**SLF4J** и немного в конце о **Logpack**

Где-то в 2006 году один из отцов-основателей log4j — Ceki Gülcü — решает выйти из стремительно тухнущей команды. Так появляется на свет очередная «обертка всего» под названием SLF4J (Simple Logging Facade for Java). Теперь это обертка вокруг: log4j, JUL, commons-logging и нового логгера под названием logback. Как видно, прогресс быстро дошел до стадии «обертка вокруг обертки». *Нетрудно спрогнозировать, что по той же схеме число обертываемых библиотек будет расти как факториал.* Однако SLF4J предлагает и другие прочие выверты. Это специальные binary-переходники: из log4j в SLF4J, из commons-logging в SLF4J и тому подобное. Делаются такие переходники для кода, исходники которого недоступны; при этом они должны подменить оригинальные JAR-ы лог-библиотек. Не берусь представить себе, какая каша при этом образуется, но если очень хочется, то можно и так.  
  
При всей моей ненависти к оберткам, положа руку на сердце, SLF4J — хорошо сделанный продукт. Были учтены все недостатки предшественников. Например, вместо шаманских плясок с поиском классов в CLASSPATH придумана более надежная схема. Теперь вся обертка делится на две части — API (который используется приложениями) и Реализация, которая представлена отдельными JAR-файлами для каждого вида логгирования (например, slf4j-log4j12.jar, slf4j-jdk14.jar и т.д.). Теперь достаточно только подключить к проекту нужный файл Реализации, после чего — опа! весь код проекта и все используемые библиотеки (при условии, что они обращаются к SLF4J API) будут логгировать в нужном направлении.  
  
Функционально SLF4J поддерживал все современные навороты типа NDC и MDC. Помимо собственно обертывания вызовов, SLF4J предлагал небольшой, но полезный бонус при форматировании строк. Бонус тут в следующем. В коде часто приходится печатать конструкции вида:

log.debug("User " + user + " connected from " + request.getRemoteAddr());

Помимо собственно печати строки, тут неявно произойдет преобразование user.toString() с последующей конкатенацией строк. Все бы ничего. В отладочном режиме скорость выполнения нас не очень волнует. Однако даже если мы выставим уровень, скажем, в INFO, окажется, что конструирование строки все равно будет происходить! Никаких чудес: строка конструируется *перед* вызовом log.debug, поэтому log4j не имеет возможности как-то это контролировать. Если представить, что этот log.debug размещен в каком-то критическом внутреннем цикле… в общем, так жить нельзя. Разработчики log4j предложили обрамлять отладочный код так:

if (log.isDebugEnabled()) {

log.debug("User " + user + " connected from " + request.getRemoteAddr());

}

Нехорошо получается. По идее все эти проблемы должна брать на себя сама logging-библиотека. Эта проблема стала просто ахиллесовой пятой log4j. Разработчики вяло реагировали на пинки, рассказывая, что в logging-вызовы теперь можно еще добавить объект (ровно один!), да еще описать, как этот объект будет записан в лог с помощью интерфейса ObjectRenderer. По большому счету, все это были отмазки и полумеры.  
  
SLF4J не был стиснут рамками совместимости со старыми версиями JDK и API, поэтому с ходу предложил более изящное решение:

log.debug("User {} connected from {}", user, request.getRemoteAddr());

В общем-то, все просто. В данной строке {} — это ссылки на параметры, которые передаются отдельно. Преобразование параметров в строку и окончательное форматирование лог-записи происходит *только* при установленном уровне DEBUG. Параметров можно передавать много. Работает! Не надо писать обрамляющий if и прочую тупость!  
  
В скобках надо отметить, что данную возможность также совершенно неожиданно реализовал язык Groovy, где есть понятие GString, т.е. строка вида

"User ${user} connected from ${request.getRemoteAddr()}"

, которая неявно связана с несколькими контекстными переменными (здесь это user, request), причем вычисление строки происходит *отложенным образом*. Это очень удобно для таких лог-библиотек как log4j — можно получить на вход GString, а затем или выбросить его без вычисления, или все-таки преобразовать в нормальную (статическую) строку — String.  
  
Короче говоря, SLF4J был сделан грамотно, с заделом на будущее. Это вызвало серьезный рост его популярности среди сообщества: сейчас SLF4J используют такие значимые проекты, как Jetty, Hibernate, Mina, Geronimo, Mule, Wicket, Nexus… в общем, практически все неудачники, зависшие в свое время на commons-logging, перешли на SLF4J. Интересно, что мешало усовершенствовать commons-logging до нужного состояния много лет назад? Но таковы реалии Open Source — развитие софта в нем происходит скорее революционно, чем эволюционно.  
  
Одновременно с SLF4J был подан к столу совершенно новый логгер — Logback. Он был сделан человеком, который на логгировании собаку съел, и на поверку действительно оказался хорошим продуктом. Logback был изначально заточен под JDK 1.5+, одним махом избавившись от всех старческих болезней обратной совместимости, свойственных проекту log4j. А это значит — varargs, java.util.concurrent и [прочие прелести](http://logback.qos.ch/reasonsToSwitch.html). Например, за счет встроенной системы runtime-фильтрации можно менять уровень логгирования в зависимости от пользовательской сессии, разбрасывать пользователей по разным лог-файлам и прочее, прочее.