Universidad de Buenos Aires

Facultad de Ingeniería
Técnicas de Diseño
(75.10)
1er Cuatrimestre 2014

Logger

Corrector: Díaz, Federico
Grupo 24, integrantes:

Apellido y nombre	Padrón	E-mail
Bogado, Sebastián Javier	91.707	sebastian.j.bogado@gmail.com
García Marra, Alejandro	91.516	alemarra@gmail.com

Tabla de Contenidos

Primera entrega

Descripción general del diseño

La idea principal entorno a este diseño es la facilidad de uso y configuración para el usuario final de alguna implementación de la interfaz Logger.

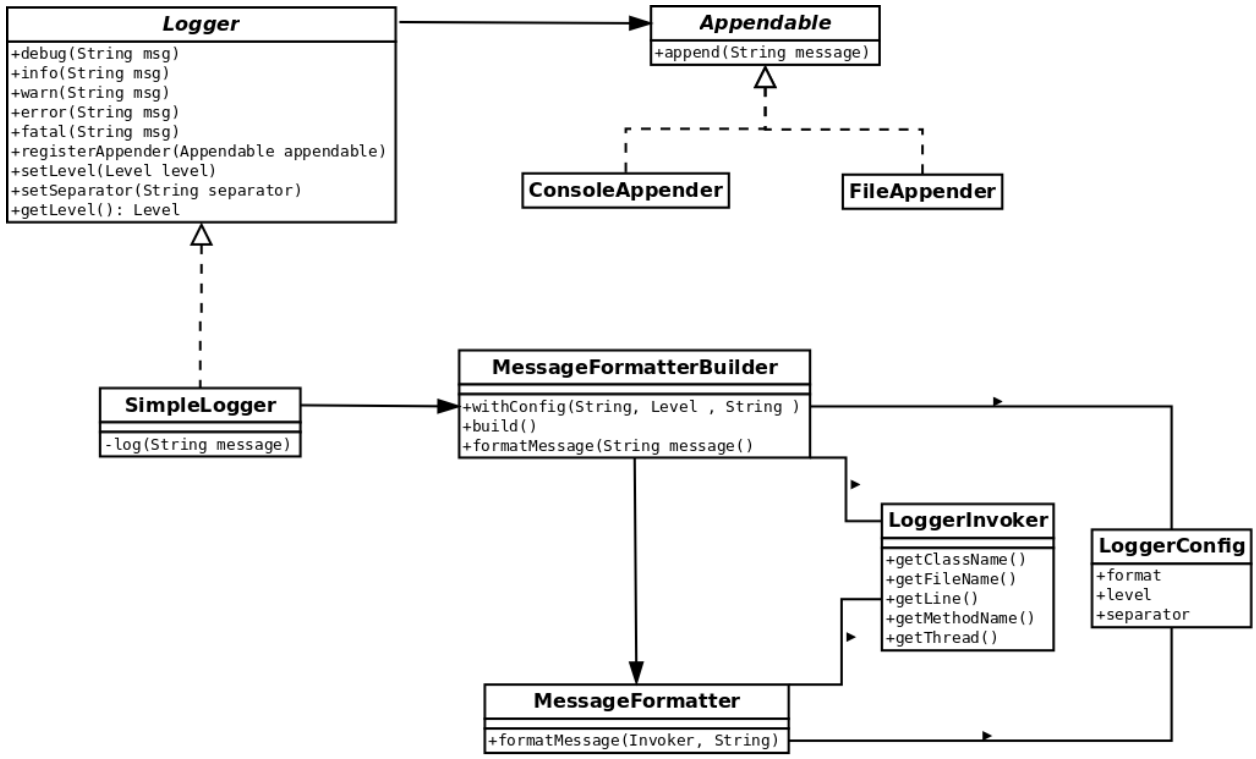
Se provee una única implementación simple de la misma, la cual utiliza el MessageFormatter básico para el formateo de mensajes. Dicho MessageFormatter encapsula la responsabilidad de aplicar las reglas de formato a cada uno de los mensajes logueados, utilizado para esto la configuración del logger (LoggerConfig) e información sobre la invocación del método de log (LoggerInvoker).

Para su creación e invocación, se provee un MessageFormatterBuilder, que reduce el acoplamiento entre el usuario y el formatter en sí. Las dependencias con LoggerConfig y LoggerInvoker no son necesarias para el usuario, pudiendo ignorar la existencia de dichas clases.

Toda implementación de Logger deberá emitir su output a través de implementaciones de Appendable. Esta interfaz provee una forma sencilla de abstraerse del mecanismo real de salida, exponiendo una api mínima y suficiente para cubrir las necesidades actuales.

