日志技术（上）

0. 学习目标

1. 日志的作用和目的
2. 日志的框架
3. JUL的使用
4. LOG4J的使用
5. JCL的使用

1. 日志的概念

1.1 日志文件

日志文件是用于记录系统操作事件的文件集合，可分为事件日志和消息日志。具有处理历史数据、诊断问题的追踪以及理解系统的活动等重要作用。

在计算机中，日志文件是记录在操作系统或其他软件运行中发生的事件或在通信软件的不同用户之间的消息的文件。记录是保持日志的行为。在最简单的情况下，消息被写入单个日志文件。

许多操作系统，软件框架和程序包括日志系统。广泛使用的日志记录标准是在因特网工程任务组（IETF）RFC5424中定义的syslog。 syslog标准使专用的标准化子系统能够生成，过滤，记录和分析日志消息。

1.1.1 调试日志

软件开发中，我们经常需要去调试程序，做一些信息，状态的输出便于我们查询程序的运行状况。为了让我们能够更加灵活和方便的控制这些调试的信息，所有我们需要专业的日志技术。java中寻找bug会需要重现。调试也就是debug 可以在程序运行中暂停程序运行，可以查看程序在运行中的情况。日志主要是为了更方便的去重现问题。

1.1.2 系统日志

系统日志是记录系统中硬件、软件和系统问题的信息，同时还可以监视系统中发生的事件。用户可以通过它来检查错误发生的原因，或者寻找受到攻击时攻击者留下的痕迹。系统日志包括系统日志、应用程序日志和安全日志。

系统日志的价值

系统日志策略可以在故障刚刚发生时就向你发送警告信息，系统日志帮助你在最短的时间内发现问题。

系统日志是一种非常关键的组件，因为系统日志可以让你充分了解自己的环境。这种系统日志信息对于决定故障的根本原因或者缩小系统攻击范围来说是非常关键的，因为系统日志可以让你了解故障或者袭击发生之前的所有事件。为虚拟化环境制定一套良好的系统日志策略也是至关重要的，因为系统日志需要和许多不同的外部组件进行关联。良好的系统日志可以防止你从错误的角度分析问题，避免浪费宝贵的排错时间。另外一种原因是借助于系统日志，管理员很有可能会发现一些之前从未意识到的问题，在几乎所有刚刚部署系统日志的环境当中。

2. JAVA日志框架

问题：

1. 控制日志输出的内容和格式
2. 控制日志输出的位置
3. 日志优化：异步日志，日志文件的归档和压缩
4. 日志系统的维护
5. 面向接口开发 -- 日志的门面

2.1 为什么要用日志框架

因为软件系统发展到今天已经很复杂了，特别是服务器端软件，涉及到的知识，内容，问题太多。在某些方面使用别人成熟的框架，就相当于让别人帮你完成一些基础工作，你只需要集中精力完成系统的业务逻辑设计。而且框架一般是成熟，稳健的，他可以处理系统很多细节问题，比如，事务处理，安全性，数据流控制等问题。还有框架一般都经过很多人使用，所以结构很好，所以扩展性也很好，而且它是不断升级的，你可以直接享受别人升级代码带来的好处。

2.2 现有的日志框架

JUL（java util logging）、logback、log4j、log4j2

JCL（Jakarta Commons Logging）、slf4j（ Simple Logging Facade for Java）

日志门面

JCL、slf4j

日志实现

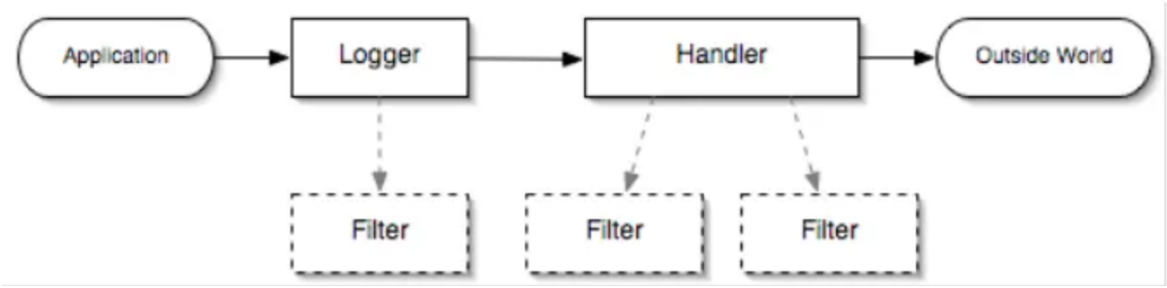
JUL、logback、log4j、log4j2

3. JUL 学习

JUL全称Java util Logging是java原生的日志框架，使用时不需要另外引用第三方类库，相对其他日志框架使用方便，学习简单，能够在小型应用中灵活使用。

