Theory: Introduction to logging

7 minutes 0 / 5 problems solved

§1. Introduction

Logging refers to the activity recording during the execution of an application. Logs are **records** that give us information about **events** in a software application. This record could be a message that is enough to understand the event, so it may include a timestamp, a description, and the severity level. These events could be **user-generated** or **system generated**. We use a log file (or, sometimes, standard output) to write these records.

A lot of developers find logging boring and try to skip it. However, proper logging will save you a lot of time when you are **troubleshooting** the application at a late stage. There are several things to be considered when we do logging. They are: **when we log? what we log? how we log?** Let's find out the answers to those.

§2. When, What and How

At runtime, an application sometimes runs into errors or behaves in an unexpected way. Logs are a great way of figuring out what went wrong. Capturing logs helps you understand the behavior of your application at runtime. They also help you debug errors and analyse how your users work with your application.

There are several common reasons to generate logs:

- troubleshooting
- auditing
- profiling
- statistics

What we log usually depends on the application. Anyhow, we should be able to understand the execution path of a program through the logs. It is important to avoid excessive logging as it is costly. For example, there's no need to log the start and end of every method, their arguments: they are easy to track. Logs were meant to cover **server issues, database issues, networking issues, errors from unanticipated user inputs, states of dynamically created objects, configuration values.**

It is discouraged to use your own log tools. There are a lot of libraries and frameworks dedicated to logging which you should try. If your application is based on an **OOP language**, log instance should be called for each class.

Providing **contextual information** in your log messages is very important as well. Often, the success or the failure of a program depends on the **user inputs**. So, you need to put them in your log messages where necessary. For example, when authenticating a user, log the "username" that is inputted. Context is also important when your program runs in a **concurrent environment**. In such a case thread name can be added to the log message.

§3. Log levels

Logs need to be done at proper levels. There are several log levels. Depending on the library, these levels can be slightly different. The most common ones are Debug, Info, Warn, Error, Fatal (from the least critical level to the most critical one).

Let's see what those log levels are for.

Debug – debug logs are used to **diagnose applications**. It is used by many people other than developers such as sysadmins, testers. Values of variables are logged at the debug level.

Info – used to log important information about an application. It is used to log service start, service stop, configurations, assumptions.

Current topic:

Introduction to logging

Topic is required for:

Standard logger

Table of contents:

↑ Introduction to logging
§1. Introduction
§2. When, What and How
§3. Log levels
§4. An example of logs
Feedback & Comments

Warn – warns are used to indicate a potential problem in the application. However, it does not affect the user at this point. Warns are considered the **first level** of application failure. Warns are usually applied to log repeated attempts of accessing a resource, missing secondary data, switching from a primary server to a back-up server.

Error – this log level is used when there is a more critical problem. This kind of issue usually affects the operation result but does not terminate the program. Errors are considered the **second level** of application failures.

Fatal – it is the third level of application failures. It is used to indicate much more serious error which causes the **termination of the program**.

§4. An example of logs

Here is an example of logs in a hypothetical web application in which users log in. Each record includes a timestamp, a description, and a level of severity.

```
[2019-02-02 01:00:10] [INFO] User 'demo' has registered
[2019-02-02 01:00:20] [INFO] User 'demo' has been logged
[2019-02-02 01:30:05] [INFO] User 'demo' has left the site
[2019-02-02 02:20:00] [ERROR] User 'demo' cannot log in because database is temporarily unavailable
```

It is good when logs allow finding the causes of various errors. Therefore, try to log all the important information.

Report a typo

169 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion