UI Errror Handling and Logging

Unable to render {include} The included page could not be found.

UI Errror Handling and Logging

Introduction

Currently the UI does not cope well with errors - often displaying Java errors / stack traces - and there is no logging. This document discusses how to improve the situation.

Desired user experience

Ideally there wouldn't be any errors - or at least any undefined errors. Things will go wrong - errors in coding, databases unavailable, bad input due to mangled URLs etc. The user should see a concise (localized) message with an indication of what they should do - and that someone else may have been alerted to a problem. This could include providing a *ticket* which they can use to follow up the error. If they do follow up on the error, they should expect that a support person should be able to use the *ticket* to look up details of the error.

Functional requirements

When an error arises in the UI a *ticket* should be generated. Sufficient details about the error and its context should be logged against the *ticket* so that there is no ambiguity - particularly important in a multi-user environment. Some errors may be hard to reproduce so capturing relevant context is important. Currently the API uses several different log files which makes it difficult to know which messages may be related.

System requirements

Typically production systems do not log debug information - the log files become too large and there can be IO issues. On the other hand, when a serious error occurs it would, in general, be useful to have the low-level debug information. I intend to look into whether we can capture debug (possibly tagged as info) information but only log it in the case of an error. This may well not go in 1.3.0 but may be possible using a custom log4j Appender.

Log files often have little discriminatory information e.g. grouper_error.log shows no information about the subject associated with the GrouperSession. grouper_event.log does have information, however, the API sometimes starts internal GrouperSystem sessions to complete its work so it is not always clear that events are associated. From a UI perspective - and I assume a Web services perspective, it would be best to tag each request with a *ticket* which unambiguously *groups* all relevant messages. A simple grep allows all associated messages to be viewed - even if they are interleaved with concurrent messages from another request.

Error messages shown to end users should be derived from nav.properties.

There is potentially a lot of relevant contextual information in request parameters and request/session attributtes which should be captured - and possibly emailed to an approriate account.

Edited comments from others from the 'descriptive error messages' thread

> Chris:
> 1. You don't want the descriptive error message to cause another exception which will pre-empt the original exception, so you need to be careful. My method GrouperUtil.toStringForLog is a failsafe toString method
> 2. Its hard to tell where these exceptions are going (to the log, or to the end user), but you have to be sensitive that you aren't causing a potential security problem by giving too much info. In this case for web services the web service client implementer would see this exception, and I think its not a huge security problem to know the potential sources, and it would be great for the client implementer to be able to solve this issue without having to bother the grouper administrator who can look in the sources.xml file... right?
> ...in this case, there is no exception if member not found, so there is an error logged. However, in some cases this is a real error, and in some cases not (member not found means not a member). In the error log, I think error-level means real error (or at least more real than this). So I think it can either be not logged, or logged more like info (I would lean more toward info instead of warn).
> In addition, I don't like having a delegate class doing the logging since the log4j log mask might want to print class / method / line# where the thing was logged, and this always prints the ErrorLog class, error method, and same line number... very inconvenient. I think we should change all these... eventually 🙂 There is another way to do it where you tell log4j what the delegate is, in which case we need to move all direct logging to the delegate way, but I think the delegate isn't buying us much so we might as well simplify and make more universal...

> Tom B:
> Why don't you open a jira issue. Seems like a "good thing" to fix, but not at the top of the list of burning issues. Maybe for 1.3.1?

I definitely think we should review logging for 1.3.1, and look at when exceptions really are errors vs *informational* within the API.

Proposal

I propose to provide an alternative log4j.properties which can, by configuration in grouper-ui/build.properties, be used in place of the Grouper provided default. The new configuration will log all messages to a single file, and all messages for the same request will be tagged with a unique ticket and will contain GrouperSession information where I can control that. I will use loggers directly rather than utility classes

I will go through the servlet Filters and Struts actions and add better exception handling and logging. There will probably be many new error messages