3.1 JUL入门

3.1.1 架构介绍



- Loggers：被称为记录器，应用程序通过获取Logger对象，调用其API来来发布日志信息。Logger通常时应用程序访问日志系统的入口程序。
- Appenders：也被称为Handlers，每个Logger都会关联一组Handlers，Logger会将日志交给关联Handlers处理，由Handlers负责将日志做记录。Handlers在此是一个抽象，其具体的实现决定了日志记录的位置可以是控制台、文件、网络上的其他日志服务或操作系统日志等。
- Layouts：也被称为Formatters，它负责对日志事件中的数据进行转换和格式化。Layouts决定了数据在一条日志记录中的最终形式。
- Level：每条日志消息都有一个关联的日志级别。该级别粗略指导了日志消息的重要性和紧迫，我可以将Level和Loggers，Appenders做关联以便于我们过滤消息。
- Filters：过滤器，根据需要定制哪些信息会被记录，哪些信息会被放过。

总结一下就是：

用户使用Logger来进行日志记录，Logger持有若干个Handler，日志的输出操作是由Handler完成的。在Handler在输出日志前，会经过Filter的过滤，判断哪些日志级别过滤放行哪些拦截，Handler会将日志内容输出到指定位置（日志文件、控制台等）。Handler在输出日志时会使用Layout，将输出内容进行排版。

3.1.2 入门案例

```
public class JULTest {

    @Test
    public void testQuick() throws Exception {
        // 1.创建日志记录器对象
        Logger logger = Logger.getLogger("com.itheima.log.JULTest");
        // 2.日志记录输出
        logger.info("hello jul");

        logger.log(Level.INFO, "info msg");

        String name = "jack";
        Integer age = 18;
        logger.log(Level.INFO, "用户信息：{0},{1}", new Object[]{name, age});
    }
}
```

3.2 日志的级别

jul中定义的日志级别

```
* java.util.logging.Level中定义了日志的级别：
        SEVERE（最高值）
        WARNING
        INFO     （默认级别）
        CONFIG
        FINE
        FINER
        FINEST（最低值）

* 还有两个特殊的级别：
        OFF，可用来关闭日志记录。
        ALL，启用所有消息的日志记录。
```

虽然我们测试了7个日志级别但是默认只实现info以上的级别

```
@Test
public void testLogLevel() throws Exception {
    // 1.获取日志对象
    Logger logger = Logger.getLogger("com.itheima.log.QuickTest");
    // 2.日志记录输出
    logger.severe("severe");
    logger.warning("warning");
    logger.info("info");
    logger.config("cofnig");
    logger.fine("fine");
    logger.finer("finer");
    logger.finest("finest");
}
```

自定义日志级别配置

```
@Test
public void testLogConfig() throws Exception {
    // 1.创建日志记录器对象
    Logger logger = Logger.getLogger("com.itheima.log.JULTest");

    // 一、自定义日志级别
    // a.关闭系统默认配置
    logger.setUseParentHandlers(false);
    // b.创建handler对象
    ConsoleHandler consoleHandler = new ConsoleHandler();
    // c.创建formatter对象
    SimpleFormatter simpleFormatter = new SimpleFormatter();
    // d.进行关联
    consoleHandler.setFormatter(simpleFormatter);
    logger.addHandler(consoleHandler);
    // e.设置日志级别
    logger.setLevel(Level.ALL);
    consoleHandler.setLevel(Level.ALL);


    // 二、输出到日志文件
    FileHandler fileHandler = new FileHandler("d:/logs/jul.log");
    fileHandler.setFormatter(simpleFormatter);
```

```
    logger.addHandler(fileHandler);


    // 2.日志记录输出
    logger.severe("severe");
    logger.warning("warning");
    logger.info("info");
    logger.config("config");
    logger.fine("fine");
    logger.finer("finer");
    logger.finest("finest");
}
```

