### 파서

#### 개요

파서는 입력 데이터를 레코드 단위로 식별하고, 식별된 레코드를 필드 단위로 분할하는데 사용됩니다.



파서는 다음과 같은 특성을 갖습니다.

**원본 레코드의 보존**

파서는 원본 레코드를 보존하도록 구성되어 있습니다. 원본을 가공하는 것이 아니라, 원본에서 필드를 식별하고 추출해 별도의 필드를 생성합니다. 로그프레소 쿼리문에서 원본 레코드는 보통 **line** 필드에 출력됩니다.

**로그 형식에 맞는 전용 파서**

동일한 유형의 시스템이라 하더라도 제조사에 따라 로그는 각각 다른 형식으로 작성되어 있습니다. 예를 들어, 팔로알토 네트웍스 방화벽 로그는 CSV, 포티넷 포티게이트 방화벽 로그는 공백문자를 구분자로 하는 key=value 형식을 갖습니다. 로그프레소 소나는 각 로그 형식에 따라 각각 다른 파서를 이용해 원본 로그에서 필드 값을 추출합니다.

다음은 팔로알토 네트웍스 방화벽에서 가져온 원본 레코드의 예시입니다.

Apr 10 04:38:54 1,2012/04/10 04:38:54,012345678911,THREAT,spyware,1,2012/04/10 04:38:49,192.0.2.255,192.0.2.2,0.0.0.0,0.0.0.0,rule1,,crusher,web-browsing,vsys1,untrust,trust,ethernet1/2,ethernet1/1,forwardAll,2012/04/10 04:38:54,25466,1,80,59127,0,0,0x200000,tcp,drop-all-packets,"habl.bin",Trojan-Spy.Win32.Zbot.wti(12620),any,medium,server-to-client,0,0x0,United States,192.0.2.0-192.0.2.255,0,

위에 있는 로그가 파싱된 결과를 확인해 보려면 로그프레소 스토어에서 [팔로알토 네트웍스 방화벽](https://logpresso.store/ko/apps/paloalto-ngfw) 앱을 받아서 설치한 다음, **분석 > 쿼리**에서 다음 쿼리문을 실행해보세요.

json "{}" | eval line="Apr 10 04:38:54 1,2012/04/10 04:38:54,012345678911,THREAT,spyware,1,2012/04/10 04:38:49,192.0.2.255,192.0.2.2,0.0.0.0,0.0.0.0,rule1,,crusher,web-browsing,vsys1,untrust,trust,ethernet1/2,ethernet1/1,forwardAll,2012/04/10 04:38:54,25466,1,80,59127,0,0,0x200000,tcp,drop-all-packets,\"habl.bin\",Trojan-Spy.Win32.Zbot.wti(12620),any,medium,server-to-client,0,0x0,United States,192.0.2.0-192.0.2.255,0," | parse overlay=t paloalto-ngfw

다음은 포티넷 포티게이트 방화벽에서 가져온 원본 레코드의 예시입니다.

date=2024-05-27 time=00:00:08 devname=FGT01234567890AB devid=FGT01234567890AB logid=0000000013 type=traffic subtype=forward level=notice vd=public srcip=198.51.100.150 srcport=39986 srcintf="npu0\_vlink0" dstip=1.1.1.1 dstport=53 dstintf="wan1" poluuid=aa2401fa-d101-51e8-1c53-c8700baa2214 sessionid=26168801 proto=17 action=accept policyid=9 policytype=policy dstcountry="Korea, Republic of" srccountry="Reserved" trandisp=snat transip=203.0.113.2 transport=39986 service="DNS" duration=180 sentbyte=58 rcvdbyte=138 sentpkt=1 rcvdpkt=1 appcat="unscanned" date=2024-05-27 time=00:00:16 devname=FGT01234567890AB devid=FGT01234567890AB logid=0814044032 type=utm subtype=voip eventtype=voip level=information vd=root session\_id=194 epoch=0 event\_id=162474 srcip=198.51.100.151 src\_port=5060 dstip=192.0.2.56 dst\_port=5060 proto=17 src\_int="internal" dst\_int="npu0\_vlink1" policy\_id=1 profile="default" voip\_proto=sip kind=register action=permit status=succeeded duration=0 dir=outbound call\_id="13dcef6f@192.168.0.3" from="sip:3049@voip.example.com:5060" to="sip:3049@voip.example.com:5060"

위에 있는 로그가 파싱된 결과를 확인해 보려면 로그프레소 스토어에서 [포티게이트](https://logpresso.store/ko/apps/fortigate) 앱을 받아서 설치한 다음, **분석 > 쿼리**에서 다음 쿼리문을 실행해보세요.

json "[ {'line': 'date=2024-05-27 time=00:00:08 devname=FGT01234567890AB devid=FGT01234567890AB logid=0000000013 type=traffic subtype=forward level=notice vd=public srcip=198.51.100.150 srcport=39986 srcintf="npu0\_vlink0" dstip=1.1.1.1 dstport=53 dstintf="wan1" poluuid=aa2401fa-d101-51e8-1c53-c8700baa2214 sessionid=26168801 proto=17 action=accept policyid=9 policytype=policy dstcountry="Korea, Republic of" srccountry="Reserved" trandisp=snat transip=203.0.113.2 transport=39986 service="DNS" duration=180 sentbyte=58 rcvdbyte=138 sentpkt=1 rcvdpkt=1 appcat="unscanned"'}, {'line': 'date=2024-05-27 time=00:00:16 devname=FGT01234567890AB devid=FGT01234567890AB logid=0814044032 type=utm subtype=voip eventtype=voip level=information vd=root session\_id=194 epoch=0 event\_id=162474 srcip=198.51.100.151 src\_port=5060 dstip=192.0.2.56 dst\_port=5060 proto=17 src\_int="internal" dst\_int="npu0\_vlink1" policy\_id=1 profile="default" voip\_proto=sip kind=register action=permit status=succeeded duration=0 dir=outbound call\_id="13dcef6f@192.168.0.3" from="sip:3049@voip.example.com:5060" to="sip:3049@voip.example.com:5060"'} ]" | parse fortigate-ngfw

파서는 이렇듯 서로 다른 형식의 데이터를 정규화가 가능한 형태로 변환하는 작업을 수행합니다.

앱 파서와 기본 파서

파서는 기본 파서와 앱 파서로 구분할 수 있습니다.

**앱 파서**

대부분의 파서는 [앱](https://docs.logpresso.com/ko/sonar/4.0/ui/chapter-apps)을 설치할 때 함께 제공됩니다. 앱 파서는 연동 대상 시스템으로부터 수집된 데이터에서 필드를 추출하는데 필요한 모든 설정이 되어 있으므로 별도로 설정할 옵션이 있는 경우가 드뭅니다.

앱 파서를 수정하거나 삭제하지 마세요. 앱의 동작에 문제를 일으킬 수 있습니다.

**기본 파서**

아무 앱을 설치하지 않은 상태에서 클러스터 관리자가 정의해 사용할 수 있는 파서들입니다. 로그프레소 소나를 설치한 다음 [파서 목록을 조회](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)해보면 아무 파서도 보이지 않습니다. 앱 파서 외에 다른 목적으로 데이터를 가공하는 파서가 필요할 때 클러스터 관리자가 추가하여 사용할 수 있습니다.

실행 위치

로그프레소 소나 [클러스터](https://docs.logpresso.com/ko/sonar/4.0/ui/section-cluster-nodes)의 형태에 따라 파서의 실행 위치가 다릅니다. 다음 표와 같이 파서는 데이터가 테이블에 저장되는 노드에서 실행됩니다.

|  |  |  |
| --- | --- | --- |
| 클러스터 | 구성 노드 (각 노드는 이중화 지원) | 파서 실행 노드 |
| 1 티어 | 분석 노드 | 분석 노드 |
| 2 티어 | 수집 노드 -> 분석 노드 | 수집 노드 |
| 3 티어 | 전달 노드 -> 수집 노드 -> 분석 노드 | 수집 노드 |

#### 파서 목록 조회/검색

**수집 > 파서**에서 파서 목록을 조회하거나 검색할 수 있습니다. 다음 그림은 [팔로알토 네트웍스 방화벽](https://logpresso.store/ko/apps/paloalto-ngfw), [블루맥스 NGF](https://logpresso.store/ko/apps/bluemax-ngf), [안랩 트러스가드](https://logpresso.store/ko/apps/ahnlab-trusguard) 앱을 설치했을 때 볼 수 있는 파서 목록입니다.



* **식별자**: 쿼리문에서 사용할 파서 식별자
* **이름**: 파서 식별에 사용하는 이름
* **설명**: 파서에 대한 부가적인 정보
* **수정일**: 앱 파서가 생성된 날짜, 또는 마지막으로 수정한 날짜

파서 목록에서 특정한 파서를 찾으려면 도구 모음에 있는 검색 도구를 사용하세요. 검색 도구는 **식별자**, **이름**, **설명**에서 입력한 단어가 포함된 파서를 찾아서 보여줍니다. 파서 검색 도구는 대소문자를 구분하지 않습니다.



#### 파서 추가

대부분의 운영 환경에서 앱 파서를 이용하는 것으로 충분합니다. 그러나, 데이터 원천으로부터 수집한 로그를 파싱하는 방법이 없거나, 앱이 제공하는 파서의 기능이 운영 환경에 적합하지 않은 경우, 파서를 추가해 사용하세요.