ConsoleAppender y FileAppender son dos implementaciones de Appendable, y pueden registrarse de forma indistinta y en cualquier cantidad para un logger particular.

Diagrama de clases



Responsabilidades de cada clase

Logger

Esta interfaz es el Logger en sí mismo, es decir, aquéllo que usaría el usuario final (un programador). Comprende la interfaz pública de la API, que consiste en métodos para configurar, para obtener una instancia y, naturalmente, para loguear en los distintos niveles.

SimpleLogger

Esta clase es una implementación básica de la interfaz Logger. Se utiliza una lista para almacenar todas las salidas (Appenders) posibles, y un MessageFormatterBuilder para el manejo del formato de mensajes.

LoggerConfig

Clase cuya función es abstraernos de las internas de los atributos de configuración del Logger.

LoggerConfigTool

Clase destinada a la configuración del logger. Puede tomar un archivo de properties con la configuración deseada y rellenar aquellos campos faltantes con valores por defecto. También puede ser construido sólo con valores por defecto.
Es parte de la interfaz pública.

Appendable

Interfaz similar en concepto a java.lang.Appendable, pero más simple. Para abstraerse del destino de los logs.

FileAppender

Clase utilizada como interfaz entre un archivo y el logger

ConsoleAppender

Clase utilizada como interfaz entre la consola y el logger

LoggerInvoker

Clase de uso interno, utilizada como abstracción por sobre los métodos (un poco más complejos) que permiten obtener información sobre el momento en que se llamó a una función de logging. Incluye, por enunciado:

- nombre del archivo
- línea del archivo
- nombre de la clase
- nombre del método

MessageFormatter

Clase de uso interno, cuya responsabilidad es generar el mensaje final a partir de un formato definido

MessageFormatterBuilder

Clase builder que permite un nivel de abstracción extra sobre el MessageFormatter. Encapsula el uso del LoggerInvoker y el LoggerConfig

Properties

Clase que replica, limitadamente, a java.util.Properties, pero con las mejoras de:

1. Soportar UTF-8 en los archivos (java.util.Properties utiliza ISO 8859-1)
2. Soportar obtener las propiedades a partir del enum definido en LoggerConfigTool

InvalidArgumentException

Excepción lanzada cuando los argumentos son inválidos.

Segunda entrega

Modificaciones al diseño

El nuevo nivel del log, trace,no afectó en nada: sólo se agregó un valor al enum, ya que todo se manejaba en función de eso. También agregamos el correspondiente método.

Agregar la posibilidad de recibir un Throwable nos hizo escribir seis o siete métodos muy simples que terminan todos llamando a un nuevo método log que recibe también Throwable. Este concatena el mensaje de log al mensaje del Throwable y llama al antiguo método log.

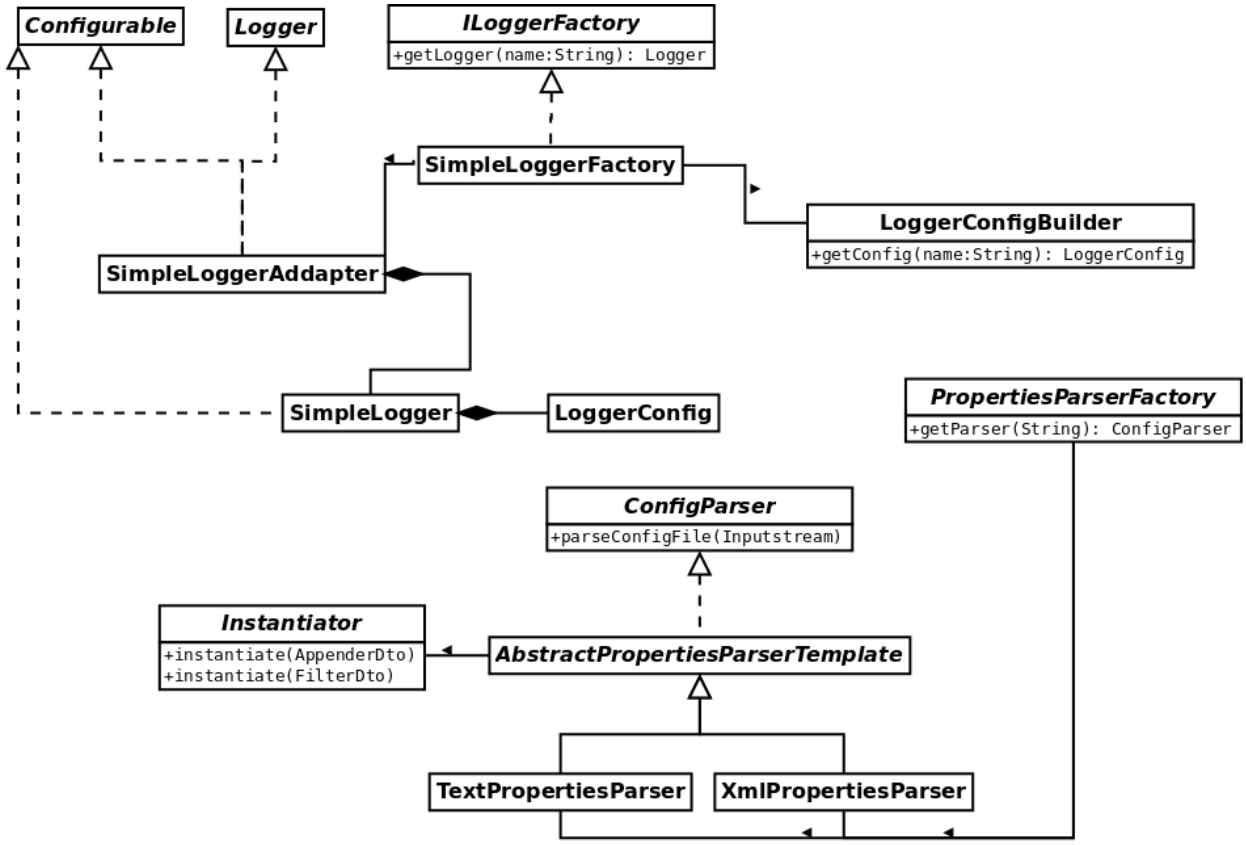
Los filters afectaron un poco más al Logger: en el método log, existía un chequeo sobre el nivel para determinar si se logueaba el mensaje o no. Esto se extrajo en un método que, además del nivel, pasa el mensaje por los filters para determinar si el mensaje debía ser loggeado o no.

