LOGGING

12 January 2025 18:15

Logging is the process of recording events, messages, or other information during the execution of a program, application, or system. It is an essential tool for developers, system administrators, and analysts to monitor, debug, and understand the behavior of software and systems.

Key Features of Logging

1. **Information Capture**: Logs can record:
   - Errors and exceptions
   - Warnings and critical issues
   - General information about program flow
   - Debugging details
2. **Levels of Logging**: Logging systems usually support levels that help categorize the importance or severity of messages:
   - **DEBUG**: Detailed information for debugging purposes.
   - **INFO**: General operational information.
   - **WARNING**: Indication of potential issues or unexpected events.
   - **ERROR**: Errors that prevent part of the program from functioning properly.
   - **CRITICAL**: Severe errors causing the program or system to crash or fail.
3. **Format**: Log messages often include details like:
   - Timestamps
   - Severity level
   - Source of the message (e.g., file, module, or function)
   - Custom message
4. **Output Destinations**:
   - Console/terminal
   - Log files
   - External systems (e.g., monitoring tools, databases)
5. **Benefits**:
   - Simplifies debugging and troubleshooting
   - Helps monitor application performance
   - Provides audit trails for security and compliance
   - Allows proactive problem detection in production systems

Example in Python

Python's built-in logging module provides a flexible framework for creating and managing log messages.

```
import logging
# Configure the logging system
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
# Example log messages
logging.debug("This is a debug message.")
logging.info("This is an informational message.")
logging.warning("This is a warning message.")
logging.error("This is an error message.")
logging.critical("This is a critical message.")
```

Applications

- Software development (debugging and diagnostics)
- System and server monitoring
- Event tracking in distributed systems
- Security audits and compliance logs

Logging is a fundamental practice in software engineering that ensures transparency and maintainability across systems.

What is logger

12 January 2025 18:25

A **logger** is a component or object within a logging framework used to create and manage log messages. It serves as the main interface through which developers write log entries. Loggers are designed to record events, messages, or errors during the execution of an application.

Key Concepts of a Logger

1. **Log Message Creation**:
    - A logger allows you to generate log messages at various levels (e.g., DEBUG, INFO, WARNING, ERROR, CRITICAL).
    - Each message usually includes a timestamp, severity level, and additional context.
2. **Hierarchical Structure**:
    - Loggers often follow a hierarchical naming convention. For example, in Python's logging module:
        - A logger named app could have child loggers like app.module1 or app.module2.
        - This structure allows fine-grained control over logging configurations for specific parts of an application.
3. **Handlers and Propagation**:
    - A logger doesn't directly write log messages to a destination (like a file or console). Instead, it passes messages to **handlers** that define where the logs are sent.
    - Loggers can propagate log messages to parent loggers unless propagation is explicitly disabled.
4. **Configuration**:
    - Loggers can be configured to determine which messages should be logged (based on severity levels) and how those messages should be formatted.
    - This configuration is done globally or per logger instance.

Benefits of Using a Logger:

- Flexibility: Loggers allow different parts of an application to log messages independently.
- Scalability: Supports multiple handlers and complex configurations.
- Debugging and Monitoring: Helps diagnose issues during development and monitor application behavior in production.

In summary, a logger is a versatile and powerful tool for managing log messages in a structured and efficient manner.

Best Practices

1. **Choose Appropriate Log Levels**:
    - Use DEBUG for development and INFO or higher in production.
2. **Avoid Logging Sensitive Data**:
    - Be cautious with logging user information, passwords, or personal data.
3. **Use Rotating Logs**:
    - Prevent log files from consuming too much disk space.
4. **Structure Logs**:
    - Include contextual information like request IDs, user IDs, or function names.

Logger in python

12 January 2025 18:26

Example in Python

Here's how a logger is used in Python's built-in logging module:

```
import logging
# Create a logger
logger = logging.getLogger('my_logger')
# Set the logging level for this logger
logger.setLevel(logging.INFO)
# Create a handler (e.g., to write logs to a file)
file_handler = logging.FileHandler('app.log')
# Create a formatter and set it for the handler
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)
# Add the handler to the logger
logger.addHandler(file_handler)
# Generate some log messages
logger.debug("This is a debug message.")
logger.info("This is an informational message.")
logger.warning("This is a warning message.")
logger.error("This is an error message.")
logger.critical("This is a critical message.")
```

Explanation of Components:

1. **Logger (logger)**:
    - Created using logging.getLogger().
    - Responsible for generating log messages.
2. **Handler (file_handler)**:
    - Determines where to send the log messages (e.g., file, console, etc.).
3. **Formatter (formatter)**:
    - Defines the format of the log messages.
4. **Log Levels**:
    - Only messages at or above the logger's level (INFO in this case) will be logged.

Setup and Configuring a Logger

12 January 2025 18:27

```
# basic setup of logger

import logging

# Basic configuration
logging.basicConfig(
    level=logging.INFO,  # Minimum log level
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'  # Log format
)

# Log some messages
logging.debug("This is a debug message.")
logging.info("This is an info message.")
logging.warning("This is a warning message.")
logging.error("This is an error message.")
logging.critical("This is a critical message.")
```

```
# configuration of logger
mport logging

# Create a custom logger
logger = logging.getLogger('my_logger')

# Set log level for the logger
logger.setLevel(logging.DEBUG)

# Create a handler
console_handler = logging.StreamHandler()  # Logs to console
file_handler = logging.FileHandler('app.log')  # Logs to a file

# Set log level for handlers
console_handler.setLevel(logging.WARNING)
file_handler.setLevel(logging.DEBUG)

# Create and attach a formatter
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)

# Add handlers to the logger
logger.addHandler(console_handler)
logger.addHandler(file_handler)

# Log messages
logger.debug("Debug message (only in file).")
logger.info("Info message (only in file).")
logger.warning("Warning message (in console and file).")
logger.error("Error message (in console and file).")
```