Log me tender

Olivier Croisier
@OlivierCroisier

Moka Technologies
mokatech.net

DEVOXX France

log(me)

Olivier Croisier

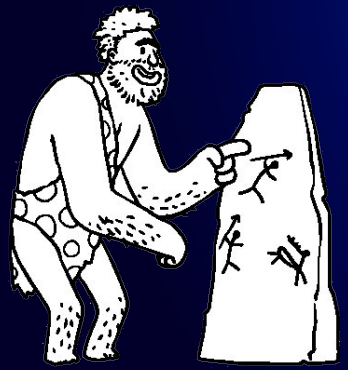
- Freelance / Moka Technologies
  Conseil, Développement & Formation
- Développeur Java / Web certifié
- Blog: `thecodersbreakfast.net`
  Twitter: `@OlivierCroisier`
  GitHub: `OlivierCroisier`

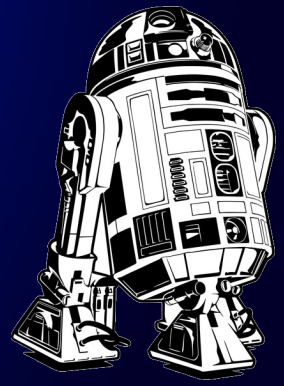
Bordeaux

Plan

- De `System.out` à nos jours
- Des problèmes persistants
- Proposition : LogService

De `System.out` à nos jours





System.out

Log primitif

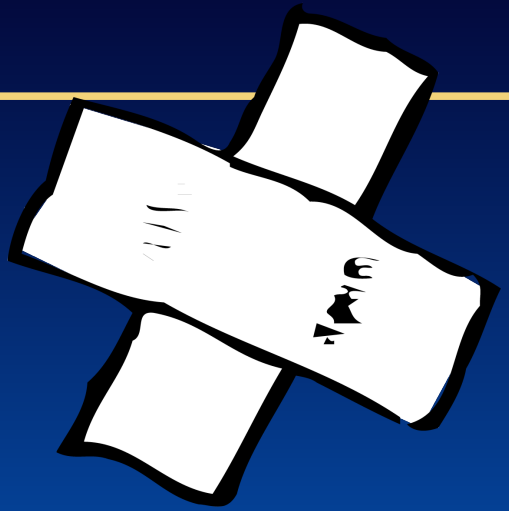
```
System.out.println("Hello World");
```

`System.out`

- `PrintWriter`
- Réassignable

Problème

API trop simple !



Log4J

Ceki Gülcü, projet Apache (2001)

Définit une API "standard"

- Impact industriel
- Inspire tous les futurs frameworks

Log4J

API

- Factory statique : `LogManager.getLogger(id)`
- `log(String msg)`
  `log(String msg, Throwable ex)`

Niveaux de log :

- `debug`, `info`, `warn`, `error`, `fatal`

Log4J

```
public class Foo {

    private static final Logger LOG = LogManager.getLogger(Foo.class);

    public void bar() {
        LOG.info("Hello world");
    }

}
```

JUL (`java.util.logging`)

JSR 47 (2002)

"Standardisation" de Log4J

API similaire à Log4J

Log4J2

Projet Apache (2012)

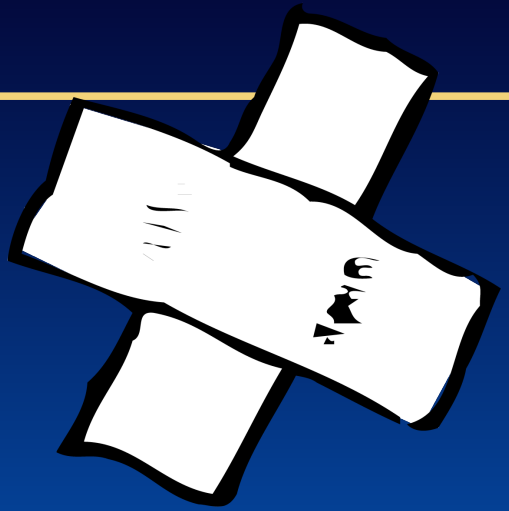
Réécriture totale

- Performances (Disruptor)
- Inspiré de Log4J et Logback

API similaire à Log4J

Problème

Trop de frameworks !



Trop de frameworks

"Dependency hell"

- Dépendances directes
- Dépendances transitives
- Classloaders

Solution

- … Un nouveau framework !

MOAR frameworks !

