, por si en algún momento decidimos migrar o intercambiarlo por otro sistema.

Llegados a este punto la operativa es sencilla y según queda documentada en la propia wiki de SLF4J, lo único que debemos hacer es tomar como referencia una implementación ya existente del sistema de logging que hayamos elegido implementar.

Con ello, bastaría con acceder al código fuente de cualquiera de las implementaciones, por ejemplo, de java.util.logging y crear un proyecto con las siguientes clases:

(DaD)

Últimos Tutoriales del Autor

» Ejecución de tests unitarios con junit en proyectos ant y su integración en jenkins y sonar para medir la cobertura.

» Soporte en Sonar para un proyecto multi-lenguaje: configuración desde Jenkins

» Instalación de Redmine (Bitnami) e integración con Subversion.

» Análisis de los sentimientos en twitter con el soporte de Apicultur.

» jBPM Form Builder: generación de formularios para jBPM5 y su integración en Guvnor.

Últimas ofertas de empleo

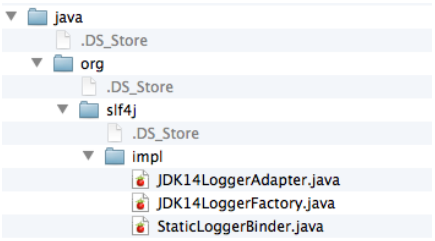
2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.



Podríamos renombrar el adaptador y la factoría. SLF4j buscará una clase con nombre StaticLoggerBinder.java en el paquete org.slf4j.impl y solo podemos tener una implementación de la misma en el classloader. La clase binder registra la factoría de loggers quien devuelve un adaptador para la clase de implementación concreta. El adaptador es realmente el encargado de traducir las operaciones de logging que se lanzan contra la fachada del logger de SLF4J a la clase de logger de la implementación concreta con la que necesitemos trabajar. Si nos basamos en una implementación ya hecha, es realmente simple, 1 punto de historia ;).

Si además no lo distribuimos a nivel de proyecto, y ya hacíamos uso de SLF4J el cambio entre el sistema anterior y el actual es transparente para los desarrolladores.

5. Referencias.

- http://www.slf4j.org/
- http://logging.apache.org/log4j/1.2/index.html
- http://docs.oracle.com/javase/6/docs/api/java/util/logging/package-summary.html
- http://commons.apache.org/proper/commons-logging/
- http://logback.qos.ch/

6. Conclusiones.

Ahora ya sabemos las diferencias entre usar SLF4J y LOG4J, o al menos deberíamos saber porque nuestro proyecto o cualquier proyecto open source tiene las dependencias de slf4j-api y además log4j y slf4j-log4j12, ¿o aún no? ;)

Un saludo.

Jose

jmsanchez@autentia.com

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

SOME RIGHTS RESERVED Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

Copyright 2003-2013 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto

XHTML 1.0 CSS RSS ATOM