

제05장

LOG

Spring

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i]))  
    } else  
      for (i in e)  
        if (r = t.call(e[i], i, e[i]))  
  return e  
},  
trim: b && !b.call("\uffff\u00a0") ?  
  return null == e ? "" : b.call(  
} : function(e) {  
  return null == e ? "" : (e + "  
},  
makeArray: function(e, t) {  
  var n = t || [];  
  return null != e && (M(Ob  
},  
isArray: function(e, t, n) {  
  var r;  
  if (t) {  
    if (n) return n.c  
    for (r = t.length;  
      if (n in t)  
  }  
}
```

# 학습목표

1. 로그 프레임워크의 개념에 대해서 알 수 있다.
2. 스프링에서 로그 프레임워크를 활용할 수 있다.

```
each: function(e, t, n) {  
  var r, i = 0,  
      o = e.length,  
      a = M(e);  
  if (n) {  
    if (a) {  
      for (; o > i; i++)  
        if (r = t.apply(e[i], n), r ===  
    } else  
      for (i in e)  
        if (r = t.apply(e[i], n), r ===  
  } else if (a) {  
    for (; o > i; i++)  
      if (r = t.call(e[i], i, e[i]))  
    } else  
      for (i in e)  
        if (r = t.call(e[i], i, e[i]))  
    return e  
  },  
  trim: b && !b.call("\uffff\u00a0") ?  
    return null == e ? "" : b.call(  
  } : function(e) {  
    return null == e ? "" : (e + "  
  },  
  makeArray: function(e, t) {  
    var n = t || [];  
    return null != e && (M(Obj  
  },  
  isArray: function(e, t, n) {  
    var r;  
    if (t) {  
      if (n) return n.c  
      for (n = t.length;  
        if (n in t)  
    }  
  }
```

# 목차

1. 로그 프레임워크 개념
2. 스프링 로그 프레임워크 활용

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break;
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break;
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break;
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e;
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e);
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "");
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; n in t && t[n] === e) return n;
  }
}

```

# 1. 로그 프레임워크 개념

## ■ Log

- 로그
- 애플리케이션이 동작하는 동안의 각종 이벤트를 기록으로 남기는 것
- 개발 기간에는 로그 패키지를 이용하여 디버깅에 활용
- 로그 출력을 특정 파일이나 DB에 저장해 둘 수 있음

## ■ Java Logging Framework

- 자바 애플리케이션을 개발할 때 사용할 수 있는 로그 처리 프레임워크
- `java.util.logging` 패키지, Log4j, Logback등이 존재

# Log4j

## ■ Log4j

- 자바 기반의 Logging 유틸리티
- 가장 오래된 Java Logging Framework
- 프로그램 개발 도중 디버깅을 위해 정해진 양식에 맞춰 화면이나 파일로 기록을 남길 수 있음

기술지원 중

Log4j 2




기술지원 종료

Log4j 1



**1. Apache Log4j Core**8,738 usages  
[org.apache.logging.log4j » log4j-core](https://org.apache.logging.log4j/log4j-core)  
Apache  
Implementation for Apache Log4J, a highly configurable logging tool that focuses on performance and low garbage generation. It has a plugin architecture that makes it extensible and supports asynchronous logging based on LMAX Disruptor.  
Last Release on Sep 17, 2022

**2. Apache Log4j**17,964 usages  
[log4j » log4j](https://log4j/log4j)  
Apache  
Legacy version of Log4J logging framework. Log4J 1 has reached its end of life and is no longer officially supported. It is recommended to migrate to Log4J 2.  
Last Release on May 26, 2012

# Log4j 로그 레벨

## ■ 로그 레벨

- 무분별한 로그로 인해 중요한 로그를 찾는 것이 어려워지는 것을 막기 위해 어떤 내용까지를 로  
그로 남길 것인지  
표준으로 정해 놓은 레벨
- FATAL > ERROR > WARN > INFO > DEBUG > TRACE 순으로 레벨이 정해짐  
(INFO 레벨 적용 시 FATAL, ERROR, WARN, INFO 레벨의 로그가 남겨짐)