Commons-logging

Projet Apache (2002)

Façade pour Log4J, JUL, LogKit

API similaire à Log4J

SLF4J

Ceki Gülcü (Log4J) (2005)

- Le standard actuel

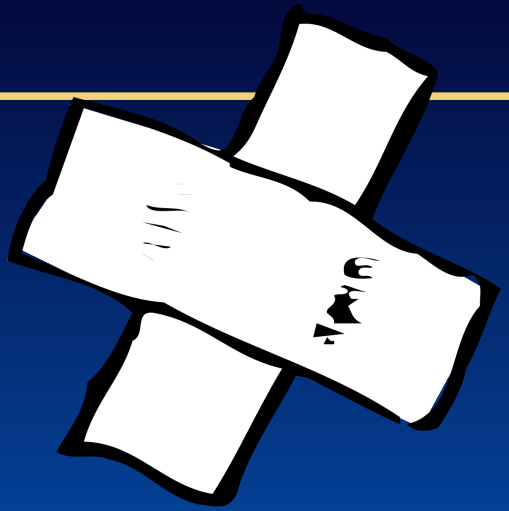
Façade pour Log4J, Log4J2, JUL, JCL, System.out...

API similaire à Log4J

Des problèmes persistants

Problème #1

Concaténations dynamiques



```
LOG.debug("User: " + user.id);
```

Concaténation dynamique

```
LOG.debug("User: " + user.id);
```

Messages souvent dynamiques

- Concaténation de chaînes

Concaténation immédiate

- Inutile selon niveau de log effectif

Concaténation dynamique

Solution : concaténations "lazy"

- Placeholders
- Vérification préalable du niveau de log

```
LOG.debug("User: {}", user.id);
```

Implémentations : SLF4J, Log4J2

Problème #2

Évaluation immédiate des paramètres

```
LOG.debug("DB: {}", DB.dump());
```

BESTEST idea ever !

Évaluation des paramètres

```
LOG.debug("DB: {}", DB.dump());
```

Evaluation immédiate

- Potentiellement coûteuse : I/O, calculs
- Parfois inutile : suivant le niveau de log effectif

Évaluation des paramètres

Solution 1 :

- Tester préalablement le niveau de log effectif
- Pattern "standard"

```
if (LOG.isDebugEnabled()) {
    LOG.debug("DB: " + DB.dump());
}
```

Évaluation des paramètres

Solution 1 : inconvénients

- Couplage technique avec la configuration du logger
- Non-DRY

```
if (LOG.isDebugEnabled()) {
    LOG.info("DB: " + DB.dump());
}
```

Évaluation des paramètres

Solution 1 : inconvénients

- Fausses bonnes idées

```
public void logDebug(String message) {
    if (LOG.isDebugEnabled()) {
        LOG.debug(message);
    }
}
```

Much SMART, wow !

Évaluation des paramètres

Solution 2 :

- Évaluation "lazy" : `Supplier<String>` (Java 8+)
- Plus besoin de test "externe"

```
LOG.debug(() -> "DB: " + DB.dump());
```

Évaluation des paramètres

Solution 2 :

- Évaluation "lazy" : `Supplier<String>` (Java 8+)

Implémentations : JUL, Log4J2… mais pas SLF4J !

- SLF4J devient le facteur limitant
- API obsolète

Évaluation des paramètres

Solution 1 : ne pas utiliser SLF4J

- Difficile en pratique

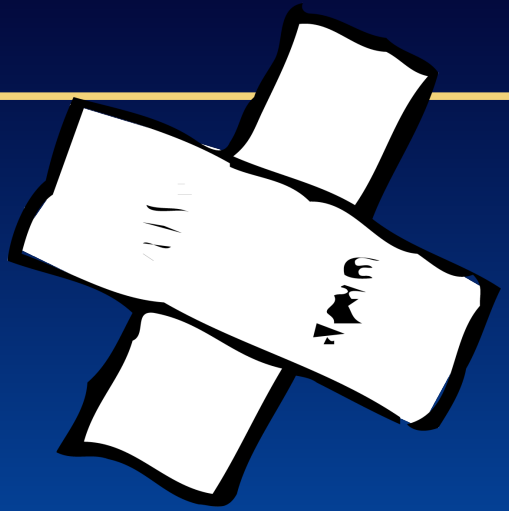
Solution 2 : Améliorer SLF4J

- Rétrocompatibilité obligatoire

Solution 3 : "Façader" SLF4J

- LogService

Problème #3



Factory statique

```
private static final Logger LOG =
  LogManager.getLogger(Foo.class);
```

Factory statique

```
private static final Logger LOG = LogManager.getLogger(Foo.class);
```