새 파서를 추가하려면,

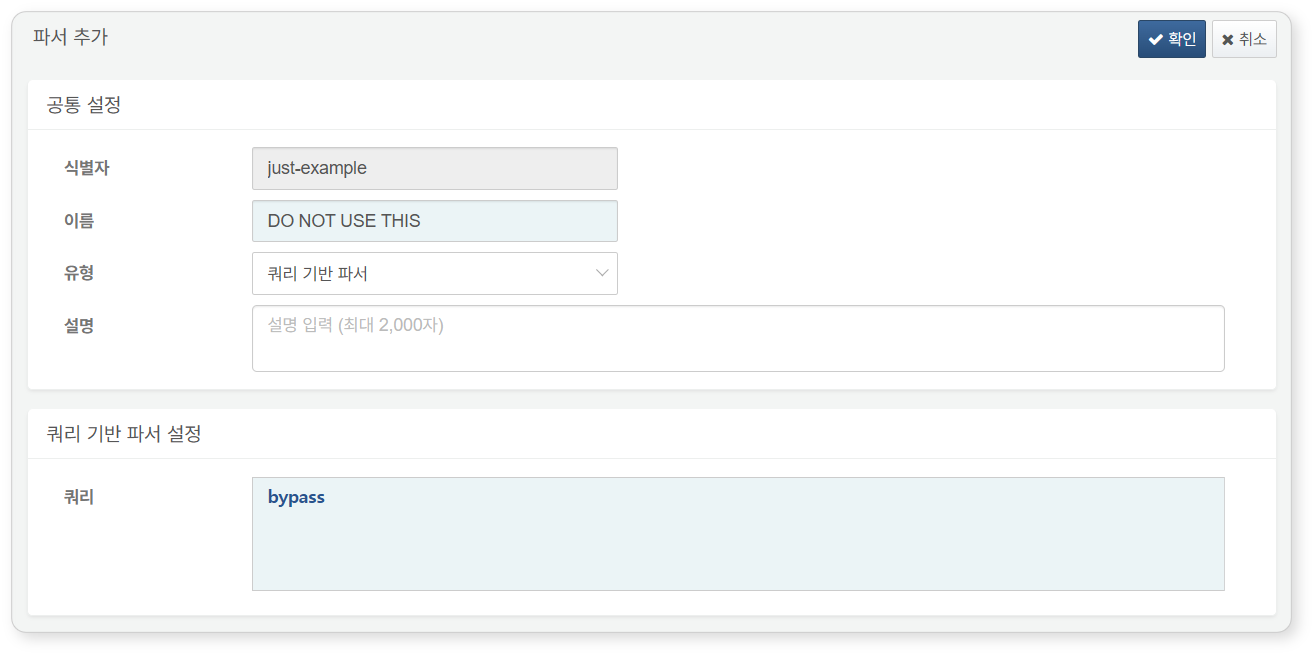
**수집 > 파서**에서 도구 모음에 있는 **추가**를 클릭하세요.



[공통 설정](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)과 [기본 파서 설정](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)을 참고하여 파서 구성에 필요한 값을 입력하거나 선택한 다음, 우측 상단에 있는 **확인**을 클릭하세요.

공통 설정

**공통 설정**에서 파서의 **식별자**, **이름**, **유형**, **설명**을 입력하거나 선택하세요. **유형**에 따라 설정할 속성이 더 있을 수 있습니다.



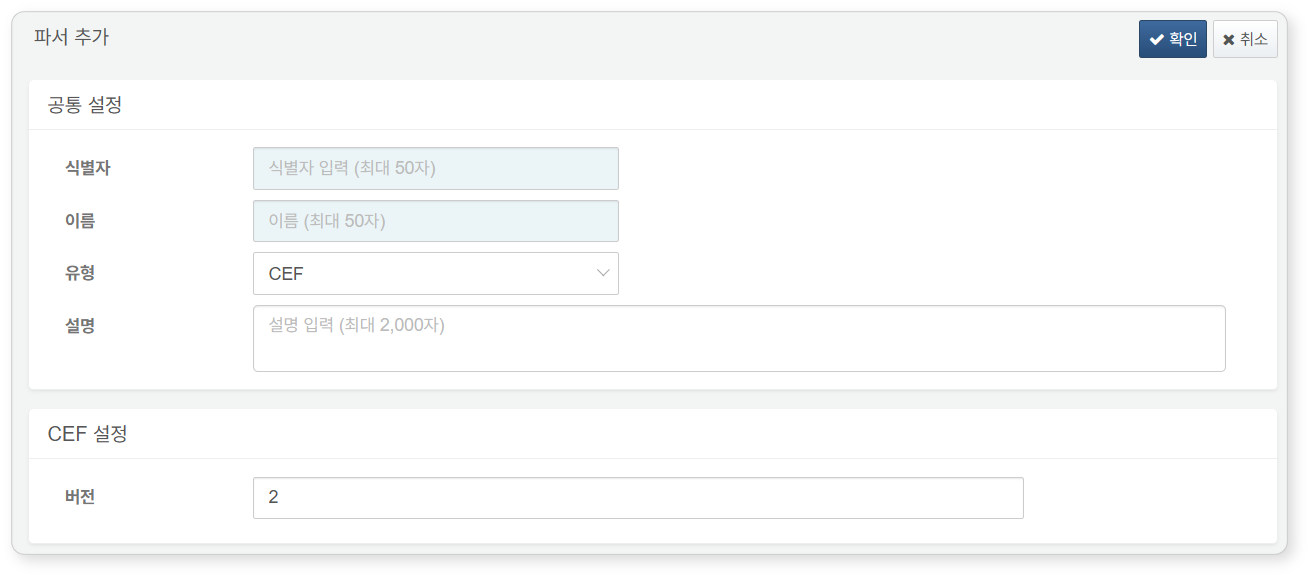
* **식별자**: 쿼리문에서 사용할 파서 식별자. 다른 파서 식별자와 겹치지 않는 유일한 이름이어야 합니다.
* **이름**: 웹 콘솔에서 사용자가 파서를 식별할 수 있도록 부여하는 이름
* **유형**: 입력 데이터의 유형, 또는 입력 데이터의 파싱에 적용할 방법(기본값: [쿼리 기반 파서](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)).
* 유형에 따라 설정해야 할 속성은 [기본 파서 설정](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)을 참고하세요. 파서 유형에 따라 입력할 속성이 없을 수도 있습니다.
* **설명**: 파서에 대한 설명. 파서에 대한 부가적인 정보를 입력할 때 사용하세요.

기본 파서 설정

여기서는 [기본 파서](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)의 유형별 속성을 설명합니다. 앱 파서는 로그프레소 스토어에서 앱 설명서를 참조하세요.

CEF

CEF(Common Event Format) 데이터를 입력으로 받아 파싱합니다. CEF는 ArcSight와 같은 ESM이나 SIEM 시스템과 연동할 때 사용하는 규격입니다.

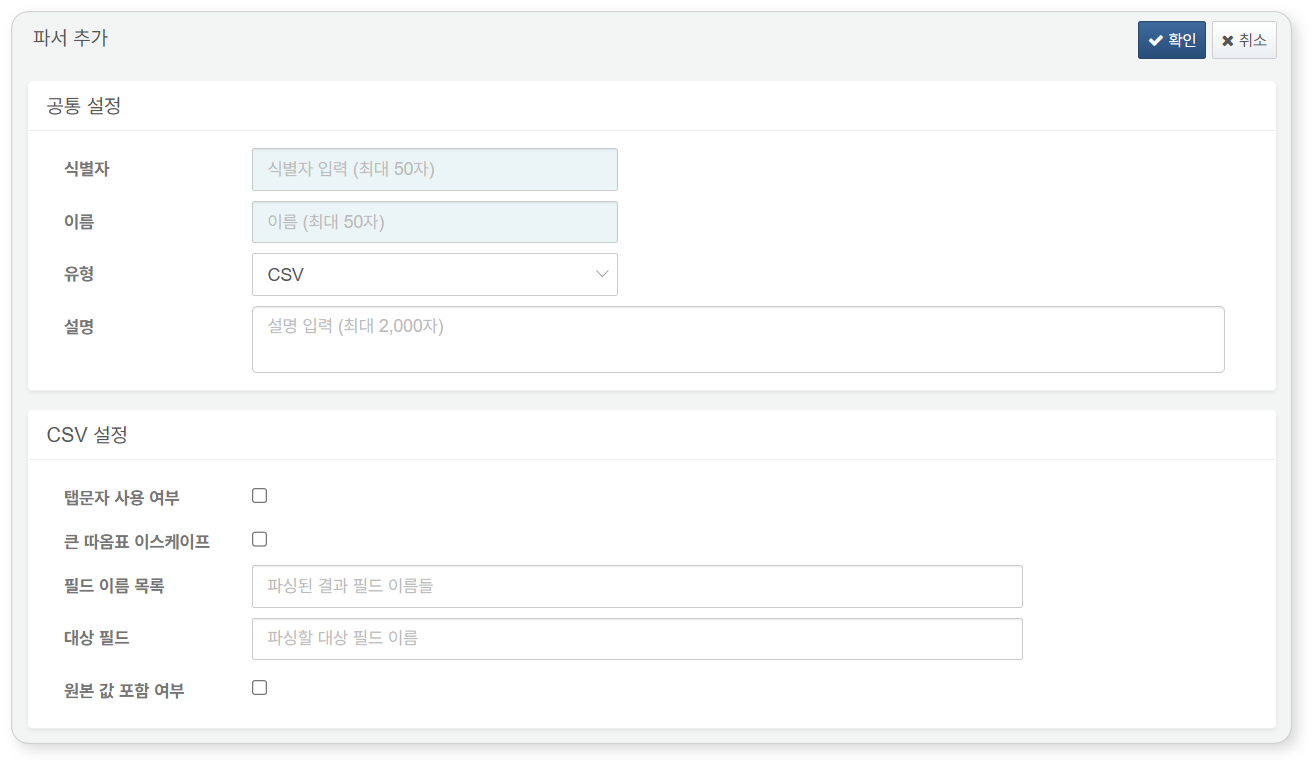


* **버전**: 파서 버전(기본값: 2). 버전 1은 하위호환성을 위한 것입니다. 새로 CEF 파서를 정의하려면 버전 2를 사용하세요.
* **1**: CEF 규격 버전 23에 따라 파싱(참조: [CEF 로그 포맷](https://ko.logpresso.com/ko/blog/2017-03-25-cef-log-format))
* **2**: CEF 규격 버전 25에 따라 파싱. 다음 표와 같이 필드 이름 정규화를 지원합니다.