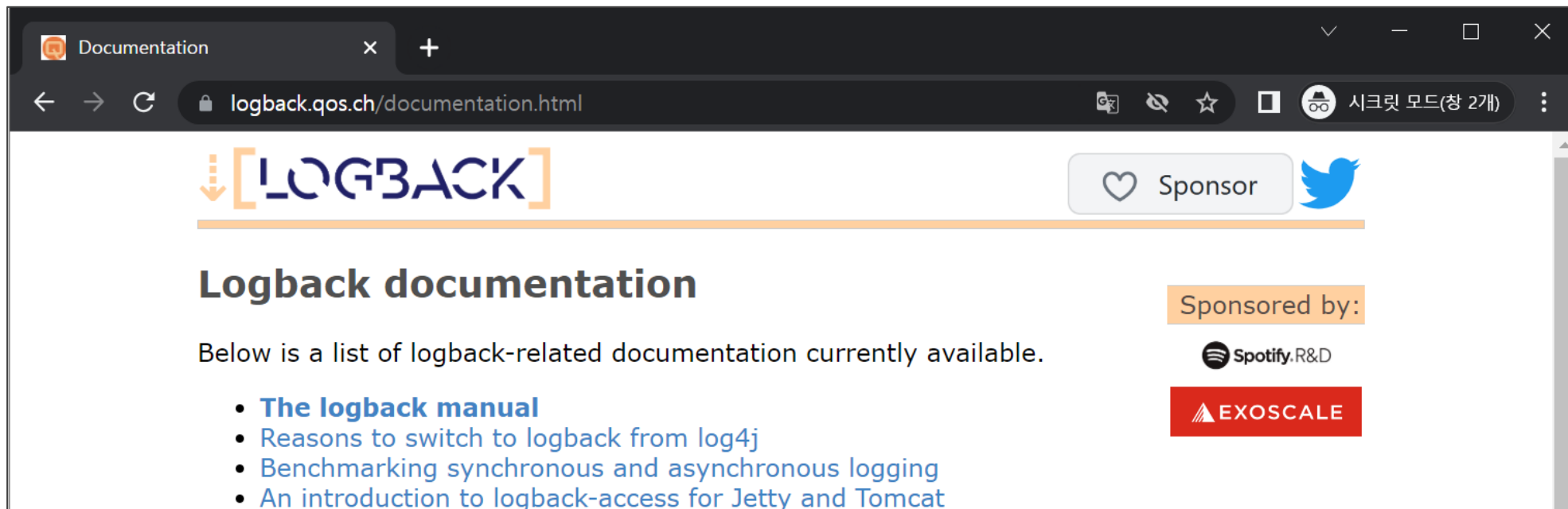
## ■ 로그 레벨의 종류

구분	의미
FATAL	응용프로그램이 중단될 가능성이 있는 매우 심각한 오류 이벤트를 지정
ERROR	응용프로그램 실행을 계속 허용할 수 있는 오류 이벤트를 지정
WARN	잠재적으로 유해한 상황을 지정
INFO	대략적인 수준에서 응용프로그램의 진행률을 강조하는 정보 메시지를 지정
DEBUG	응용프로그램을 디버깅하는데 가장 유용한 세부 정보 이벤트를 지정 (개발 단계에서 주로 사용하는 레벨, 운영 단계에서는 로그가 너무 많아 감당이 안 될 수 있음)
TRACE	DEBUG보다 세분화된 정보 이벤트를 지정

# Logback

## ■ Logback

- Log4j의 후속 버전 Java Logging Framework
- Log4j보다 10배의 성능 개선이 이루어지고 메모리 점유도 줄어듦
- 현재 가장 널리 사용되고 있음
- 6단계의 로그 레벨을 가짐  
(OFF > ERROR > WARN > INFO > DEBUG > TRACE)





# Commons Logging

## ■ Commons Logging

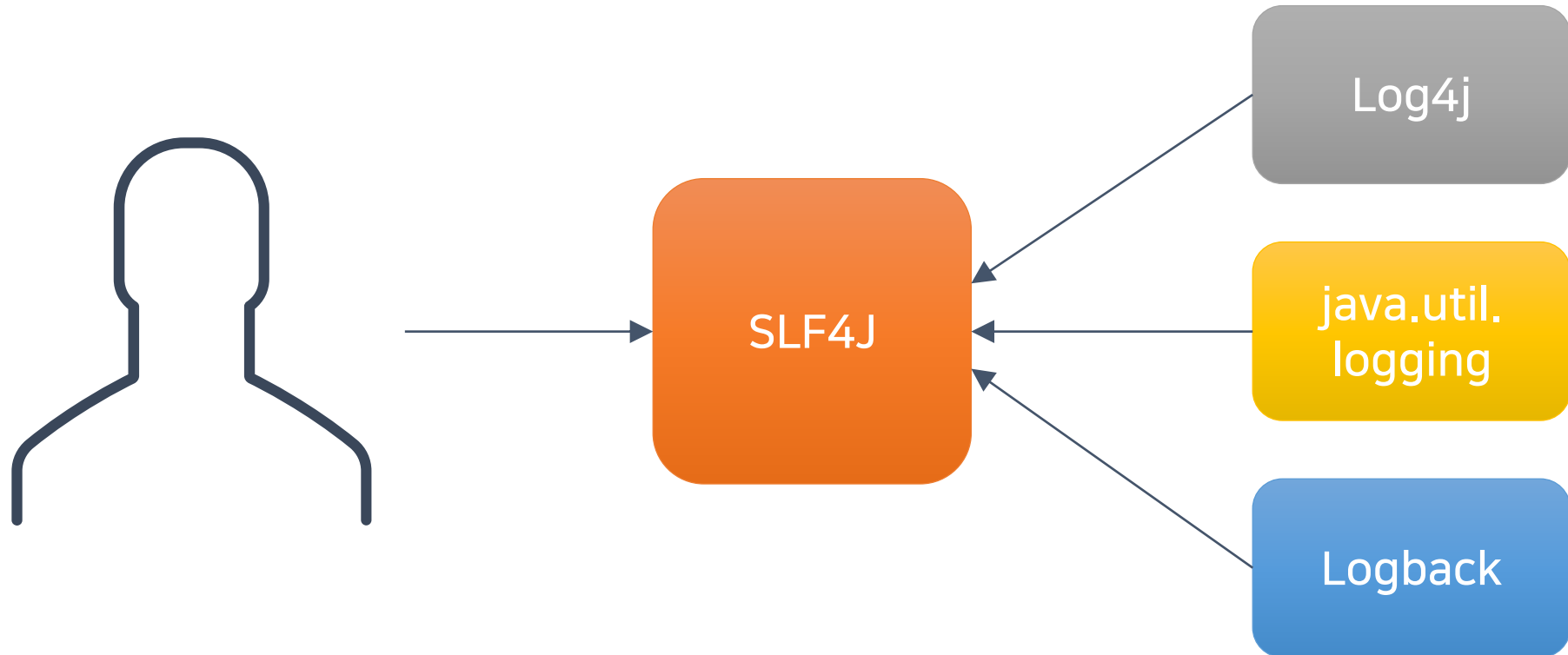
- 정식 명칭은 Jakarta Commons Logging(JCL)
- 아파치 재단에서 제공하는 공통 로깅 API로 facade 역할을 수행함  
(\* facade : 퍼사드, 어떤 소프트웨어의 간략한 인터페이스를 제공하는 디자인 패턴)
- 런타임 시점에 다른 Logging Framework를 찾기 때문에 비효율적임