3.3 Logger之间的父子关系

JUL中Logger之间存在父子关系，这种父子关系通过树状结构存储，JUL在初始化时会创建一个顶层RootLogger作为所有Logger父Logger，存储上作为树状结构的根节点。并父子关系通过路径来关联。

```
@Test
public void testLogParent() throws Exception {
    // 日志记录器对象父子关系
    Logger logger1 = Logger.getLogger("com.itheima.log");
    Logger logger2 = Logger.getLogger("com.itheima");

    System.out.println(logger1.getParent() == logger2);
    // 所有日志记录器对象的顶级父元素  class为java.util.logging.LogManager$RootLogger
name为""
    System.out.println("logger2 parent:" + logger2.getParent() + ", name: " +
logger2.getParent().getName());


    // 一、自定义日志级别
    // a.关闭系统默认配置
    logger2.setUseParentHandlers(false);
    // b.创建handler对象
    ConsoleHandler consoleHandler = new ConsoleHandler();
    // c.创建formatter对象
    SimpleFormatter simpleFormatter = new SimpleFormatter();
    // d.进行关联
    consoleHandler.setFormatter(simpleFormatter);
    logger2.addHandler(consoleHandler);
    // e.设置日志级别
    logger2.setLevel(Level.ALL);
    consoleHandler.setLevel(Level.ALL);


    // 测试日志记录器对象父子关系
    logger1.severe("severe");
    logger1.warning("warning");
    logger1.info("info");
    logger1.config("config");
    logger1.fine("fine");
    logger1.finer("finer");
    logger1.finest("finest");
}
```

3.4 日志的配置文件

默认配置文件路径$JAVAHOME\jre\lib\logging.properties

```
@Test
public void testProperties() throws Exception {
    // 读取自定义配置文件
    InputStream in =
JULTest.class.getClassLoader().getResourceAsStream("logging.properties");
    // 获取日志管理器对象
    LogManager logManager = LogManager.getLogManager();
    // 通过日志管理器加载配置文件
    logManager.readConfiguration(in);

    Logger logger = Logger.getLogger("com.itheima.log.JULTest");
    logger.severe("severe");
    logger.warning("warning");
    logger.info("info");
    logger.config("config");
    logger.fine("fine");
    logger.finer("finer");
    logger.finest("finest");
}
```

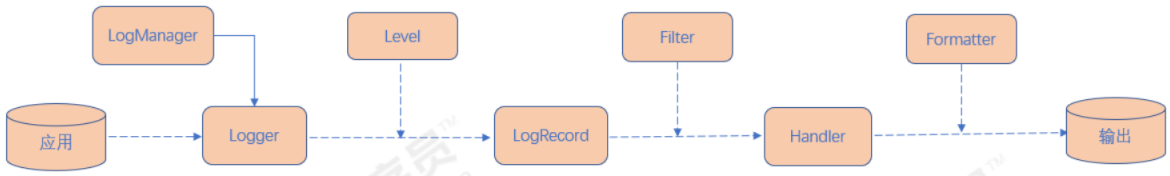
配置文件：

```
## RootLogger使用的处理器（获取时设置）
handlers= java.util.logging.ConsoleHandler
# RootLogger日志等级
.level= INFO

## 自定义Logger
com.itheima.handlers= java.util.logging.FileHandler
# 自定义Logger日志等级
com.itheima.level= INFO
# 忽略父日志设置
com.itheima.useParentHandlers=false


## 控制台处理器
# 输出日志级别
java.util.logging.ConsoleHandler.level = INFO
# 输出日志格式
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter


## 文件处理器
# 输出日志级别
java.util.logging.FileHandler.level=INFO
# 输出日志格式
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
# 输出日志文件路径
java.util.logging.FileHandler.pattern = /java%u.log
# 输出日志文件限制大小（50000字节）
```

```
java.util.logging.FileHandler.limit = 50000
# 输出日志文件限制个数
java.util.logging.FileHandler.count = 10
# 输出日志文件 是否是追加
java.util.logging.FileHandler.append=true
```