|  |  |  |
| --- | --- | --- |
| 정규화 전 | 정규화 후 | 타입 |
| src | src\_ip | IP 주소 |
| spt | src\_port | 32비트 정수 |
| dst | dst\_ip | IP 주소 |
| dpt | dst\_port | 32비트 정수 |
| sourceTranslatedAddress | nat\_src\_ip | IP 주소 |
| sourceTranslatedPort | nat\_src\_port | 32비트 정수 |
| destinationTranslatedAddress | nat\_dst\_ip | IP 주소 |
| destinationTranslatedPort | nat\_dst\_port | 32비트 정수 |
| dvc | device\_ip | IP 주소 |
| dvchost | device\_host | 문자열 |
| dvcmac | device\_mac | 문자열 |
| dvcpid | device\_pid | 64비트 정수 |
| sourceDnsDomain | src\_domain | 문자열 |
| sourceServiceName | src\_service | 문자열 |
| destinationDnsDomain | dst\_domain | 문자열 |
| destinationServiceName | dst\_service | 문자열 |
| shost | src\_host | 문자열 |
| smac | src\_mac | 문자열 |
| dhost | dst\_host | 문자열 |
| dmac | dst\_mac | 문자열 |
| suid | src\_user\_id | 문자열 |
| suser | src\_user | 문자열 |
| duid | dst\_user\_id | 문자열 |
| duser | dst\_user | 문자열 |
| spid | src\_pid | 64비트 정수 |
| sproc | src\_process | 문자열 |
| dpid | dst\_pid | 64비트 정수 |
| dproc | dst\_process | 문자열 |
| fileCreateTime | file\_ctime | 날짜/시각 |
| fileModificationTime | file\_mtime | 날짜/시각 |
| fname | file\_name | 문자열 |
| filePath | file\_path | 문자열 |
| fileType | file\_type | 문자열 |
| fsize | file\_size | 64비트 정수 |
| fileHash | file\_hash | 문자열 |
| in | rcvd\_bytes | 64비트 정수 |
| out | sent\_bytes | 64비트 정수 |
| proto | protocol | 문자열 |
| app | app | 문자열 |
| act | action | 문자열 |
| cat | category | 문자열 |
| cnt | event\_count | 32비트 정수 |

CSV

CSV(Comma-Separated Values) 또는 TSV(Tab-Separated Values) 형식의 데이터를 입력으로 받습니다. 입력 데이터에서 구분자(쉼표 또는 탭 문자)로 식별되는 각 토큰을 **필드 이름 목록**과 대조하고 필드에 순서대로 할당하는 방식으로 파싱합니다.

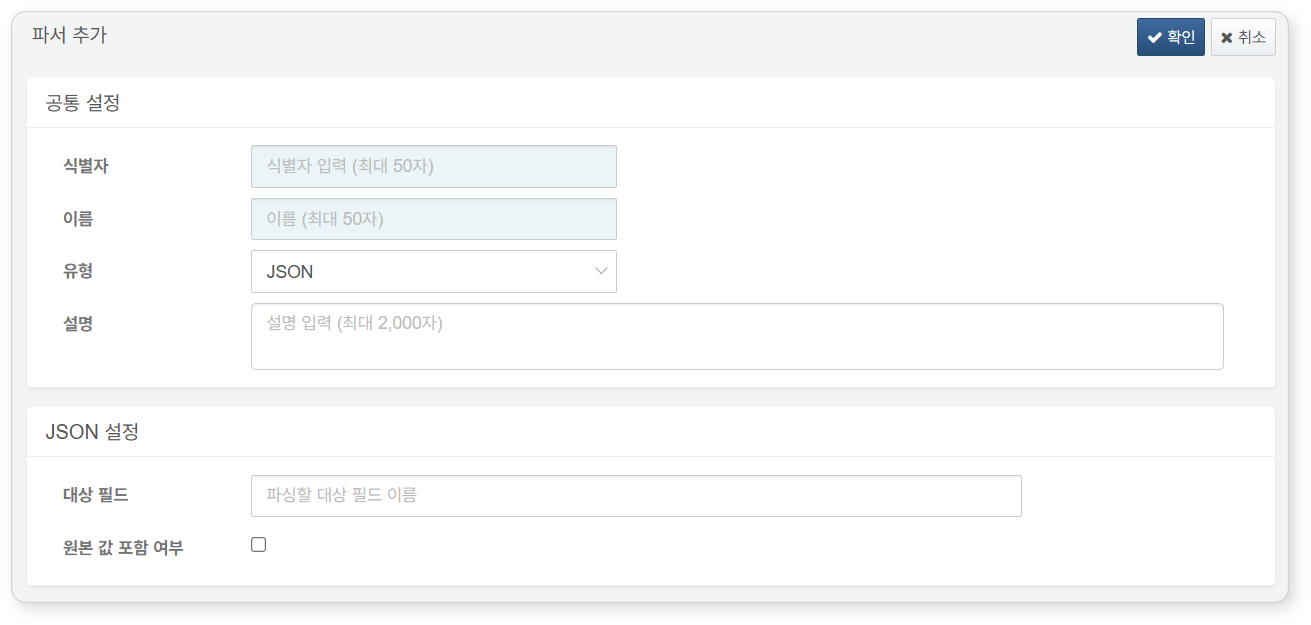


* **탭 문자 사용 여부**: 구분자로 쉼표(,) 대신 탭 문자 사용 여부(기본값: 선택 안 함)
* **큰 따옴표 이스케이프**: 특수문자를 구분하기 위한 이스케이프 문자로 큰 따옴표 사용 여부(기본값: 선택 안 함)
* **필드 이름 목록**: 파싱된 결과 필드의 이름들 쉼표(,)로 구분해서 순서대로 입력(기본값: 없음, 최대 10,000자). 미지정 시 column0, column1, ... columnN 필드에 토큰을 할당합니다.
* **대상 필드**: 파싱할 입력 필드의 이름(기본값: 없음, 최대 500자). 미지정 시 **line** 필드에 있는 데이터를 파싱합니다.
* **원본 값 포함 여부**: CSV로 파싱된 결과 외에 원본 필드 값 포함 여부(기본값: 선택 안 함). 원본 데이터를 **line** 필드에 포함해 출력합니다.

CSV 파서는 내부적으로 쉼표(,) 또는 탭 문자로 구분되는 토큰마다 각각 순서대로 column0, column1, ... columnN 필드에 할당하는 방식으로 동작합니다. 이때, column0, column1, ... columnN 필드 이름을 **필드 이름 목록**에 의해 지정된 이름으로 변경하는 과정을 거칩니다. 만약 **필드 이름 목록**으로 지정한 필드 이름이 10개이고, 토큰이 13개라면, 11번째부터 13번째 토큰은 각각 column10, column11, column12에 할당됩니다. 반대로, 토큰이 부족하면 해당 필드 값으로 null을 할당합니다.

JSON

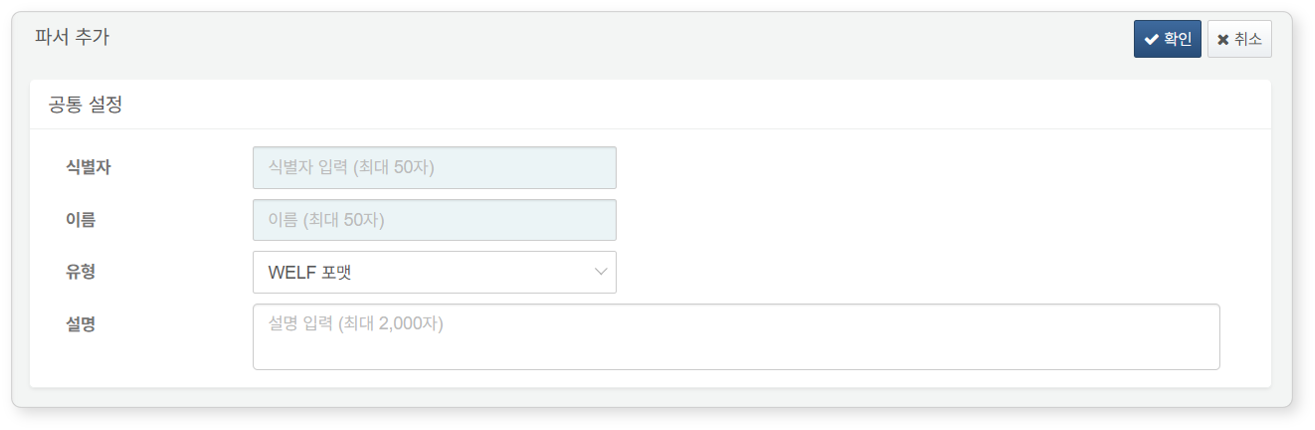
JSON 형식의 데이터를 입력으로 받아 파싱합니다. 입력 데이터가 중첩된 JSON인 경우, JSON 데이터 안에 있는 JSON 데이터는 맵 또는 배열 형태로 구성되는 복합 필드에 할당됩니다.



* **대상 필드**: 파싱할 입력 필드의 이름(기본값: 없음, 최대 255자). 미지정 시 **line** 필드에 있는 데이터를 파싱합니다.
* **원본 값 포함 여부**: JSON으로 파싱된 결과 외에 원본 필드 값 포함 여부(기본값: 선택 안 함). 원본 데이터를 **line** 필드에 포함해 출력합니다.

