

# Зачем нужны логи в Java

JSP & Servlets  
5 уровень, 0 лекция

ОТКРЫТА

## 1.1 Знакомство с логами

**Лог** – это список произошедших событий. Почти как морской журнал или дневник. Ну а, соответственно, **логгер** — это объект, с помощью которого можно вести логирование. В программировании принято логировать практически все. А в Java – так вообще все и даже немного больше.

```
// in class 1
private Logger LOG = LoggerFactory.getLogger(...);

// in class 2
private static final Logger LOGGER = ....;

// in class 3
private static Logger log = Logger.getLogger(...);

// in class 4
private Logger LOG_INSTANCE = ...;

// etc, etc,
```

Дело в том, что Java-программы – это очень часто большие серверные приложения без UI, консоли и так далее. Они обрабатывают одновременно запросы тысяч пользователей и частенько при этом возникают различные ошибки. Особенно, когда разные нити начинают друг другу мешать.

И, фактически, единственным способом поиска редко воспроизводимых ошибок и сбоев в такой ситуации есть **запись в лог/файл всего**, что происходит в каждой нити.

Чаще всего в лог пишется информация о параметрах метода, с которыми он был вызван, все перехваченные ошибки, и еще много промежуточной информации. Чем полнее лог, тем легче восстановить последовательность событий и отследить причины возникновения сбоя или ошибки.

Но чем больше лог, тем сложнее с ним работать. Иногда логи достигают нескольких гигабайт в сутки. Это нормально.

## 1.2 Неудачные логи

В качестве первых логов разработчики использовали просто **вывод в консоль**. Это удобно делать во время дебага приложения – когда в консоль пишется вся важная информация и значения переменных. Но такой лог совершенно неприменим во время обычной работы приложения.

Во-первых, **приложение может хотеть само что-то вывести в консоль и пользователь совсем не хочет видеть служебную информацию, предназначенную программисту.**

Во-вторых, **размер буфера консоли ограничен, много туда не напишешь.**

И наконец, **в-третьих, информацию об ошибках программы, которая собирается за долгое время, нужно отправить разработчикам программы. И удобнее всего писать всю эту информацию сразу в файл.**

Первую проблему разработчики быстро решили – придумали еще один поток вывода – `System.err`. Вы можете писать в него сообщения и они будут отправляться в отдельный поток, а не на стандартную консоль.

И даже проблему с записью в файл удалось решить:

```
1 // Определяем файл в который будем писать лог
2 System.setErr(new PrintStream(new File("log.txt")));
3 // Выводим сообщения
4 System.err.println("Сообщение 1");
5 System.err.println("Сообщение 2");
6 // Выводим сообщение об ошибке
7 try {
8     throw new Exception("Сообщение об ошибке");
9 } catch (Exception e) {
10     e.printStackTrace();
11 }
```

Но даже в таком виде, это не решало всей проблемы, поэтому было решено создать специальную библиотеку, которая бы писала сообщения лога в файл. Она делала это по-умному и позволяла гибко настраивать фильтры логируемых событий и данных.

Весь процесс логирования, по сути, состоит из трех частей:

- Первая часть – это **сбор** информации.
- Вторая часть – это **фильтрация** собранной информации.
- Третья часть – это **запись** отобранной информации.

### 1.3 Знакомство с логгером log4j

Первым популярным логгером в Java-сообществе стал логгер `log4j`. Подключить его в проект очень просто, для этого нужно добавить всего пару строк в ваш pom.xml

```
1 <dependency>
2     <groupId>log4j</groupId>
3     <artifactId>log4j</artifactId>
4     <version>2.2.17</version>
5 </dependency>
```

Взаимодействие твоей программы с таким логгером выглядело бы примерно так:

```
1 class Manager {
2     private static final Logger logger = LoggerFactory.getLogger(Manager.class);
3
4     public boolean processTask(Task task) {
5         logger.debug("processTask id = " + task.getId());
6         try {
7             task.start();
8             task.progress();
9             task.complete();
10            return true;
11        } catch (Exception e) {
12            logger.error("Unknown error", e);
13            return false;
14        }
15    }
16 }
```

Тут происходит три вещи:

Зеленым цветом выделено создание объекта `Logger`. Его объект сохраняется в статическую переменную для удобной дальнейшей работы с ним. А также в метод `getLogger()` передается информация о классе, в котором происходит сбор информации.

Синим цветом выделена строка, где мы логируем информацию представляющую ценность только во время дебага. Для этого используется специальный метод – `debug()`

Ну и наконец красным цветом выделена строка, где мы сохраняем в лог возникшее исключение. Исключения – это потенциальные ошибки, поэтому для записи в лог используется метод `error()`.

<   Предыдущая лекция

Следующая лекция   >



Комментарии (1)

популярные   новые   старые

JavaCoder

Введите текст комментария

Musa Muradzade

Уровень 20, Санкт-Петербург, Россия

18 июня, 01:27

...

1.2 .... придумалі ...

Ответить

-

+2

+

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

 Русский

▼

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

 ДСТУПНО В  
Google Play

 Загрузите в  
App Store