## ■ SLF4J

- Simple Logging Facade for Java
- Commons Logging과 마찬가지로 공통 로깅 API 역할을 수행
- Log4j, Logback 등과 같이 다른 Logging Framework에 대한 인터페이스 역할을 수행
- 컴파일 시점에 다른 Logging Framework를 바인딩하기 때문에 효율적임(기존의 Commons Logging의 단점을 보완)
- 가장 많이 사용하는 로그 인터페이스임

## ■ SLF4J 동작 이해

- 개발자는 SLF4J 인터페이스에 맞춰 로그코드 작업
- SLF4J는 실제로 동작할 Logging Framework에 연결
- Logging Framework를 수정하더라도 로그코드에 변경이 없음



## ■ SLF4J 활용과 브릿지



SLF4J 인터페이스를 이용해 Log4j를 사용하는 예시

```

each: function(e, t, n) {
  var r, i = 0,
      o = e.length,
      a = M(e);
  if (n) {
    if (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
    } else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break
  } else if (a) {
    for (; o > i; i++)
      if (r = t.call(e[i], e[i]), r === !1) break
  } else
    for (i in e)
      if (r = t.call(e[i], e[i]), r === !1) break;
  return e
},
trim: b && !b.call("\uffff\u00a0") ? function(e) {
  return null == e ? "" : b.call(e)
} : function(e) {
  return null == e ? "" : (e + "").replace(C, "")
},
makeArray: function(e, t) {
  var n = t || [];
  return null != e && (M(Object(e)) ? x.merge(n, "string"
),
isArray: function(e, t, n) {
  var r;
  if (t) {
    if (n) return m.call(t, e, n);
    for (r = t.length, r = r ? 0 > n ? Math.max(0, r + n) : n; n in t && t[n] === e) return n
  }
}

```

## 2. 스프링 로그 프레임워크 활용

# Logback 활용 디펜던시

## ■ LOGBACK


```
<dependency>  
  <groupId>org.slf4j</groupId>  
  <artifactId>slf4j-api</artifactId>  
  <version>${org.slf4j-version}</version>  
</dependency>
```

## ■ SLF4J


```
<dependency>  
  <groupId>ch.qos.logback</groupId>  
  <artifactId>logback-classic</artifactId>  
  <version>1.4.14</version>  
</dependency>
```

# Logback 설정

## ■ src/main/resources

▼  src/main/resources

 META-INF

 logback.xml

src/main/resources/logback.xml 추가

## ■ logback.xml 예시

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>
        %d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
      </pattern>
    </encoder>
  </appender>

  <logger name="com.spring.app" level="INFO" />

  <root level="OFF">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

# logback.xml

## ■ logback.xml 구성

- appender
  - 로그 메시지를 출력할 대상(콘솔, 파일, 원격 소켓 서버, 메일, DB 등)
- encoder
  - 로그 출력 형식(Pattern)을 지정
- logger
  - 특정 패키지의 로그 처리, 로그 메시지를 appender에게 전달하는 로그의 주체
- root
  - 전체 로그 처리, logger로 선택한 log 이외의 정보를 처리



# Appender

## ■ Appender 종류

종류	기능
ch.qos.logback.core.ConsoleAppender	콘솔에 로그 메시지 출력
ch.qos.logback.core.FileAppender	하나의 파일에 로그 메시지 출력
ch.qos.logback.core.rolling.RollingFileAppender	파일을 바꿔가며 로그 메시지 출력
ch.qos.logback.classic.net.SocketAppender	원격 서버에 로그 메시지 출력
ch.qos.logback.classic.net.SMTPAppender	로그 메시지를 이메일로 전송
ch.qos.logback.classic.db.DBAppender	DB에 로그 메시지를 출력