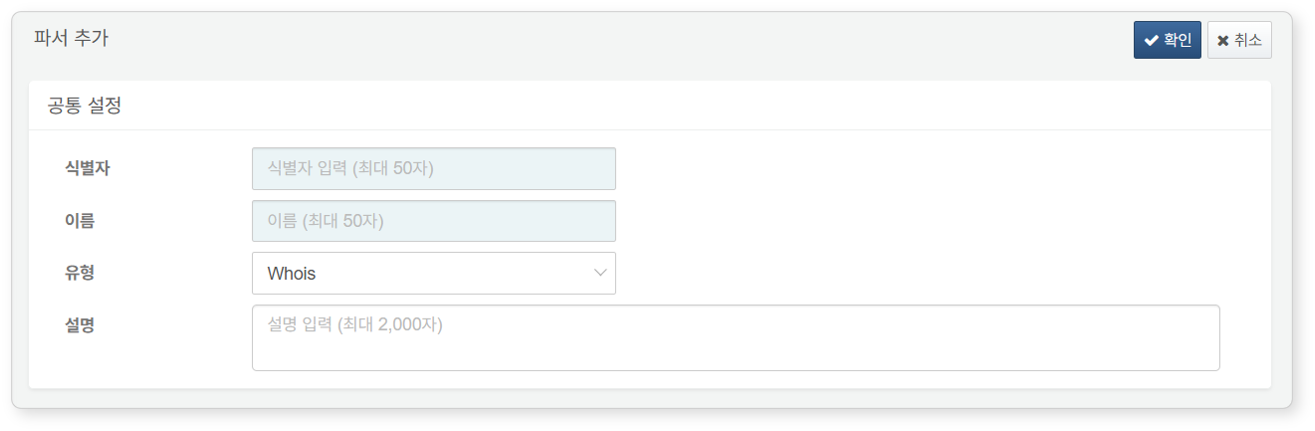
WELF

WELF(WebTrends Enhanced Log Format) 형식 데이터를 입력으로 받아 파싱합니다. WELF 파서는 지정할 속성이 없습니다.



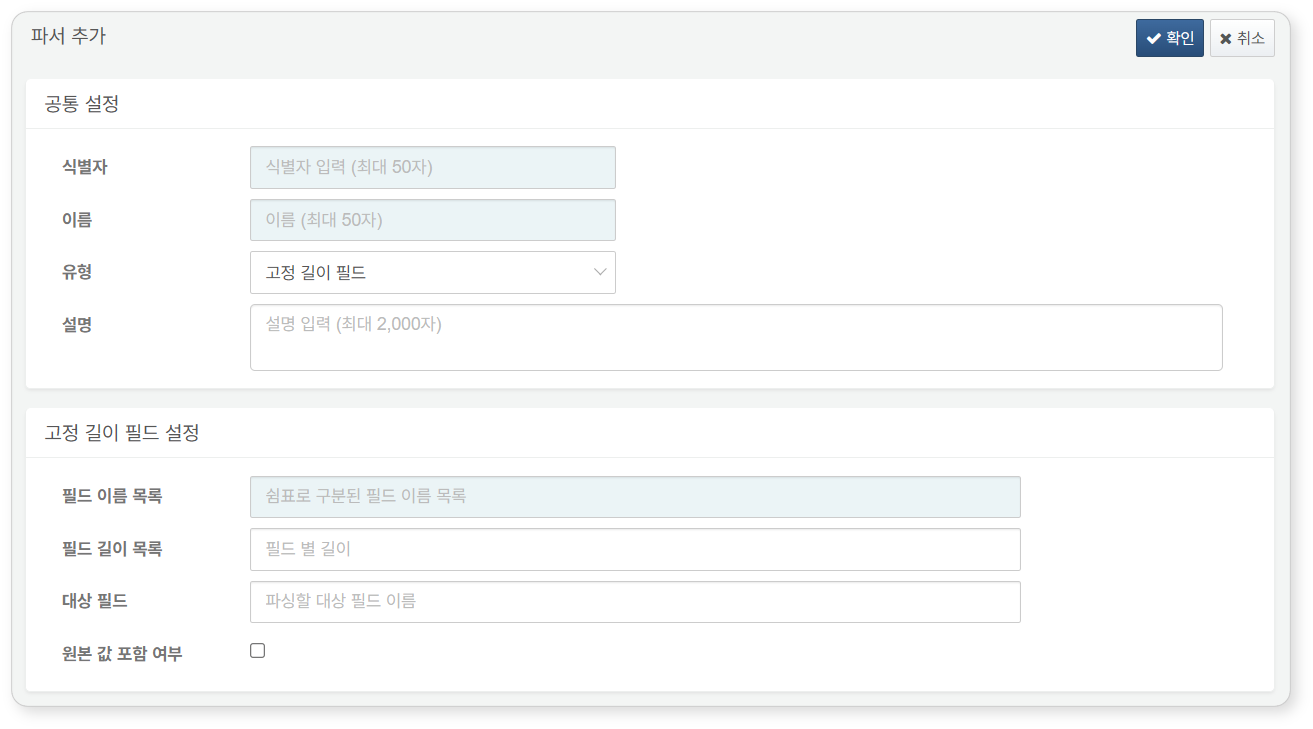
Whois

whois 명령의 출력 결과를 입력으로 받아 파싱합니다. Whois 파서는 지정할 속성이 없습니다.



고정 길이 필드

입력되는 데이터가 고정된 길이의 문자열로 구분될 때, 지정된 길이 단위로 잘라서 필드에 할당하여 파싱합니다.

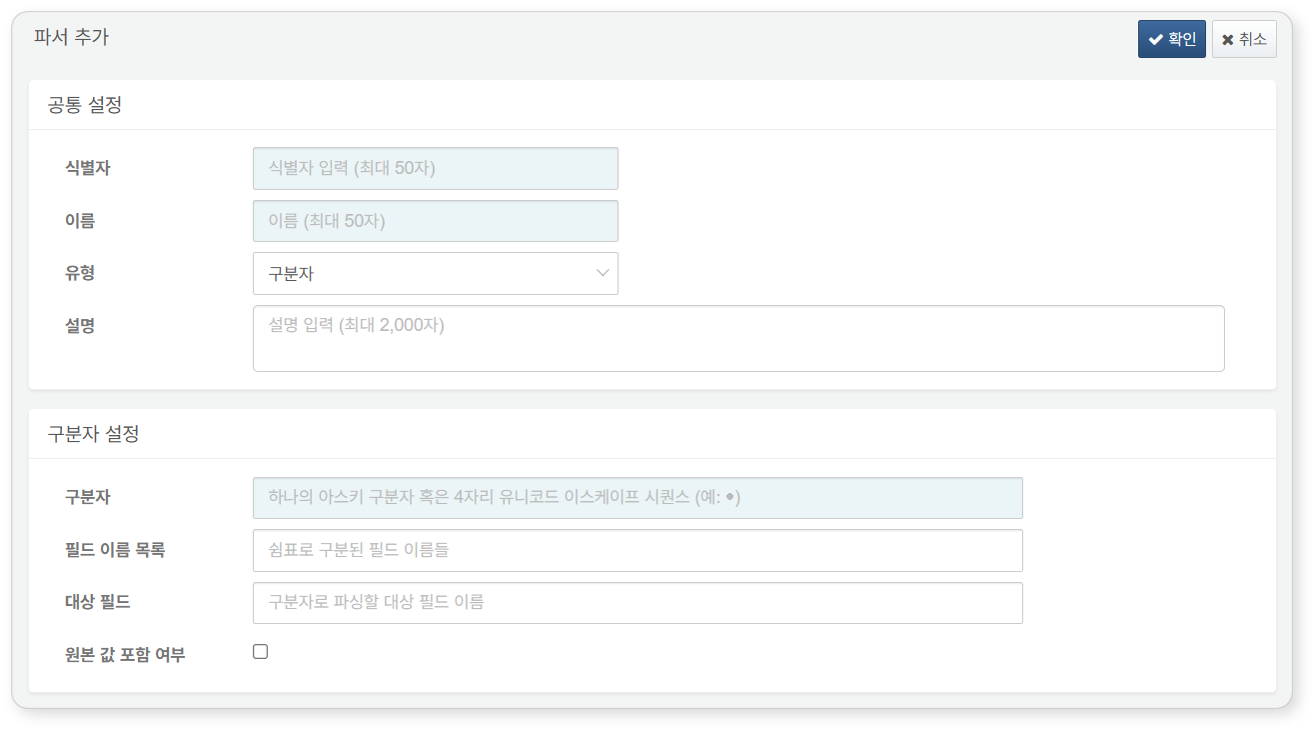


* **필드 이름 목록**: 파싱된 결과 필드의 이름들 쉼표(,)로 구분해서 순서대로 입력(기본값: 없음, 최대 10,000자)
* **필드 길이 목록**: **필드 이름 목록**에서 각 필드에 해당하는 글자수 길이를 나타내는 숫자를 쉼표(,)로 구분해서 순서대로 입력(기본값: 없음, 최대 길이:10,000자)
* **대상 필드**: 파싱할 입력 필드의 이름(기본값: 없음, 최대 255자). 미지정 시 **line** 필드에 있는 데이터를 파싱합니다.
* **원본 값 포함 여부**: 파싱된 결과 외에 원본 필드 값 포함 여부(기본값: 선택 안 함). 원본 데이터를 **line** 필드에 포함해 출력합니다.

고정 길이 필드 파서는 **필드 길이 목록**을 기준으로 데이터를 파싱합니다. 데이터가 **필드 길이 목록**보다 긴 경우, 나머지 데이터는 무시합니다. **필드 길이 목록**이 **필드 이름 목록**보다 짧은 경우, 해당 필드부터는 null을 할당합니다.

