
🎯 Understanding Logging Levels in Spring Boot

🔴 TL;DR Answer:

If you set the log level to `ERROR` in `application.properties`, **only error logs and higher** (like FATAL, if configured) will be shown — even if your code has `info`, `debug`, `trace`, etc.

📘 Detailed Explanation

🧩 Code Example

```
@Service
@Slf4j
public class UserService {

    public void registerUser(String username) {
        log.trace("Entering registerUser method with username: {}", username);

        if (username == null || username.isEmpty()) {
            log.warn("Username is empty or null!");
            return;
        }

        log.debug("Checking if username {} is already taken...", username);

        if ("admin".equalsIgnoreCase(username)) {
            log.error("Registration failed: Username '{}' is reserved!", username);
            return;
        }

        log.info("User '{}' registered successfully!", username);
        log.trace("Exiting registerUser method...");
    }

    public void getUserDetails(String username) {
        log.trace("Entering getUserDetails method for username: {}", username);

        if ("testUser".equalsIgnoreCase(username)) {
            log.debug("Fetching details for test user.");
```

```
            log.info("User details found for '{}'", username);
        } else {
            log.warn("User '{}' not found in the system!", username);
        }

        log.trace("Exiting getUserDetails method...");
    }
}
```

⚙️ Config in `application.properties`:

logging.level.root=ERROR

❓ What Will Happen?

Only `log.error(...)` will appear in your logs.
All other logs (`trace`, `debug`, `info`, `warn`) will be ignored **at runtime**, because the logging framework filters out anything below the configured level.

📊 Log Level Hierarchy (From highest to lowest verbosity):

OFF > FATAL > ERROR > WARN > INFO > DEBUG > TRACE > ALL

🤔 Then Why Write All Levels in Code?

1. Future Flexibility

You might run the app in `ERROR` level in production to reduce log noise, but during:

- Local development
- Debugging
- QA/UAT testing

…you can temporarily set the level to `DEBUG` or `TRACE` **without changing the code**.

2. Granular Control

You can configure log levels per package or per class:

logging.level.com.mycompany.userservice=DEBUG
logging.level.root=ERROR

This lets you:

- Keep global logs quiet
- Be verbose in only one area temporarily

3. Environment-Specific Behavior

- Development → `DEBUG` or `INFO`
- Production → `WARN` or `ERROR`

This helps you:

- Avoid performance hits
- Keep logs clean in production
- Still get full insights during dev/testing

✅ Final Thought:

> **Logging in code = potential log statements**
> **Log level in properties = what actually gets printed**

So, keep `log.debug()` or `log.trace()` in the code — it's like leaving **optional breadcrumbs**.
You can enable/disable them **dynamically** through `application.properties` without touching the code again.
