Уровни событий

JSP & Servlets
5 уровень, 2 лекция

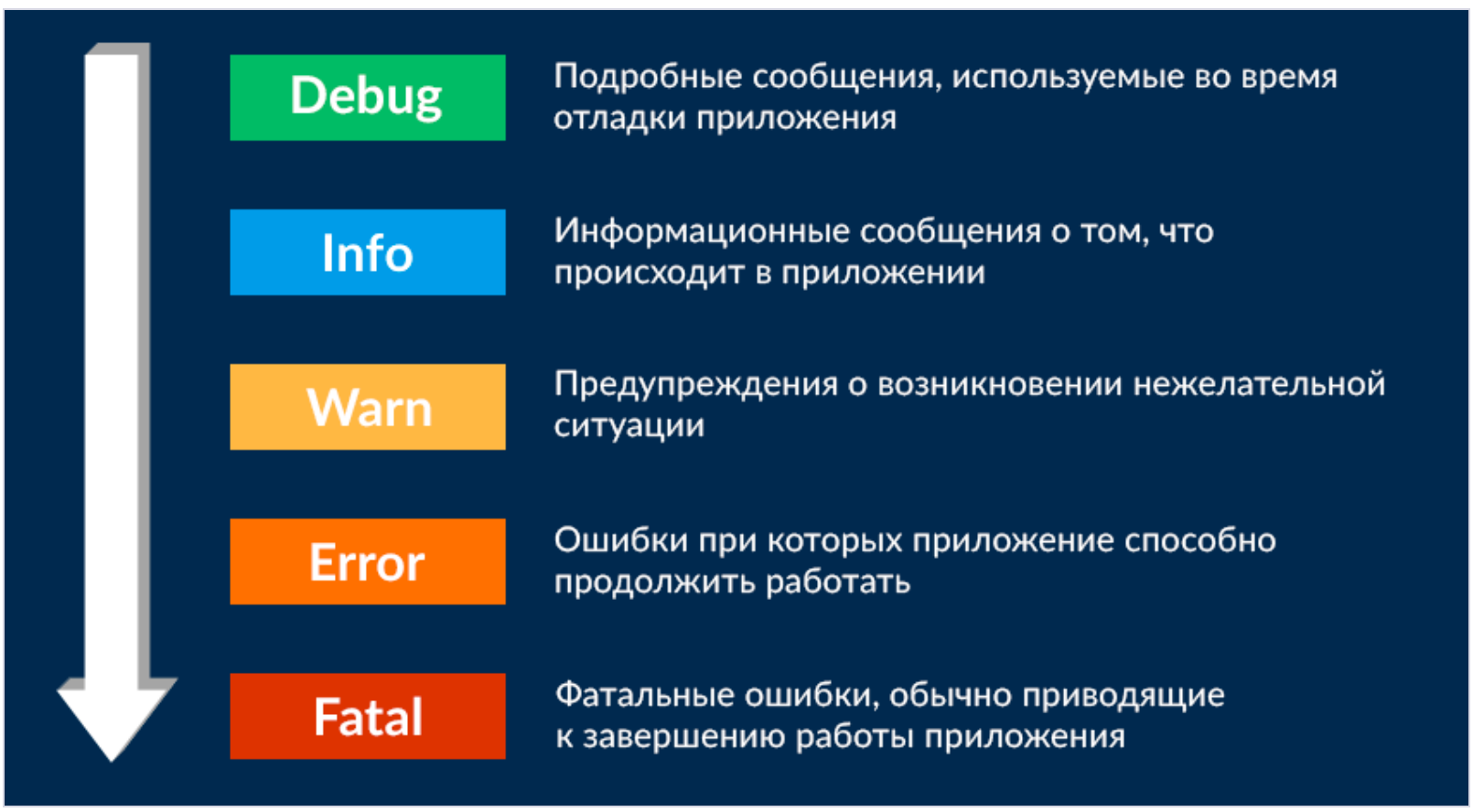
ОТКРЫТА

3.1 Список уровней событий

Логирование – это процесс записи каких-либо событий, которые происходят во время работы программы. Ваша обязанность как программиста — `запротоколировать все важное`, потому что потом, когда на `production` будут странные и/или серьезные ошибки, кроме этих логов у вас больше ничего не будет.

Любая ошибка будет устранена в разы быстрее, если у вас будет вся информация о ней и обо всех предысториях вызовов. Но отсюда следует простой вывод – логировать вообще все: вызовы всех методов, значения всех параметров.

Это тоже не выход – слишком много информации так же плохо, как и слишком мало. Нам нужно умное логирование. Сделанное человеком для человека. И тут мы подходим к первому факту про логирование – все записи в лог еще во время их создания делятся на категории.



Программист, когда пишет какое-нибудь событие в лог, должен сам решить, насколько это важная информация. Уровень важности события выбирает автор сообщения. В `log4j` существует 5 уровней важности логируемой информации:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`
- `FATAL`

Ниже расскажем о них подробнее.

3.2 DEBUG

Уровень `DEBUG` считается наименее важным. Информация, которая пишется в лог с таким уровнем важности, нужна только во время дебага приложения. Для того чтобы записать в лог информацию нужную во время дебага используется метод `debug()`.