3.5 日志原理解析

1. 初始化LogManager
    1. LogManager加载logging.properties配置
    2. 添加Logger到LogManager
2. 从单例LogManager获取Logger
3. 设置级别Level，并指定日志记录LogRecord
4. Filter提供了日志级别之外更细粒度的控制
5. Handler是用来处理日志输出位置
6. Formatter是用来格式化LogRecord的



JUL流程示意图

4. LOG4J 学习

Log4j是Apache下的一款开源的日志框架，通过在项目中使用 Log4j，我们可以控制日志信息输出到控制台、文件、甚至是数据库中。我们可以控制每一条日志的输出格式，通过定义日志的输出级别，可以更灵活的控制日志的输出过程。方便项目的调试。

官方网站： http://logging.apache.org/log4j/1.2/

4.1 Log4j入门

1. 建立maven工程
2. 添加依赖

```xml
<dependencies>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

3. java代码

```java
public class Log4jTest {

    @Test
    public void testQuick() throws Exception {

        // 初始化系统配置，不需要配置文件
        BasicConfigurator.configure();

        // 创建日志记录器对象
        Logger logger = Logger.getLogger(Log4jTest.class);
        // 日志记录输出
        logger.info("hello log4j");

        // 日志级别
        logger.fatal("fatal");  // 严重错误，一般会造成系统崩溃和终止运行
        logger.error("error");  // 错误信息，但不会影响系统运行
        logger.warn("warn");    // 警告信息，可能会发生问题
        logger.info("info");    // 程序运行信息，数据库的连接、网络、IO操作等
        logger.debug("debug");  // 调试信息，一般在开发阶段使用，记录程序的变量、参数等
        logger.trace("trace");  // 追踪信息，记录程序的所有流程信息
    }
}
```

4. 日志的级别

```
* 每个Logger都被了一个日志级别（log level），用来控制日志信息的输出。日志级别从高到低分为：
        fatal 指出每个严重的错误事件将会导致应用程序的退出。
        error 指出虽然发生错误事件，但仍然不影响系统的继续运行。
        warn  表明会出现潜在的错误情形。
        info  一般和在粗粒度级别上，强调应用程序的运行全程。
        debug 一般用于细粒度级别上，对调试应用程序非常有帮助。
        trace 是程序追踪，可以用于输出程序运行中的变量，显示执行的流程。

