Scrivi messaggi di log decenti, altrimenti Gesù piange

Vorrei proporre alcune considerazioni sulla apparentemente oscura arte dello scrivere messaggi di log. Dico *apparentemente oscura* perché si tratta di un argomento apparentemente banale ma che in realtà si rivela complesso.

Trattandosi di osservazioni generali, queste andrebbero prese con una dose di buon senso e sotto stretta osservazione del proctologo di fiducia; ciò non toglie che con ogni probabilità troverai le seguenti note ragionevolmente utili poiché basate sull'esperienza personale (a.k.a. sangue versato sul campo).

Niente guerre di religione, altrimenti Feuerbach piange

Nel mondo java, log4j è considerato lo standard de facto quando si parla di framework di logging. Tuttavia, il mio consiglio è di usare un framework più moderno, come slf4j o logback. Logback è basato sull'api di slf4j ed è sufficiente dare un'occhiata agli esempi nella documentazione per essere produttivi nel giro di un quarto d'ora.

Vorrei evitare di spiegare i vantaggi di questi framework rispetto a log4j o java.util.logging perché credo l'argomento sia non solo abbondantemente trattato in letteratura ma anche facilmente foriero di guerre di religione. Le guerre di religione sono per definizione irrazionali nonché mortalmente noiose. Sei abbastanza intelligente da documentarti e decidere da solo.

Al di là di un eventuale framework e del linguaggio di programmazione usato, i messaggi di log sono essenziali sia in fase di test/debug che in fase di monitoring in produzione.

Il log è spesso sottovalutato e trattato come una seccatura che va espletata più per dovere che per una reale necessità. Tuttavia, ti posso assicurare che non c'è nulla di peggio che essere chiamati a risolvere un problema urgente in produzione e trovarsi davanti a un log incasinato.

Nessuno ti farà mai collegare col debugger in produzione. I log sono la tua unica linea di difesa.

In particolare, nel momento in cui qualche migliaio di utenti è in procinto di spendere un mucchio di soldi sul sito del tuo cliente e l'applicazione è scritta da te e per qualche oscuro motivo la possibilità di tale ingente esborso dovesse essere negata agli utenti del suddetto sito, vorrai essere sicuro di avere la possibilità di capire velocemente cosa sta succedendo dietro le quinte. In casi simili, dire che a scrivere male nel log Gesù piange è al massimo un eufemismo neanche troppo colorito.

Garbage in, garbage out

Uno dei concetti elementari del software è quello di input/output. Conoscendo l'input e lo stato di tutte le variabili coinvolte nel processo di elaborazione, sai che puoi aspettarti un certo output.

Se l'output non è coerente con le aspettative, vorrà dire che analizzando il contesto, l'input, il codice e lo stato delle variabili locali e globali coinvolte, dovresti riuscire a capire cosa non quadra.

Non è necessario tracciare nel log ogni cambiamento di stato di ogni variabile, specie se la funzione o il metodo sono sufficientemente complessi.

Esempio di log inutile:

```
public BuyResponse buyItem(BuyRequest request) {
    log.debug(" ----- buyItem ------ start -----");
    // ...
    log.debug(" ----- buyItem ------ stop -----");
    return response;
}
```

Esempio di log più utile:

```
public BuyResponse buyItem(BuyRequest request) {
    log.debug("buyItem(): request={}", request);
    // ...
    log.debug("buyItem(): request={}, response={}", request, response);
    return response;
}
```

In questo modo posso controllare l'input e l'output del metodo. Ripetendo l'input anche all'uscita del metodo, sarà più facile correlare i messaggi in fase di analisi. Tuttavia non sempre è utile questo tipo di verbosità.

Come al solito, il buon senso dovrebbe guidarti nella scelta. Ad ogni modo, input, output e cambiamenti di stato di variabili importanti andrebbero tracciati almeno a livello di debug.

Assicurati che le classi che includi nei log abbiano un override sensato del metodo `toString()`, altrimenti nei log leggerai solo un mucchio di inutili indirizzi a puntatori e non lo stato corrente delle variabili degli oggetti.

Loggare a un certo livello

Tutti i framework di logging hanno un meccanismo di livelli per cui è possibile filtrare l'output in modo da ottenere solo i messaggi che davvero ci interessano.

Spesso capita di fare copia e incolla di statement di log per cui un messaggio che dovrebbe essere a livello di errore invece finisce a livello di debug. Occhio al copia e incolla!

Formattazione coerente e senza fronzoli

Cerca di mantenere uno stile coerente e senza troppi fronzoli nei messaggi di log. Disattiva il *blocco maiuscole*, abbi pietà. Questo ti aiuterà nella fase di analisi ed eviterà che le tue applicazioni abbiano l'aspetto di una roba scritta da uno sbarbatello in visual basic.

Metti i messaggi in riga

Per quanto possibile, evita di introdurre caratteri di a capo nei messaggi di log. Quando userai `grep` per andare alla ricerca dei messaggi, è importante che ad una singola riga di output corrisponda un messaggio coerente.

All'inizio, andare a capo sembra dare un aspetto migliore all'output ma col tempo ti renderai conto che avresti fatto meglio a non farlo.

Occhio al copia e incolla

Figure 1: Allarme rosso

Come già accennato, può capitare di fare copia e incolla di statement di log.

Ogni volta che faccio copia e incolla, mi si accende una speciale lampadina nella testa che serve a ricordarmi:

> controlla ciò che hai copiato almeno due volte perché sicuramente hai dimenticato di modificare qualcosa

Questo vale più in generale e non solo per i messaggi di log. Se avessi un euro per ogni frammento di codice sbagliato per via di un copia e incolla frettoloso, in questo momento sarei sulla spiaggia di un'isola tropicale a sorseggiare cocktail attorniato da compiacenti signorine.

Eccezioni

Il log di un errore dovrebbe portarsi dietro l'istanza di eccezione catturata e, se possibile ed utile, un minimo di contesto. Spesso mi capita di vedere codice del tipo:

```
catch (ConnectionException e) {
    log.error("Error: " + e);
}
```

Sarebbe meglio una cosa del genere:

```
catch (ConnectionException e) {
    log.error("connection error: user={}", user, e);
}
```

In questo modo non perdo lo stack trace dell'eccezione catturata e posso ricondurla ad una specifica istanza di utente.

Conosci gli strumenti

Altrettanto importante è conoscere i meccanismi di deployment e configurazione del framework che stai usando. Potrebbe tornarti utile ad esempio impostare un filtro specifico su un certo logger. Impara inoltre a conoscere i meccanismi di formattazione dei messaggi.

Configurando opportunamente un formatter, i framework moderni permettono di tracciare ad esempio timestamp al millisecondo, nomi di classi, metodi e nome del thread. Il nome del thread è essenziale in un'applicazione concorrente per seguire il flusso di esecuzione.
