内配工具集

*Ump 自动监控插件（**ump-profiler**）*

What's ump-profiler?

ump-profiler 是一个基于 spring-aop 机制 ,通过注解方式优雅地进行 UMP 监控的组件。通过 ump-profiler，你可以以一种熟悉而又舒适的方式进行 UMP 监控。

Why ump-profiler?

众所周知，京东要求 service 级别的绝大多数方法添加 UMP 监控，这样运维和开发就可以细粒度地监控方法运行的状况（异常、响应时间等）。ump 监控提供了一个依赖：

```
<dependency>
    <groupId>com.jd.ump</groupId>
    <artifactId>profiler</artifactId>
    <version>3.1.0</version>
</dependency>
```

具体到API是：

```
try {
    //some operations here...
} catch (ExceptionA e) {
    logger.error("error a");
    Profiler.functionError(callerInfo);
} catch (ExceptionB e) {
    logger.error("error b");
    Profiler.functionError(callerInfo);
} catch (ExceptionC e) {
    logger.error("error b");
    Profiler.functionError(callerInfo);
}finally {
    logger.error("error b");
    Profiler.registerInfoEnd(callerInfo);
}
```

多么熟悉的处理方式啊，但这真的就是我们想要的吗？

- 为了这个监控功能，我们需要将所有的代码“绑架”于一个 try-catch 中
- 必须在每一个 catch 块中添加 UMP 的选项，重复严重
- 业务逻辑中充斥这这些“外围代码 “，整洁与优雅已经沦为传说
- **不只是 ump 监控，其他诸如：关键方法访问日志、关键方法加强校验、服务软开关等许多带有“切面“属性的东西，你都打算写在代码里？**

ump-profiler 只是开始，美好的事情即将发生。

How To Use ump-profiler?

ump-profiler 基于 spring-aop 模块，所以要求项目基于 spring 构建，如果项目没有添加 spring-aop 依赖，则自动添加 aop 依赖。

ump-profiler 的使用非常简单，完全类似于 spring-tx 注解驱动的方式。here we go....

1. 添加 ump-profiler 依赖

```
<dependency>
    <groupId>com.jd.ipc.inner</groupId>
    <artifactId>ump-profiler</artifactId>
    <version>0.0.1 </version>
</dependency>
```

2. Xml 中引入 profiler 命名空间，并且配置一些基本信息

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:profiler="http://code.360buy.com/schema/profiler"
    xsi:schemaLocation="
     http://code.360buy.com/schema/profiler
     http://code.360buy.com/schema/profiler.xsd
 ">
    <profiler:annotation-dirven
        base-package="com.jd.distribution"
        heart-beats-key="inner_trans.system"
        jvm-info-key="inner_trans.jvm" />
</beans>
```

其中：

base-package 表示想要监控的包路径（该包及其子包）；

其它两项分别对应这两个 API：

```
Profiler.InitHeartBeats(key);
Profiler.registerJVMInfo(key);
```

3. 在待监控方法上加入@ Monitored 注解

```
@Monitored("my_uml_key")
public boolean operationA() {
        System.out.println("operation a");
        return true;
}
@Monitored("my_uml_key2")
public void operationB() {
        throw new MakeUmpAlerm("报警信息");
        return;
}
```

配置完毕，赶快测试一下吧~~~**（demo 在本文档的同级目录中）**

代码结构是不是清晰了许多？更多便捷的用法，请参看`@Monitored`注解的 api。

NOTE

ump-profiler 虽然方便，但是也不免会有一些无法触及到“角落”。所以 ump-profiler 允许手动处理监控信息，并且完全不会冲突。**这也意味着不需要对现有代码进行哪怕一行的改动**（原来的 UMP 初始化建议去掉）。

Performance

关于 uml-profiler 性能的讨论其实可以延伸为对 spring-aop 性能的讨论。

可以确定的是，以 cglib 为代表的，基于字节码增强的技术已经受到业界的广泛接受。

这里仍然提供了一些压测数据供参考：

- 测试工具

Apache Jmeter

- 模拟行为

  10 个 thread 并发，每线程请求 500 次。

- 结果

- 更多数据

  docs.codehaus.org/display/AW/AOP+Benchmark

已通过测试的 spring-versions

测试区间集中在 Spring-3.0.x → Spring-4.1.x

经测试，该区间内的所有 spring 版本都已通过。

Q&A

- 为什么 **ump-profiler** 在 **spring4** 下无法通过单元测试？
  a) spring-test 中的 api 有所变化，请使用 ump-profiler-test 验证功能
-