* 还有两个特殊的级别：
        OFF，可用来关闭日志记录。
        ALL，启用所有消息的日志记录。
```

> 注：一般只使用4个级别，优先级从高到低为 ERROR > WARN > INFO > DEBUG

4.2 Log4j组件

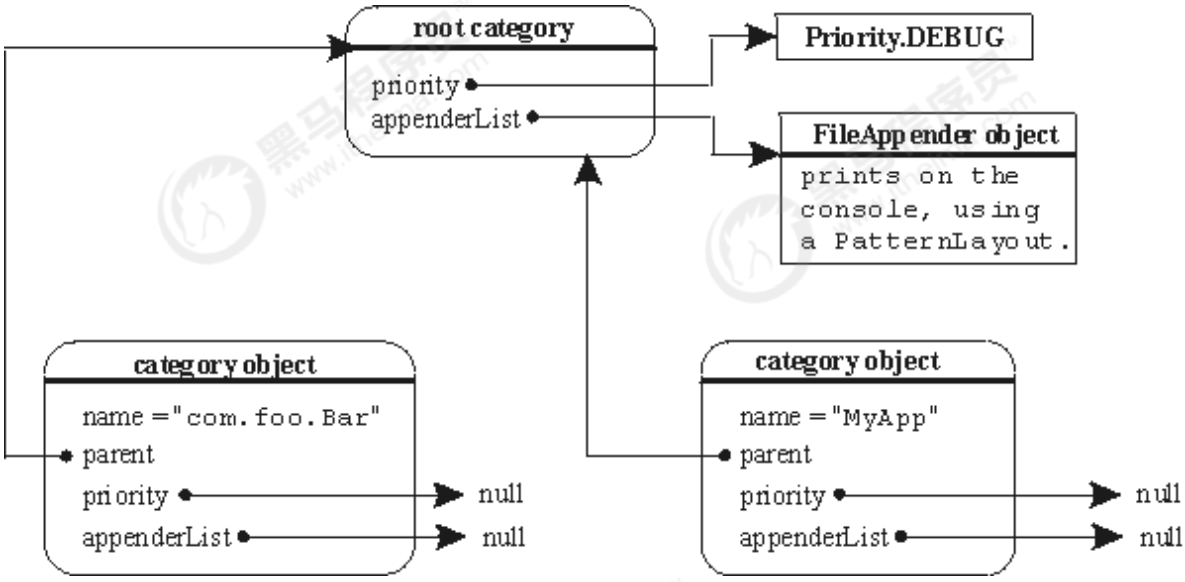
Log4J 主要由 Loggers (日志记录器)、Appenders（输出端）和 Layout（日志格式化器）组成。其中 Loggers 控制日志的输出级别与日志是否输出；Appenders 指定日志的输出方式（输出到控制台、文件等）；Layout 控制日志信息的输出格式。

4.2.1 Loggers

日志记录器，负责收集处理日志记录，实例的命名就是类"XX"的full quailied name（类的全限定名），Logger的名字大小写敏感，其命名有继承机制：例如：name为org.apache.commons的logger会继承name为org.apache的logger。

Log4J中有一个特殊的logger叫做"root"，他是所有logger的根，也就意味着其他所有的logger都会直接或者间接地继承自root。root logger可以用Logger.getRootLogger()方法获取。

但是，自log4j 1.2版以来，`Logger`类已经取代了`Category`类。对于熟悉早期版本的log4j的人来说，`Logger`类可以被视为`Category`类的别名。



4.2.2 Appenders

Appender 用来指定日志输出到哪个地方，可以同时指定日志的输出目的地。Log4j 常用的输出目的地有以下几种：

输出端类型	作用
ConsoleAppender	将日志输出到控制台
FileAppender	将日志输出到文件中
DailyRollingFileAppender	将日志输出到一个日志文件，并且每天输出到一个新的文件
RollingFileAppender	将日志信息输出到一个日志文件，并且指定文件的尺寸，当文件大小达到指定尺寸时，会自动把文件改名，同时产生一个新的文件
JDBCAppender	把日志信息保存到数据库中

4.2.3 Layouts

布局器 Layouts用于控制日志输出内容的格式，让我们可以使用各种需要的格式输出日志。Log4j常用的Layouts:

格式化器类型	作用
HTMLLayout	格式化日志输出为HTML表格形式
SimpleLayout	简单的日志输出格式化，打印的日志格式为（info - message）
PatternLayout	最强大的格式化期，可以根据自定义格式输出日志，如果没有指定转换格式，就是用默认的转换格式

4.3 Layout的格式

在 log4j.properties 配置文件中，我们定义了日志输出级别与输出端，在输出端中分别配置日志的输出格式。

```
* log4j 采用类似 C 语言的 printf 函数的打印格式格式化日志信息，具体的占位符及其含义如下：

        %m   输出代码中指定的日志信息
        %p   输出优先级，及 DEBUG、INFO 等
        %n   换行符（Windows平台的换行符为 "\n"，Unix 平台为 "\n"）
        %r   输出自应用启动到输出该 log 信息耗费的毫秒数
        %c   输出打印语句所属的类的全名
        %t   输出产生该日志的线程全名
        %d   输出服务器当前时间，默认为 ISO8601，也可以指定格式，如：%d{yyyy年MM月dd日
HH:mm:ss}
        %l   输出日志时间发生的位置，包括类名、线程、及在代码中的行数。如：
Test.main(Test.java:10)
        %F   输出日志消息产生时所在的文件名称
        %L   输出代码中的行号
        %%   输出一个 "%" 字符

* 可以在 % 与字符之间加上修饰符来控制最小宽度、最大宽度和文本的对其方式。如：

        %5c     输出category名称，最小宽度是5，category<5，默认的情况下右对齐
        %-5c    输出category名称，最小宽度是5，category<5，"-"号指定左对齐,会有空格
        %.5c    输出category名称，最大宽度是5，category>5，就会将左边多出的字符截掉，<5不
会有空格
        %20.30c category名称<20补空格，并且右对齐，>30字符，就从左边交远销出的字符截掉
```

4.4 Appender的输出

控制台，文件，数据库