Référence statique

- Non configurable
- Non testable
- Non mockable

Factory statique

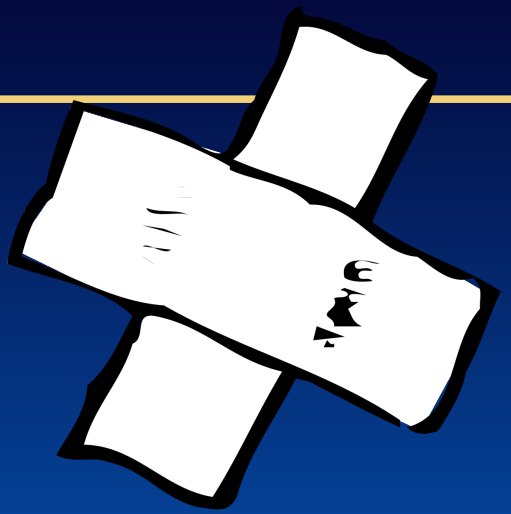
Solution : référence non-statique

Notion de "service de log"

- Injectable par IOC
- Testable (test du service de log)
- Mockable (test des services utilisateurs)
- AOP-able (performances, coupe-circuit)

Problème #4

Des frameworks trop bas niveau



```
log.debug("process "+pId+" started")
log.info("PROC_START;pid="+pId);
```

Des frameworks trop bas niveau

Non-DRY : harmonisation externe nécessaire

- Niveau de log
- Format des messages

Couplage fort

- Producteur / consommateur
- Framework de log

Des frameworks trop bas niveau

Solution : un service de log haut niveau

- Façade du framework technique
- Expose une API technique et métier *ad hoc*
- Centralise la gestion des messages (format, niveau)

```
public void beginProcess(int id) {
    logger.info (() → "Starting process "+id);
}
```

Des frameworks trop bas niveau

Solution : un service de log haut niveau

- Possibilité de raisonner en termes d'événements
- Un événement → `1..N` logs

```
public void restApiCalled(String url, long ms, int code) {
    if (ms ≥ 10)     perfLogger.info (() → url+";"+ms);
    if (code != 200) alertLogger.warn(() → url+";"+code);
}
```

LogService

LogService

Design goals

- Une API restreinte et moderne
- Rétro-compatibilité et extensibilité
- Façade pour SLF4J
- Composant IOC

Une API restreinte et moderne

Méthode "`log`" fondamentale

- Couvre tous les use-cases
- `Supplier` résout la plupart des problèmes

```
interface LogService {
    void log( LogLevel level, Supplier<String> msg, Throwable ex );
}
```

Rétro-compatibilité

Rétro-compatibilité via les méthodes dérivées

```
default void debug (Supplier<String> msg, Throwable ex) {
    log(LogLevel.DEBUG, msg, ex);
}

default void info (Supplier<String> msg, Throwable ex) {
    log(LogLevel.INFO, msg, ex);
}

(...)
```

Extensibilité

Possibilité d'ajouter ses propres méthodes orientées métier

```
void restApiCalled(String path, long ms, int code);
void userLogged(Principal user, Device device);
```

Façade pour SLF4J

```
public class SLF4JLogService implements LogService {

    private final Logger logger;

    protected SLF4JLogService(String loggerName) {
        this.logger = LoggerFactory.getLogger(loggerName);
    }

    @Override
    public void log(LogLevel lvl, Supplier<String> msg, Throwable ex) {
        log(LogLevelConfig.forLogLevel(lvl), msg, ex);
    }

}
```

Composant IOC

Factory toujours nécessaire pour le use-case "1 logger par classe"

- Plus simple qu'un composant de scope "prototype"

```
public interface LogServiceFactory {

    LogService getLogService(String loggerName);

    default LogService getLogService(Class<?> loggerNameByClass) {
        return getLogService(loggerNameByClass.getClass());
    }

}
```

Composant IOC

```
@Component
public class SLF4JLogServiceFactory implements LogServiceFactory {

    @Override
    public LogService getLogService(String loggerName) {
        return new SLF4JLogService(loggerName);
    }

}
```

Utilisation

```
@Service
class MyService {

    private final LogService logService;

    public MyService(LogServiceFactory factory) {
        this.logService = factory.getLogService(MyService.class);
    }

    public void foo() {
        logService.info("Hello World");
    }

}
```

Conclusion

Conclusion

Un domaine en pleine évolution…

- Big Data et analyse temps réel
- Micro-services et Cloud

…mais encore mal industrialisé

- Frameworks et patterns en retard
- Peu de spécifications dans les projets

Conclusion

Beaucoup reste à faire

- Sensibilisation DevOps
- Nouveaux frameworks et patterns à inventer

Voir aussi :

- ExceptionContext : trying to improve business exceptions
  `http://bit.ly/2nQQqB5`

Question ?

Merci !

Log me tender

Thank you !

Olivier Croisier
@OlivierCroisier

Moka Technologies
mokatech.net

DEVOXX France