구분자

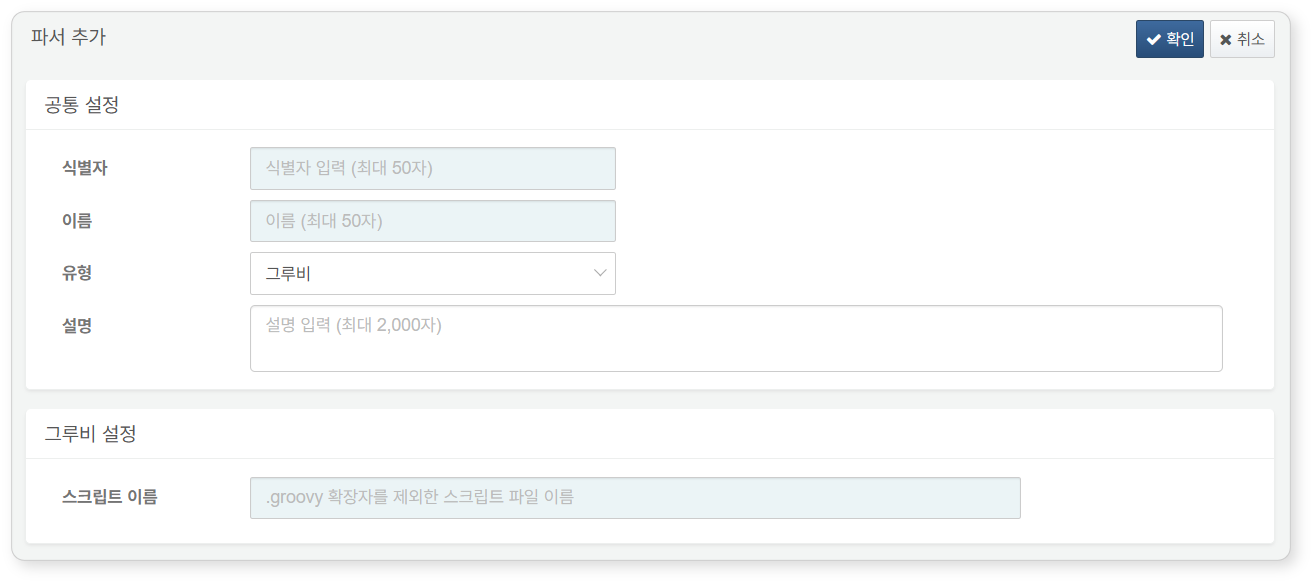
입력 데이터에서 구분자로 지정된 문자로 식별되는 각 토큰을 **필드 이름 목록**으로 지정된 필드에 순서대로 할당하여 파싱합니다.



* **구분자**: 구분자로 사용할 문자(최소 1자, 최대 10자). 4자리 유니코드 이스케이프 시퀀스(예: \u002C)를 입력할 수도 있습니다. 유니코드 이스케이프 시퀀스는 [SYMBL](https://symbl.cc/en/unicode-table/)에서 검색할 수 있습니다.
* **필드 이름 목록**: 파싱된 결과 필드의 이름들 쉼표(,)로 구분해서 순서대로 입력(기본값: 없음, 최대 10,000자). 미지정 시 column0, column1, ... columnN 필드에 토큰을 할당합니다.
* **대상 필드**: 파싱할 입력 필드의 이름(기본값: 없음, 최대 255자). 미지정 시 **line** 필드에 있는 데이터를 파싱합니다.
* **원본 값 포함 여부**: 파싱된 결과 외에 원본 필드 값 포함 여부(기본값: 선택 안 함). 원본 데이터를 **line** 필드에 포함해 출력합니다.

그루비

[그루비(Groovy)](http://www.groovy-lang.org) 언어로 작성된 스크립트를 이용해 입력 데이터를 파싱합니다.



* **스크립트 이름**: 확장자인 .groovy를 제외한 스크립트 파일의 이름(최대 255자)

그루비는 파이썬, 루비같은 언어의 영향을 받아 개발된 동적 객체 지향 언어로 JVM에서 동작합니다. 실행할 수 있는 스크립트 파일은 다음과 같은 제약 조건을 만족해야 합니다.

* 스크립트 파일 이름은 다음과 같은 형식으로 지정해야 합니다: CLASS\_NAME.groovy
* 그루비 파서는 로그프레소 설치 디렉터리의 data/araqne-logdb-groovy/parser\_scripts 디렉터리에 있는 스크립트 파일만 실행할 수 있습니다.
* 로그프레소가 제공하는 패키지를 임포트해서 사용해야 합니다. 다음과 같은 패키지를 필요에 따라 사용하십시오.
* groovy.transform.CompileStatic (성능상 이점이 있으므로 권장)
* org.araqne.logdb.groovy.GroovyQueryScript (필수)
* org.araqne.logdb.QueryStopReason
* org.araqne.logdb.Row (필수)
* org.araqne.logdb.RowBatch
* org.araqne.logdb.RowPipe

Groovy 스크립트의 성능을 향상시키려면 다음과 같은 사항을 참고하세요.

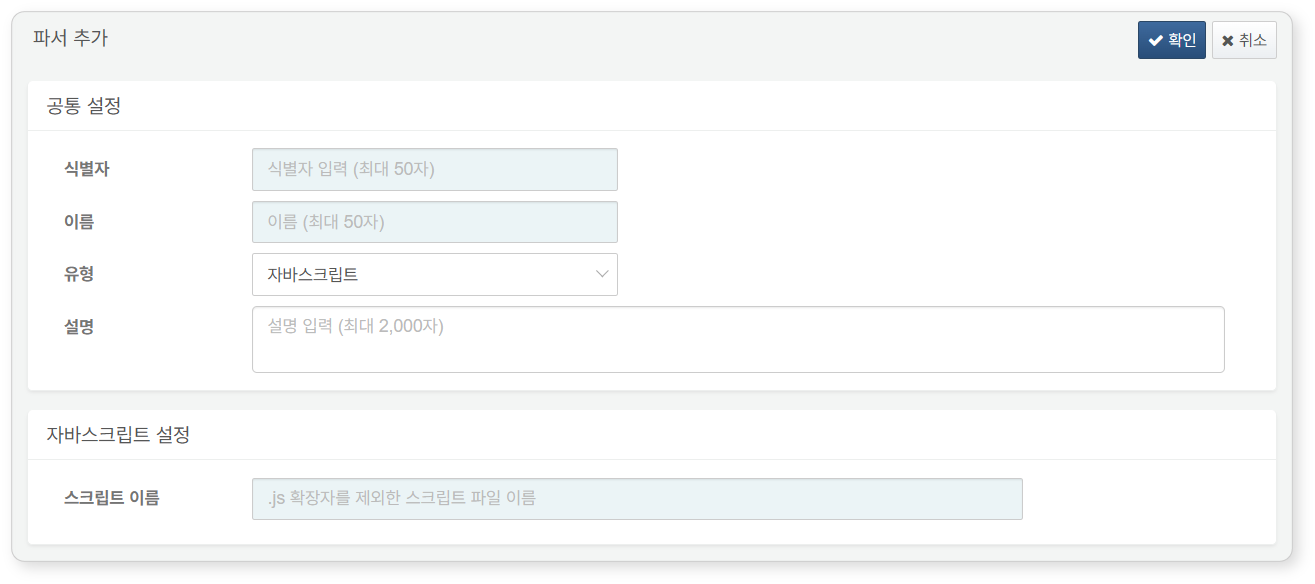
* 문자열 처리 메서드는 가능하면 사용하지 않도록 하세요. 문자열 객체가 많아지면 JVM에서 가비지 컬렉션이 빈번하게 일어납니다.
* split(), tokenize() 메서드 대신 indexOf()이나 substring()을 사용하세요. 코드는 길어지지만 더 좋은 처리 성능을 제공합니다.
* Pattern.compile()을 반복적으로 사용하지 마세요. Matcher.reset()을 호출해 Matcher 인스턴스를 재사용하는 방식이 더 좋은 성능을 제공합니다.
* 예외 발생을 최소화하세요.
* 예외가 빈번하게 발생하면 처리 성능이 현저하게 떨어집니다.
* 가능하다면, 발생할 수 있는 오류 케이스는 조건 검사를 통해 처리하세요.

예를 들어 다음과 같은 스크립트를 ToAscii.groovy라는 이름으로 data/araqne-logdb-groovy/parser\_scripts 디렉터리에 저장한 후 호출해 사용할 수 있습니다. 이 스크립트는 PCAP 파일에서 디코딩되어 **payload** 필드에 할당된 바이너리 값을 입력으로 받아 32번째 문자부터 127번째 문자를 ASCII 형식으로 인코딩해 보여줍니다.

import groovy.transform.CompileStatic;import org.araqne.logdb.Row;import org.araqne.logdb.groovy.GroovyQueryScript;@CompileStatic(groovy.transform.TypeCheckingMode.SKIP)class ToAscii extends GroovyQueryScript {def void onRow(Row row) { byte[] payload = row.get('payload') char[] chars = new char[payload.length]; for (int i = 0; i < payload.length; i++) { char c = (char) payload[i] if (c < 32 || c > 126) c = '.' chars[i] = c } row.put('text', new String(chars)) pipe.onRow(row) }}

자바스크립트

자바스크립트 언어로 작성된 스크립트를 이용해 입력 데이터를 파싱합니다.



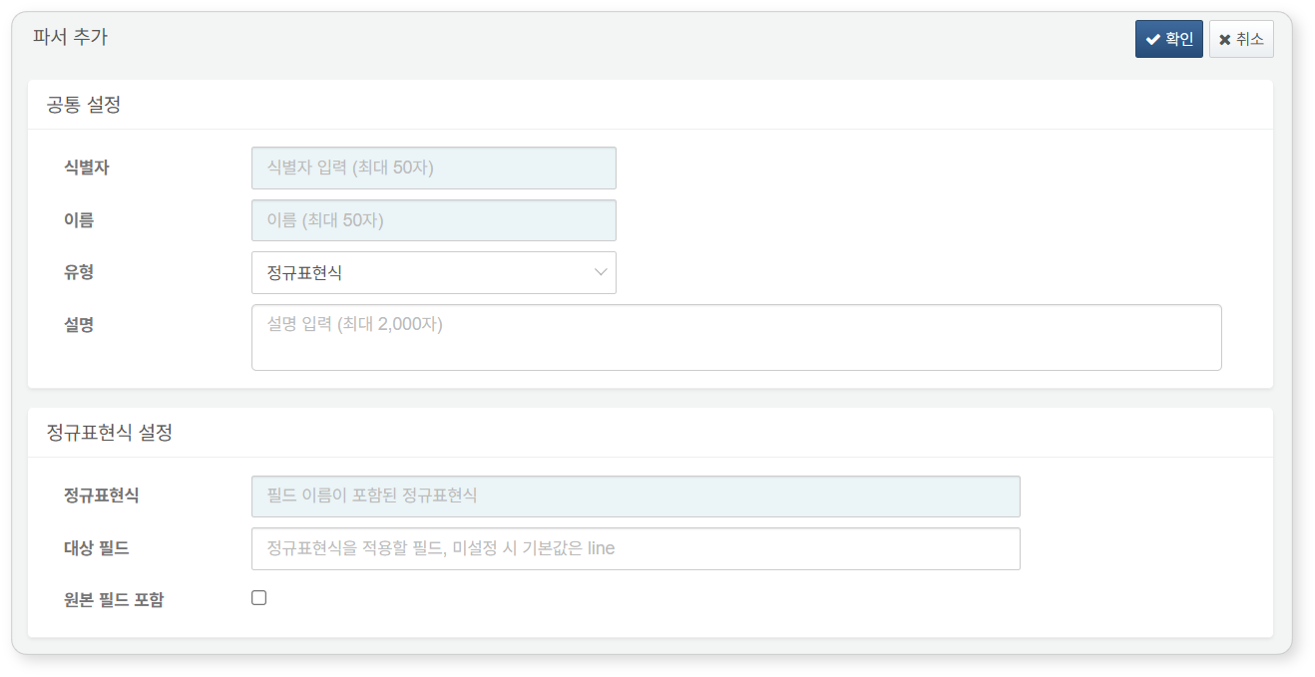
* **스크립트 이름**: 확장자인 .js를 제외한 스크립트 파일 이름(최대 255자). 실행할 스크립트 파일은 시스템 스위치 araqne.log.data를 통해 정의된 디렉터리의 하위 디렉터리인 araqne-logdb-nashorn/parser\_scripts 안에 있어야 합니다.