```
#指定日志的输出级别与输出端
log4j.rootLogger=INFO,Console

# 控制台输出配置
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n


# 文件输出配置
log4j.appender.A = org.apache.log4j.DailyRollingFileAppender
#指定日志的输出路径
log4j.appender.A.File = D:/log.txt
log4j.appender.A.Append = true
#使用自定义日志格式化器
log4j.appender.A.layout = org.apache.log4j.PatternLayout
#指定日志的输出格式
log4j.appender.A.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [%t:%r] -
[%p]  %m%n
#指定日志的文件编码
log4j.appender.A.encoding=UTF-8

#mysql
log4j.appender.logDB=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.logDB.layout=org.apache.log4j.PatternLayout
log4j.appender.logDB.Driver=com.mysql.jdbc.Driver
log4j.appender.logDB.URL=jdbc:mysql://localhost:3306/test
log4j.appender.logDB.User=root
log4j.appender.logDB.Password=root
log4j.appender.logDB.Sql=INSERT INTO
log(project_name,create_date,level,category,file_name,thread_name,line,all_categ
ory,message) values('itcast','%d{yyyy-MM-dd
HH:mm:ss}','%p','%c','%F','%t','%L','%l','%m')
```

```
CREATE TABLE `log` (
  `log_id` int(11) NOT NULL AUTO_INCREMENT,
  `project_name` varchar(255) DEFAULT NULL COMMENT '目项名',
  `create_date` varchar(255) DEFAULT NULL COMMENT '创建时间',
  `level` varchar(255) DEFAULT NULL COMMENT '优先级',
  `category` varchar(255) DEFAULT NULL COMMENT '所在类的全名',
  `file_name` varchar(255) DEFAULT NULL COMMENT '输出日志消息产生时所在的文件名称 ',
  `thread_name` varchar(255) DEFAULT NULL COMMENT '日志事件的线程名',
  `line` varchar(255) DEFAULT NULL COMMENT '号行',
  `all_category` varchar(255) DEFAULT NULL COMMENT '日志事件的发生位置',
  `message` varchar(4000) DEFAULT NULL COMMENT '输出代码中指定的消息',
  PRIMARY KEY (`log_id`)
);
```

4.5 自定义Logger

```
# RootLogger配置
log4j.rootLogger = trace,console


# 自定义Logger
log4j.logger.com.itheima = info,file
log4j.logger.org.apache = error
```

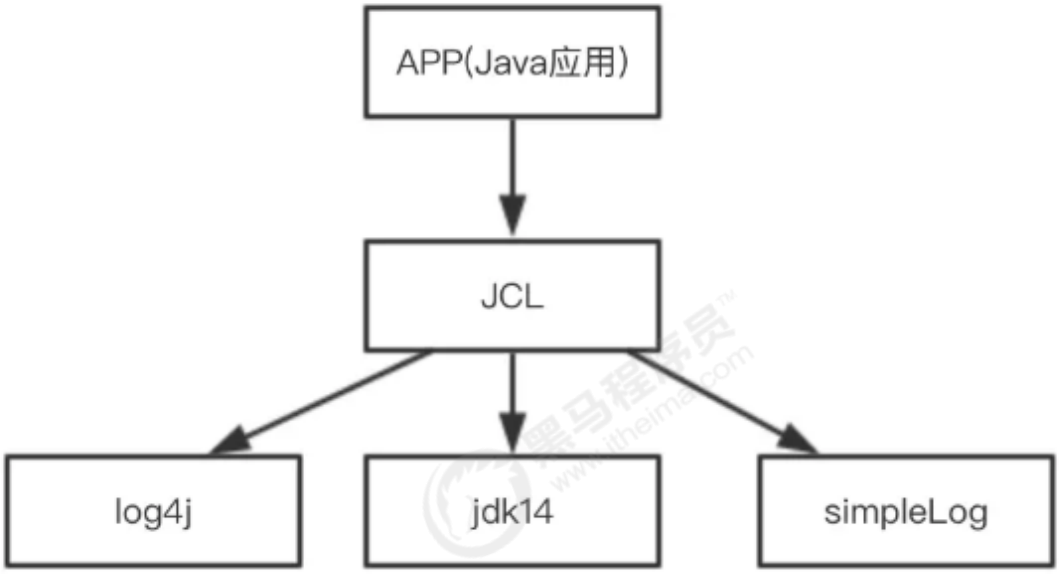
```
@Test
public void testCustomLogger() throws Exception {
    // 自定义 com.itheima
    Logger logger1 = Logger.getLogger(Log4jTest.class);
    logger1.fatal("fatal");  // 严重错误，一般会造成系统崩溃和终止运行
    logger1.error("error");  // 错误信息，但不会影响系统运行
    logger1.warn("warn");    // 警告信息，可能会发生问题
    logger1.info("info");    // 程序运行信息，数据库的连接、网络、IO操作等
    logger1.debug("debug");  // 调试信息，一般在开发阶段使用，记录程序的变量、参数等
    logger1.trace("trace");  // 追踪信息，记录程序的所有流程信息

