Plataforma Java

Java Logging

Java Logging

1. Conceito
2. Criar um Logger
3. Byte Stream
4. Text Stream
5. Data Stream
6. Object Stream
7. Buffered Read/Write

Java Logging

Conceito

É uma atividade comum e necessária que uma aplicação gere informações para atividades de detecção de problemas

Este registro de atividade é denominado *Logging* e é realizado em um local centralizado

O objeto responsável pela escrita dos registros de logging é chamado *Logger*

Normalmente cada registro de log é categorizado (ERROR, WARN, INFO, DEBUG, etc)

O Java fornece um framework padrão para Logging encontrado no pakage *java.util.logging*

Existem diversos frameworks de Logging: Log4J, SLF4J, Logback, etc

Java Logging

Logger

Para registrar logs é necessário um objeto do tipo Logger

Normalmente cada classe possui um Logger com o nome da classe

Os loggers são organizados em forma de hierarquia, por exemplo: um logger para a chave com.exemplo é filho do logger para com, que por sua vez é filho do logger para String vazio

```
import java.util.logging.Logger;

// assumes the current class is called MyLogger
private final static Logger LOGGER =
Logger.getLogger(MyLogger.class.getName());
```

Java Logging

Logger Level

A API de Logger padrão do Java fornece os seguintes níveis de Log:

- ERROR
- WARN
- INFO
- FINE
- FINNER
- FINEST

Se o nível de log da aplicação está configurado para INFO, entãos qualquer registro de log INFO ou superior (WARN e ERROR) será registrado. Se estiver FINE, então qualquer registro de log FINE ou superior (INFO, WARN e ERROR) será registrado, e assim sucessivamente

Java Logging

Logger Handler

Handler são os objetos responsáveis por receber o registro de Log e enviar a um destino específico (console, arquivo, etc)

Handlers padrão:

- ConsoleHandler: Envia os registros de log para o console
- FileHandler: Envia os registro de log para arquivo

Os níveis INFO ou superior são enviados automaticamente para o Console

Java Logging

Logger Formatter

Cada Handler pode ser formatado com um objeto Formatter

Formatadores padrão:

- SimpleFormatter: Formatam os registros de log em texto simples
- XMLFormatter: Formatam os registro de log em XML

Caso desejado é possível criar seus próprios formatadores

```
import java.util.logging.Formatter;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.LogRecord;

// this custom formatter formats parts of a log record to a single line
class MyHtmlFormatter extends Formatter {
    // this method is called for every log records
    public String format(LogRecord rec) {
        StringBuffer buf = new StringBuffer(1000);
        buf.append("<tr>\n");

        // colorize any levels >= WARNING in red
        if (rec.getLevel().intValue() >= Level.WARNING.intValue()) {
            buf.append("\t<td style=\"color:red\">");
            buf.append("<b>");
            buf.append(rec.getLevel());
            buf.append("</b>");
        } else {
            buf.append("\t<td>");
            buf.append(rec.getLevel());
        }
```

Java Logging

Logger Manager

Objeto responsável pelo gerenciamento da configuração de objetos Logger

```java
LogManager.getLogManager().getLogger(Logger.GLOBAL_LOGGER_NAME).setLevel(Level.FINE);
```

Java Logging

Registrando Logs

```
public class UseLogger {
    // use the classname for the logger, this way you can refactor
    private final static Logger LOGGER =
Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);


    public void doSomeThingAndLog() {
        // ... more code

        // now we demo the logging

        // set the LogLevel to Severe, only severe Messages will be written
        LOGGER.setLevel(Level.SEVERE);
        LOGGER.severe("Info Log");
        LOGGER.warning("Info Log");
        LOGGER.info("Info Log");
        LOGGER.finest("Really not important");
```