예를 들어 다음과 같은 스크립트를 EaiTrParser.js라는 이름으로 data/araqne-logdb-nashorn/parser\_scripts 디렉터리에 저장한 후 호출해 사용할 수 있습니다. 이 스크립트는 주로 EUC-KR 인코딩으로 작성된 로그 파일을 파싱하는데 사용됩니다. 각 필드는 고정 길이를 가지며, 이를 이용해 각 필드에 값을 할당합니다. 만약 오류가 발생하면, 오류 정보를 error 필드에 할당합니다.

var stdout = java.lang.System.out;var jstring = Java.type("java.lang.String");function EaiTrParser() {}EaiTrParser.prototype.parse = function (m) { var b = m.body.getBytes("euc-kr"); try { m.a000\_if\_id = new jstring(b, 0, 16, "euc-kr").trim(); m.a001\_account = new jstring(b, 16, 11, "euc-kr").trim(); m.a002\_trd\_dt = new jstring(b, 27, 8, "euc-kr").trim(); m.a003\_trd\_sno = new jstring(b, 35, 10, "euc-kr").trim(); m.a004\_trd\_tp\_cd = new jstring(b, 45, 2, "euc-kr").trim(); m.a005\_act\_trd\_cfc\_cd = new jstring(b, 47, 2, "euc-kr").trim(); m.a006\_act\_trd\_dtl\_cd = new jstring(b, 49, 2, "euc-kr").trim(); m.a007\_act\_pdt\_llf\_cd = new jstring(b, 51, 2, "euc-kr").trim(); m.a008\_dca\_tp\_cd = new jstring(b, 53, 2, "euc-kr").trim(); m.a009\_sps\_cd = new jstring(b, 55, 5, "euc-kr").trim(); m.a010\_biz\_pcs\_no = new jstring(b, 60, 7, "euc-kr").trim(); m.a011\_act\_trd\_dit\_cd = new jstring(b, 67, 2, "euc-kr").trim(); m.a012\_can\_yn = new jstring(b, 69, 1, "euc-kr").trim(); m.a013\_org\_trd\_sno = new jstring(b, 70, 10, "euc-kr").trim(); m.a014\_rgd\_trd\_tp\_cd = new jstring(b, 80, 2, "euc-kr").trim(); m.a015\_dca\_rgd\_no = new jstring(b, 82, 7, "euc-kr").trim(); m.a016\_sct\_rgd\_no = new jstring(b, 89, 7, "euc-kr").trim(); m.a017\_iem\_cd = new jstring(b, 96, 12, "euc-kr").trim(); m.a018\_byn\_dt = new jstring(b, 108, 8, "euc-kr").trim(); m.a019\_syn\_ttn\_dit\_cd = new jstring(b, 116, 1, "euc-kr").trim(); m.a020\_trd\_qty = new jstring(b, 117, 18, "euc-kr").trim(); m.a021\_trd\_amt = new jstring(b, 135, 15, "euc-kr").trim(); m.a021\_trd\_amt\_dot = new jstring(b, 150, 3, "euc-kr").trim(); m.a022\_trd\_uit\_pr = new jstring(b, 153, 12, "euc-kr").trim(); m.a022\_trd\_uit\_pr\_dot = new jstring(b, 165, 3, "euc-kr").trim(); m.a023\_chk\_trd\_amt = new jstring(b, 168, 18, "euc-kr").trim(); m.a024\_ect\_chk\_trd\_amt = new jstring(b, 186, 18, "euc-kr").trim(); m.a025\_trd\_pna = new jstring(b, 204, 18, "euc-kr").trim(); m.a026\_int\_amt = new jstring(b, 222, 18, "euc-kr").trim(); m.a027\_trd\_dda = new jstring(b, 240, 18, "euc-kr").trim(); m.a028\_rit\_osm\_sel\_cst = new jstring(b, 258, 18, "euc-kr").trim(); m.a029\_trd\_orn\_fee = new jstring(b, 276, 12, "euc-kr").trim(); m.a029\_trd\_orn\_fee\_dot = new jstring(b, 288, 3, "euc-kr").trim(); m.a030\_trd\_tax = new jstring(b, 291, 18, "euc-kr").trim(); m.a031\_icm\_tax = new jstring(b, 309, 18, "euc-kr").trim(); m.a032\_rsd\_tax = new jstring(b, 327, 18, "euc-kr").trim(); m.a033\_fmfs\_tax = new jstring(b, 345, 18, "euc-kr").trim(); m.a034\_cyc\_icm\_tax = new jstring(b, 363, 18, "euc-kr").trim(); m.a035\_cyc\_rsd\_tax = new jstring(b, 381, 18, "euc-kr").trim(); m.a036\_frs\_fee = new jstring(b, 399, 18, "euc-kr").trim(); m.a037\_frs\_trd\_tax = new jstring(b, 417, 18, "euc-kr").trim(); m.a038\_frs\_icm\_tax = new jstring(b, 435, 18, "euc-kr").trim(); m.a039\_frs\_rsd\_tax = new jstring(b, 453, 18, "euc-kr").trim(); m.a040\_frs\_fmfs\_tax = new jstring(b, 471, 18, "euc-kr").trim(); m.a041\_fee\_rt = new jstring(b, 489, 6, "euc-kr").trim(); m.a041\_fee\_rt\_dot = new jstring(b, 495, 9, "euc-kr").trim(); m.a042\_ect\_txa = new jstring(b, 504, 18, "euc-kr").trim(); m.a043\_sby\_bse\_pr = new jstring(b, 522, 9, "euc-kr").trim(); m.a043\_sby\_bse\_pr\_dot = new jstring(b, 531, 6, "euc-kr").trim(); m.a044\_sas\_bse\_pr = new jstring(b, 537, 9, "euc-kr").trim(); m.a044\_sas\_bse\_pr\_dot = new jstring(b, 546, 6, "euc-kr").trim(); m.a045\_sas\_amt = new jstring(b, 552, 18, "euc-kr").trim(); m.a046\_tns\_amt = new jstring(b, 570, 18, "euc-kr").trim(); m.a047\_tns\_fee = new jstring(b, 588, 18, "euc-kr").trim(); m.a048\_trd\_bf\_dca = new jstring(b, 606, 18, "euc-kr").trim(); m.a049\_trd\_af\_dca = new jstring(b, 624, 18, "euc-kr").trim(); m.a050\_trd\_bf\_sba\_amt = new jstring(b, 642, 18, "euc-kr").trim(); m.a051\_trd\_af\_sba\_amt = new jstring(b, 660, 18, "euc-kr").trim(); m.a052\_trd\_bf\_bnc\_qty = new jstring(b, 678, 18, "euc-kr").trim(); m.a053\_trd\_af\_bnc\_qty = new jstring(b, 696, 18, "euc-kr").trim(); m.a054\_trd\_bf\_rba = new jstring(b, 714, 18, "euc-kr").trim(); m.a055\_trd\_af\_rba = new jstring(b, 732, 18, "euc-kr").trim(); m.a056\_rvb\_orn\_amt = new jstring(b, 750, 18, "euc-kr").trim(); m.a057\_rvb\_orn\_qty = new jstring(b, 768, 18, "euc-kr").trim(); m.a058\_cli\_pe\_fnm = new jstring(b, 786, 40, "euc-kr").trim(); m.a059\_abk\_uit\_pr = new jstring(b, 826, 12, "euc-kr").trim(); m.a060\_abk\_uit\_pr\_dot = new jstring(b, 838, 3, "euc-kr").trim(); m.a060\_ctc\_int\_rt = new jstring(b, 841, 3, "euc-kr").trim(); m.a061\_ctc\_int\_rt\_dot = new jstring(b, 844, 8, "euc-kr").trim(); m.a061\_opi\_ogt\_cfc\_cd = new jstring(b, 852, 4, "euc-kr").trim(); m.a062\_opi\_ogt\_cd = new jstring(b, 856, 12, "euc-kr").trim(); m.a063\_opi\_tab\_cd = new jstring(b, 868, 4, "euc-kr").trim(); m.a064\_ata\_opi\_act\_no = new jstring(b, 872, 20, "euc-kr").trim(); m.a065\_cfd\_trd\_yn = new jstring(b, 892, 1, "euc-kr").trim(); m.a066\_krx\_ntc\_trd\_cd = new jstring(b, 893, 5, "euc-kr").trim(); m.a067\_fcl\_sip\_no = new jstring(b, 898, 7, "euc-kr").trim(); m.a068\_cfd\_lon\_cd = new jstring(b, 905, 2, "euc-kr").trim(); m.a069\_lon\_dt = new jstring(b, 907, 8, "euc-kr").trim(); m.a070\_lon\_orn\_amt = new jstring(b, 915, 18, "euc-kr").trim(); m.a071\_lon\_rdp\_amt = new jstring(b, 933, 18, "euc-kr").trim(); m.a072\_lon\_int\_rt = new jstring(b, 951, 3, "euc-kr").trim(); m.a072\_lon\_int\_rt\_dt = new jstring(b, 954, 8, "euc-kr").trim(); m.a073\_xrn\_dt = new jstring(b, 962, 8, "euc-kr").trim(); m.a074\_lon\_int\_amt = new jstring(b, 970, 18, "euc-kr").trim(); m.a075\_trd\_bf\_lon\_amt = new jstring(b, 988, 18, "euc-kr").trim(); m.a076\_trd\_af\_lon\_amt = new jstring(b, 1006, 18, "euc-kr").trim(); m.a077\_trd\_bf\_lon\_qty = new jstring(b, 1024, 18, "euc-kr").trim(); m.a078\_trd\_af\_lon\_qty = new jstring(b, 1042, 18, "euc-kr").trim(); m.a079\_rdp\_dfr\_amt = new jstring(b, 1060, 18, "euc-kr").trim(); m.a080\_bf\_dd\_hhd\_chk\_use\_amt = new jstring(b, 1078, 18, "euc-kr").trim(); m.a081\_bf\_dd\_bnk\_chk\_use\_amt = new jstring(b, 1096, 18, "euc-kr").trim(); m.a082\_tdy\_hhd\_chk\_use\_amt = new jstring(b, 1114, 18, "euc-kr").trim(); m.a083\_tdy\_bnk\_chk\_use\_amt = new jstring(b, 1132, 18, "euc-kr").trim(); m.a084\_trd\_mdi\_cd = new jstring(b, 1150, 2, "euc-kr").trim(); m.a085\_rvb\_pbk\_trd\_yn = new jstring(b, 1152, 1, "euc-kr").trim(); m.a086\_sve\_trd\_yn = new jstring(b, 1153, 1, "euc-kr").trim(); m.a087\_fno\_trd\_yn = new jstring(b, 1154, 1, "euc-kr").trim(); m.a088\_ksfc\_pof\_stk\_trd\_yn = new jstring(b, 1155, 1, "euc-kr").trim(); m.a089\_fc\_trd\_yn = new jstring(b, 1156, 1, "euc-kr").trim(); m.a090\_dmt\_its\_trd\_yn = new jstring(b, 1157, 1, "euc-kr").trim(); m.a091\_ose\_its\_trd\_yn = new jstring(b, 1158, 1, "euc-kr").trim(); m.a092\_cd\_cp\_trd\_yn = new jstring(b, 1159, 1, "euc-kr").trim(); m.a093\_rp\_trd\_yn = new jstring(b, 1160, 1, "euc-kr").trim(); m.a094\_trs\_trd\_yn = new jstring(b, 1161, 1, "euc-kr").trim(); m.a095\_rcv\_trd\_yn = new jstring(b, 1162, 1, "euc-kr").trim(); m.a096\_bf\_lst\_mm\_pmt\_tal = new jstring(b, 1163, 18, "euc-kr").trim(); m.a097\_bf\_lst\_pmt\_dt = new jstring(b, 1181, 8, "euc-kr").trim(); m.a098\_rgs\_dt = new jstring(b, 1189, 8, "euc-kr").trim(); m.a099\_rgs\_tm = new jstring(b, 1197, 8, "euc-kr").trim(); m.a100\_rgs\_tab\_cd = new jstring(b, 1205, 4, "euc-kr").trim(); m.a101\_rgs\_emp\_no = new jstring(b, 1209, 6, "euc-kr").trim(); m.a102\_rgs\_trm\_no = new jstring(b, 1215, 8, "euc-kr").trim(); m.a103\_rgs\_cuc\_mdi\_cd = new jstring(b, 1223, 2, "euc-kr").trim(); m.a104\_rgs\_usr\_ip\_adr = new jstring(b, 1225, 32, "euc-kr").trim(); m.a105\_rgs\_hts\_usr\_id = new jstring(b, 1257, 8, "euc-kr").trim(); m.a106\_alt\_dt = new jstring(b, 1265, 8, "euc-kr").trim(); m.a107\_alt\_tm = new jstring(b, 1273, 8, "euc-kr").trim(); m.a108\_alt\_tab\_cd = new jstring(b, 1281, 4, "euc-kr").trim(); m.a109\_alt\_emp\_no = new jstring(b, 1285, 6, "euc-kr").trim(); m.a110\_alt\_trm\_no = new jstring(b, 1291, 8, "euc-kr").trim(); m.a111\_alt\_cuc\_mdi\_cd = new jstring(b, 1299, 2, "euc-kr").trim(); m.a112\_alt\_usr\_ip\_adr = new jstring(b, 1301, 32, "euc-kr").trim(); m.a113\_alt\_hts\_usr\_id = new jstring(b, 1333, 8, "euc-kr").trim(); m.a114\_amn\_tab\_cd = new jstring(b, 1341, 4, "euc-kr").trim(); m.a115\_amn\_mo\_tab\_cd = new jstring(b, 1345, 4, "euc-kr").trim(); m.filler = new jstring(b, 1349, 651, "euc-kr").trim(); } catch (error) { m.error = error; } return m;};