    // 自定义 org.apache
    Logger logger2 = Logger.getLogger(Logger.class);
    logger2.fatal("fatal logger2");  // 严重错误，一般会造成系统崩溃和终止运行
    logger2.error("error logger2");  // 错误信息，但不会影响系统运行
    logger2.warn("warn logger2");    // 警告信息，可能会发生问题
    logger2.info("info logger2");    // 程序运行信息，数据库的连接、网络、IO操作等
    logger2.debug("debug logger2");  // 调试信息，一般在开发阶段使用，记录程序的变量、参
数等
    logger2.trace("trace logger2");  // 追踪信息，记录程序的所有流程信息
}
```

5. JCL 学习

全称为Jakarta Commons Logging，是Apache提供的一个通用日志API。

它是为 "所有的Java日志实现"提供一个统一的接口，它自身也提供一个日志的实现，但是功能非常常弱（SimpleLog）。所以一般不会单独使用它。他允许开发人员使用不同的具体日志实现工具: Log4j, Jdk自带的日志（JUL)

JCL 有两个基本的抽象类：Log(基本记录器)和LogFactory(负责创建Log实例)。



5.1 JCL入门

1. 建立maven工程
2. 添加依赖

```
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.2</version>
</dependency>
```

3. 入门代码

```
public class JULTest {

    @Test
    public void testQuick() throws Exception {
        // 创建日志对象
        Log log = LogFactory.getLog(JULTest.class);
        // 日志记录输出
        log.fatal("fatal");
        log.error("error");
        log.warn("warn");
        log.info("info");
        log.debug("debug");
    }
}
```

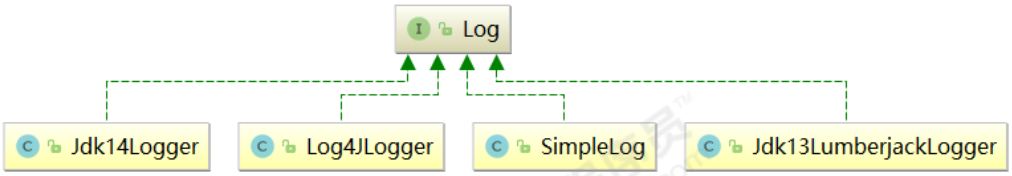
我们为什么要使用日志门面：

1. 面向接口开发，不再依赖具体的实现类。减少代码的耦合
2. 项目通过导入不同的日志实现类，可以灵活的切换日志框架
3. 统一API，方便开发者学习和使用

4. 统一配置便于项目日志的管理

5.2 JCL原理

1. 通过LogFactory动态加载Log实现类



2. 日志门面支持的日志实现数组

```
private static final String[] classesToDiscover =
    new String[]{"org.apache.commons.logging.impl.Log4JLogger",
                 "org.apache.commons.logging.impl.Jdk14Logger",
                 "org.apache.commons.logging.impl.Jdk13LumberjackLogger",
                 "org.apache.commons.logging.impl.SimpleLog"};
```

3. 获取具体的日志实现

```
for(int i = 0; i < classesToDiscover.length && result == null; ++i) {
    result = this.createLogFromClass(classesToDiscover[i], logCategory,
true);
}
```