El reemplazo de {} por parámetros fue construído desde afuera con unos métodos estáticos que no modifican en nada a nuestro Logger original, porque no era exigido por enunciado pero sí por el binding.

El exporter a JSON requirió atención para que este fuese coherente con el exporter de tipo String, pudiéndose controlar, entonces, los campos que componen el mensaje JSON final.

La incorporación del las nuevas configuraciones fue lo que más atención y esfuerzo requirió, ya que la posibilidad de utilizar clases desconocidas en compile-time forzó el uso de reflexion y un manejo más abstracto de las configuraciones. También el formato tipo .properties con campos dinámicos (n appenders o filters para m configs distintas) fue más complejo de manejar que el mismo archivo en xml, ya que debíamos cubrir las mismas configuraciones posibles, pero implementando todo el parser (para el XML se utilizó una librería externa).

Diagrama de clases



Si bien no se muestra en el diagrama por cuestiones de claridad, en esta estrega se modificó el MessageFormatterBuilder para retornar una implementación MessageFormatter de tipo StringFormatter o JSONFormatter.

Responsabilidades modificadas de clases existentes

LoggerConfig
Ya no se encarga del parseo de los archivos de configuración. Por lo demás, se mantuvo igual.

MessageFormatter
Pasó a ser una clase abstracta, pues la salida puede ser string o JSON

Responsabilidades de clases nuevas

Filter
Interfaz que deben respetar los filtros custom del programador, y cualquier otro filtro.

PatternFilter
Filtro que recibe una expresión regular y filtra todos los mensajes que matcheen.

JSONFormatter
Clase de uso interno, cuya responsabilidad es generar el mensaje final a partir de un formato definido, con salida en JSON

StringFormatter
Clase de uso interno, cuya responsabilidad es generar el mensaje final a partir de un formato definido

Instantiator
Instacia Appendable's o Filter's custom, definidos por el programador

PropertiesParserFactory
Factory que determina, según la extensión del archivo, qué parser se usará

TextPropertiesParser
Anteriormente llamada Properties, tiene la misma función: imitar a java.util.Properties, pero con cosas ligadas ya al Logger

XmlPropertiesParser
Clase encargada de parsear el config xml

ConfigParser
Interfaz que debe implementar todo parser del archivo de config

AbstractPropertiesParserTemplate
Clase que reúne métodos comunes entre los parsers, además de implementar ConfigParser

ParametrizedMessageInterpreter

Clase encargada de reemplazar los placeholders '{}' con parámetros

SimpleLoggerAdapter

Adapter para el slf4j

SimpleLoggerFactory

Factory de loggers, requerida por slf4j

LoggerConfigBuilder

Clase encargada de generar configuraciones para cada logger según su nombre, definidas en archivos de config.

UnsuportedFormatException

Excepción lanzada cuando el archivo de config no tiene una extensión soportada (.properties o .xml)