정규표현식

입력 데이터를 정규표현식을 이용하여 파싱합니다.



* **정규표현식**: 필드값 추출에 적용할 정규표현식(최대 10,000자). 정규표현식과 일치하는 문자열을 필드에 할당할 수 있도록 필드 이름을 정규표현식 안에 그룹 이름 형식으로 포함해야 합니다(예: (?<date>\d{4}-\d{2}-\d{2}). 여기서 date는 정규표현식과 일치하는 값을 할당할 필드의 이름)
* **대상 필드**: 파싱할 입력 필드의 이름(최대 255자). 지정하지 않으면 **line** 필드에 있는 데이터를 파싱합니다.
* **원본 값 포함 여부**: 파싱된 결과 외에 원본 필드 값 포함 여부(기본값: 선택 안 함). 원본 데이터를 **line** 필드에 포함해 출력합니다.

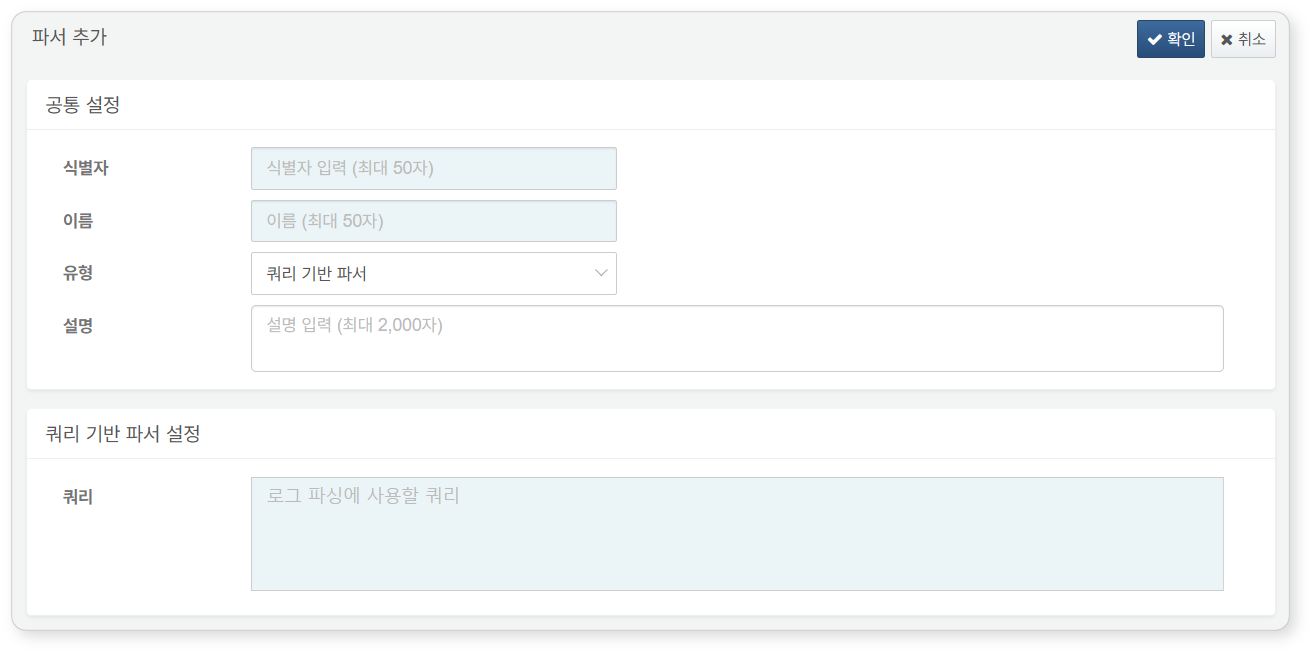
정규표현식은 매우 유연하고 강력한 문자열 검색 기능을 제공합니다. 그러나 사용하기 어려워 많은 주의가 필요합니다. 정규표현식 파서를 사용하기 전에 정규표현식에 대해 충분히 학습하고 사용하세요.

다음은 웹 서버 로그를 입력으로 받아 파싱할 때 사용할 수 있는 정규표현식 예시입니다.

이 정규표현식은 Apache, Nginx 웹 로그의 기본 형식을 가정하고 있습니다. 만약 로그 형식을 변경한 경우 정규표현식을 수정해서 사용해야 합니다.

쿼리 기반 파서

입력 데이터를 [로그프레소 쿼리](https://docs.logpresso.com/ko/sonar/4.0/ui/ko/query)를 이용해 파싱합니다. 쿼리 문법을 그대로 사용할 수 있으므로 다른 파서에 비해 상대적으로 복잡하고 정교한 파싱을 수행할 수 있습니다. 사용자 정의 파서를 생성할 때 기본값으로 지정되는 파서로, 가장 많이 사용됩니다.