Пример:

```java
class Manager {
    private static final Logger logger = LoggerFactory.getLogger(Manager.class);

    public boolean processTask(Task task) {
        logger.debug("processTask id = " + task.getId());
        try {
            task.start();
            task.progress();
            task.complete();
            return true;
        } catch (Exception e) {
            logger.error("Unknown error", e);
            return false;
        }
    }
}
```

Обрати внимание, метод `debug` находится в самом начале метода (метод еще не успел ничего сделать) и пишет в лог значение переданной в метод переменной. Это самый частый сценарий использования метода `debug()`.

3.3 INFO и WARN

Следующие два уровня – это `INFO` и `WARN`. Для них существуют два метода – `info()` и `warn()`.

Уровень `INFO` используется просто для информационных сообщений: происходит то-то и то-то. Когда начинаешь разбор ошибки в логе, бывает очень полезно почитать ее предысторию. Для этого отлично подходит метод `info()`.

Уровень `WARN` используется для записи предупреждений (от слова **warning**). Обычно с таким уровнем важности пишется информация о том, что что-то пошло не так, но программа знает, как поступить в данной ситуации.

Например, в процессе записи файла на диск, выяснилось, что такой файл уже существует. Тут программа может записать в лог предупреждение (warning), но показать пользователю диалоговое окно и предложить выбрать другое имя файла.

Пример:

```java
class FileManager {
    private static final Logger logger = LoggerFactory.getLogger(FileManager.class);

    public boolean saveFile(FileData file) {
        logger.info(“сохраняем файл ” + file.getName());
        boolean resultOK = SaveUtils.save(file);
        if (resultOK) return true;

        logger.warn(“проблема с записью файла ” + file.getName());
        String filename = Dialog.selectFile();
        boolean result = SaveUtils.save(file, filename);
        return result;
    }
```

3.4 ERROR и FATAL

И наконец два самых важных уровня логирования – `ERROR` и `FATAL`. Для них тоже есть специальные методы с одноименными названиями: `error()` и `fatal()`.

Ошибки тоже решили разделить на две категории – обычные ошибки и **фатальные ошибки**. Фатальная ошибка чаще всего приводит к аварийному закрытию приложения (для десктопных приложений) или падению веб-сервиса (для веб-приложений).

Еще хороший пример – это операционная система Windows. Если у тебя просто упала программа, то с точки зрения операционной системы – это `Error`. А если упала сама операционная система и вы видите синий экран смерти Windows, то это уже `Fatal error`.

В Java-приложениях чаще всего события `Error` и `Fatal` связаны с возникающими исключениями. Пример:

```
class Manager {
    private static final Logger logger = LoggerFactory.getLogger(Manager.class);

    public boolean processTask(Task task) {
        logger.debug("processTask id = " + task.getId());
        try {
            task.start();
            task.progress();
            task.complete();
            return true;
        } catch (Exception e) {
            logger.error("Unknown error", e);
            return false;
        }
    }
}
```

3.5 Что нужно логировать

Разумеется, логировать все подряд не стоит. В большинстве случаев это резко ухудшает читабельность лога, а ведь лог пишется в первую очередь для того, чтобы его читали.

Кроме того, нельзя писать в лог различную личную и финансовую информацию. Сейчас с этим строго и легко можно нарваться на штрафы или судебные процессы. Рано или поздно такой лог утечет на сторону и тогда проблем на оберешься.

Так что же нужно логировать?

Во-первых, нужно логировать **начало работы приложения**. После того как приложение запустилось, рекомендуется вывести в лог его режим работы и различные важные настройки – так будет проще в будущем читать лог.

Во-вторых, нужно логировать **состояние всех третьесторонних сервисов**, с которыми твое приложение работает: системы рассылок, любые внешние сервисы. Как минимум нужно залогировать момент подключения к ним, чтобы убедиться, что они штатно работают.

В-третьих, логировать нужно **все исключения**. Если они ожидаемые, то информацию по ним можно записать компактно. Полная информация об исключениях дает 50%-80% важной информации при поиске ошибки.

Также нужно логировать **завершение работы приложения**. Приложение должно завершаться штатно и не сыпать при этом в лог десятки ошибок. Часто в этом месте можно найти подвисшие задачи, проблемы с пулом потоков или проблемы с удалением временных файлов.

Обязательно логируй то, что связано с **безопасностью и авторизацией пользователя**. Если пользователь 10 раз подряд пробует залогиниться или сбросить пароль, эта информация должна быть отражена в логах.

Логируй максимум **информации об асинхронных задачах** – часто исключения в таких потоках теряются. По асинхронной задаче обязательно логируй ее старт и завершение. Успешное завершение нужно логировать так же, как и проблемное.

Что еще? Запуск задач, выполняемых по таймеру, запуск хранимых `SQL-процедур`, синхронизацию данных, все, что касается распределенных транзакций. Думаю, для начала хватит. В будущем ты сам дополнишь этот список.

‹ Предыдущая лекция

Следующая лекция ›

+16

Комментарии (2) популярные новые старые

JavaCoder

Введите текст комментария

Михаил Уровень 51, Санкт-Петербург, Russian Federation 14 сентября, 14:16

В примере с INFO в 7й строке не хватает возвращаемого значения "true"

Ответить 0

Jacque Frescov Уровень 28, Russian Federation 30 августа, 00:08

У тебя упала программа - Error
Вышел синий экран - Fatal error
Хейтеры, люблю вас!

Ответить +3

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store