The purpose of this page is to discuss logging practices in the Spring Framework.

Development Logging

Logging in development needs to be selected and presented with care to maximize value and lower the noise-to-signal ratio. This section aims to define what that means more concretely.

DEBUG vs TRACE

DEBUG and TRACE log levels should work *together* as a two-tier system. That doesn't mean having most messages at DEBUG level and some at TRACE, but rather having a definition of what belongs at each level, and in some cases having intelligently designed messages that show nuanced output for DEBUG vs TRACE.

The purpose of *DEBUG logging* is to provide feedback on what's happening in a way that is a) minimal, b) presented in a compact way, and c) focused on high value bits of information that are generally useful over and over again, as opposed to being useful for debuggin a specific issue, i.e. the 80/20 rule. DEBUG level is the window display of the development experience. It should provide a reasonable, out-of-the-box experience, without the need to tweak settings by package to avoid a fire hose.

TRACE logging follows the same mindset as DEBUG, except for the 80/20 rule. It should still aim to be minimal, compact, and focused on value, but also allow more extended information to be logged, including some for specific issues. Like DEBUG, TRACE aims to avoid a fire hose but it's expected that log settings by package may need to be customized to obtain the right amount of detail for the problem at hand.

The Fine-Tuning Process

Log messages can only be calibrated in the full context of log output for specific scenarios. It's impossible otherwise to spot issues such as duplication (e.g. request path logged in many places), inconsistent level of detail (vs other related components), verbosity, and other unintended consequences. We cannot gauge overall effectiveness in isolation either.

Duplication and verbosity may seem like small issues. Surely we can err on the side of extra information? For once that adds noise without value, but more importantly consider what isn't done. To address such issues we have to consider where certain information is best shown relative to this or other scenarios, we must iterate over wording, and we must think about the relative value of information, and potentially what else we might be better to show instead.

All of that is extra work of course, but just like we need tests to prove code works, iterating over and reviewing actual logging output is the only way to create good logging and a good development experience.

Questions To Ask

Q: Is the information relevant when solving a specific issue (TRACE), or is it important enough to see all the time in development (DEBUG)?

For example, HTTP request mappings (request conditions, handlers, and methods with full signatures) is very useful information, but do you really need/want to see hundreds or thousands of those on every startup? More likely, most of the time, we're forced to skip over that information, and in the process miss out on other potentially important bits.

Instead we could show summary information by `HandlerMapping`:

```
17:04:11.919 [main] DEBUG RequestMappingHandlerMapping - Detected 84 mappings in 'requestMap
17:04:11.941 [main] DEBUG SimpleUrlHandlerMapping - Patterns [/] in 'viewControllerHandlerMa
17:04:11.965 [main] DEBUG SimpleUrlHandlerMapping - Patterns [/resources/**] in 'resourceHan
```

That's minimal, compact, and aids with understanding. It gives assurance about what handler mappings are present, along with valuable bits that can fit in such compact form, and also not forgetting to add the bean name, which is essential for recognizing different `HandlerMapping` instances. Note that `BeanNameHandlerMapping` was also present for the above output (with `@EnableWebMvc`) but did not add a message at DEBUG level since it did not find any mappings. It does log at TRACE level, when we may suspect an issue or want more information.

Q: What is the most compact way to present the information?

This requires iterating over a log message until it's compact, and verbosity is reduced without loss of meaning.

For example:

```
Processing GET request for [/spring-mvc-showcase/data/param]
```

Becomes:

```
GET /spring-mvc-showcase/data/param
```

Or perhaps we can add more high value (request parameters at DEBUG, plus also headers at TRACE):

```
GET /spring-mvc-showcase/data/param, parameters={foo:[bar]}
```

Q: Is there an opportunity to add more value?

For example:

```
Successfully completed request
```

Becomes:

```
Completed 304 NOT_MODIFIED
```

Q: Does a word duplicate what is already known from the category name? If so, consider dropping it.

For example:

```
org.springframework.web.servlet.view.freemarker.FreeMarkerView - Rendering FreeMarker templa
```

Becomes:

```
org.springframework.web.servlet.view.freemarker.FreeMarkerView - Rendering [foo/bar.ftl]
```

Q: What value does a log message provide? Could, and should, it be left out?

For examples `FreeMarkerView` can be ordered ahead of `JstlView`, as it can check if a template is present and hence yield to the next strategy. Seeing the message "No FreeMarker view found for URL ..." seems useful on its own, but when followed by a message from `JstlView` forwarding to a JSP page, it adds unnecessary noise even at TRACE level. The only thing it helps to confirm is that FreeMarker is ordered ahead of JSPs which we don't need to re-confirm on every request.

Q: How much volume does it add vs how much value does it bring?

Showing all request mappings for annotated methods at DEBUG level is arguably too much volume to see by default in development, but for `SimpleUrlHandlerMapping` it's a different situation since showing all patterns (but not handlers) is both feasible and valuable.

On a single HTML page there may be 20-30, or more static resources to load. For a fine-grained component such as `CachingResourceResolver` we can't afford to show much, if anything at all, at DEBUG level. It just doesn't bring enough value for the volume it adds. Showing cache add/remove operations creates a lot of noise even at TRACE level. We might however log a small message "Resource served from cache" whenever a resource is served from cache, which tells us that the `ResourceResolver` chain was bypassed, and we could show that at TRACE based on expected average volume vs value.