* **쿼리**: 입력으로 받은 데이터를 파싱하는 로그프레소 쿼리문(최대 50,000자)

다음은 웹 서버 로그에서 유저 에이전트 정보를 입력으로 받아 분류하는 쿼리문의 예시입니다.

eval agent\_type = case( user\_agent == "\*Googlebot\*", "google", user\_agent == "\*GoogleDocs\*", "google docs", user\_agent == "\*bingbot\*" or user\_agent == "\*BingPreview\*", "bing", user\_agent == "\*facebookexternalhit\*", "facebook", user\_agent == "\*Yandex\*", "yandex", user\_agent == "\*api.slack.com\*", "slack", user\_agent == "\*CensysInspect\*", "censys", user\_agent == "\*Chrome\*", "chrome", user\_agent == "\*Safari\*", "safari", user\_agent == "\*Firefox\*", "firefox", user\_agent == "\*Python\*" or user\_agent == "\*python\*", "python", user\_agent == "\*Go-http-client\*", "golang", user\_agent == "\*MJ12bot\*", "mj12", user\_agent == "\*Expanse, a Palo Alto Networks\*", "xpanse", user\_agent == "\*SemrushBot\*", "semrush", user\_agent == "\*PetalBot\*", "petal", user\_agent == "\*AhrefsBot\*", "ahrefs", "misc" )

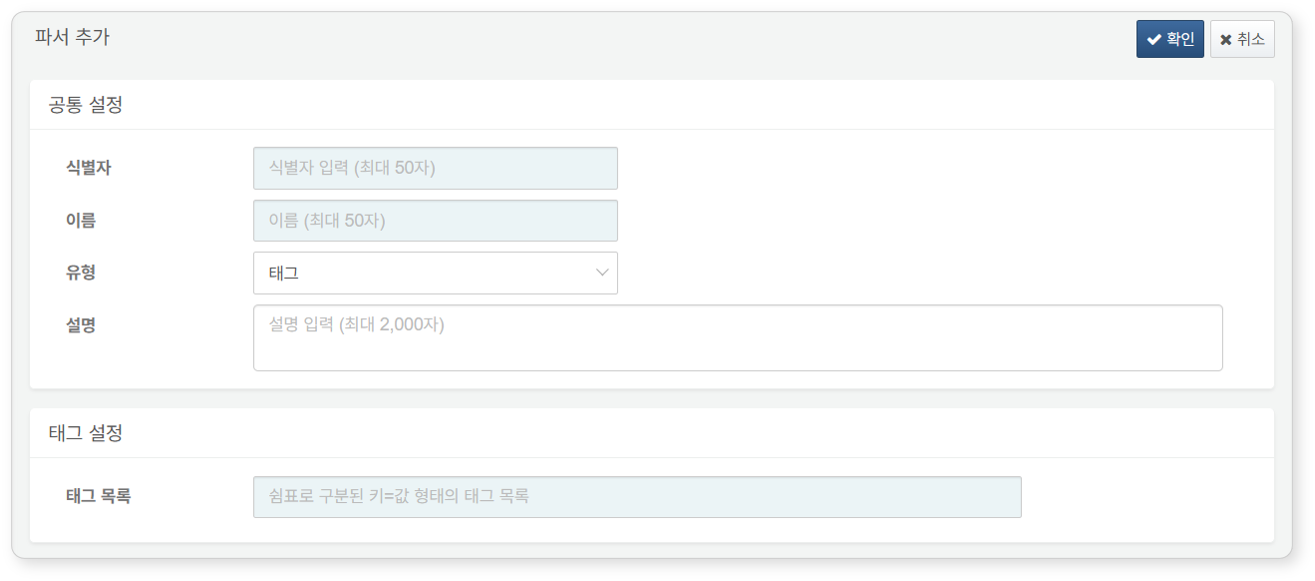
쿼리 기반 파서는 이미 입력으로 전달 받는 데이터가 있으므로 데이터를 [가공](https://docs.logpresso.com/ko/sonar/4.0/ui/ko/query/section-data-processing)하거나, 다른 데이터와 [매핑](https://docs.logpresso.com/ko/sonar/4.0/ui/ko/query/section-data-mapping), [병합](https://docs.logpresso.com/ko/sonar/4.0/ui/ko/query/section-data-merge)에 필요한 쿼리 명령으로 시작하는 경우가 많습니다.

폐기 예정

여기서 열거한 파서들은 하위호환성을 위한 것으로, 사용을 중단할 예정입니다. [쿼리 기반 파서](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)로 대체해 사용하세요.

태그

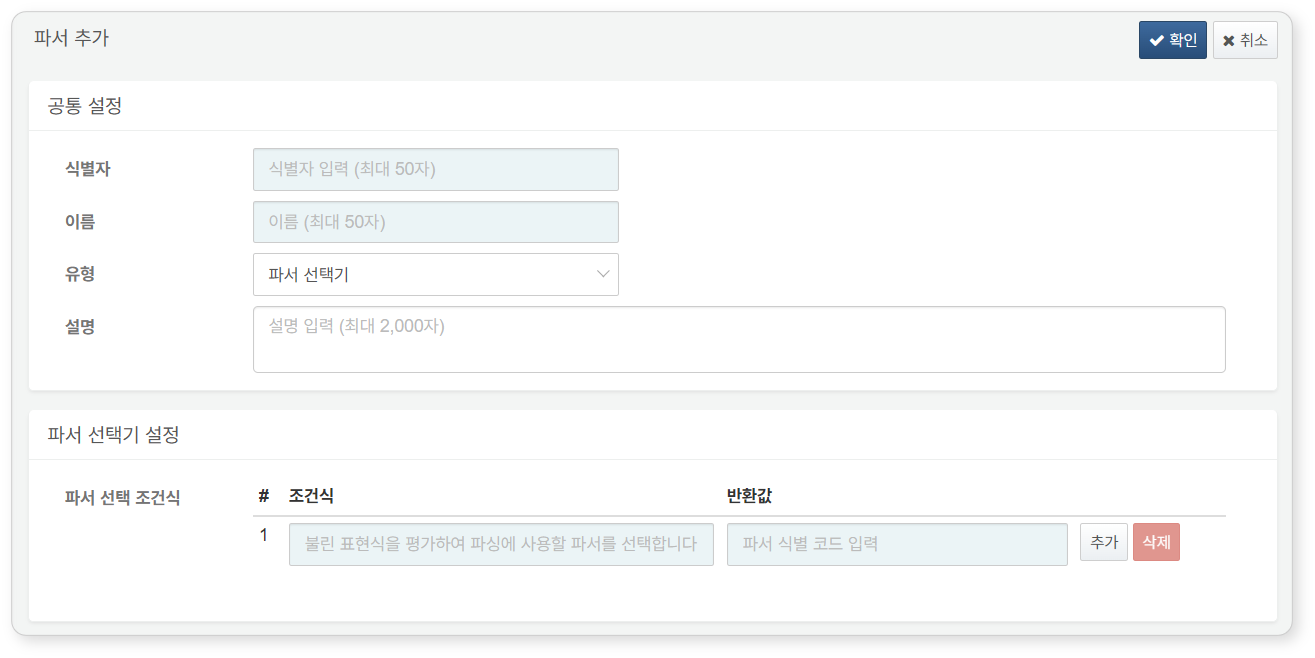
입력 데이터의 메타데이터에 키-값 형태의 태그를 추가하여 파싱합니다.



* **태그 목록**: 키=값 쌍(메타데이터 태그)을 쉼표로 구분하여 입력(최대 10,000자)

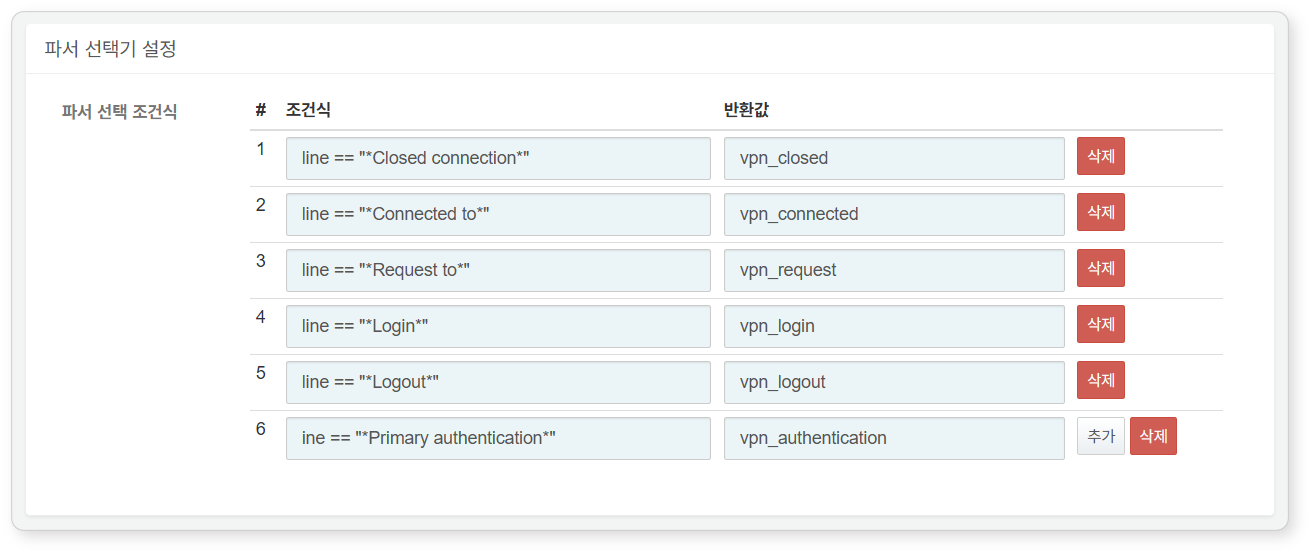
파서 선택기

표현식 조건과 일치하는 첫번째 파서를 선택하여 원본 로그를 파싱합니다.



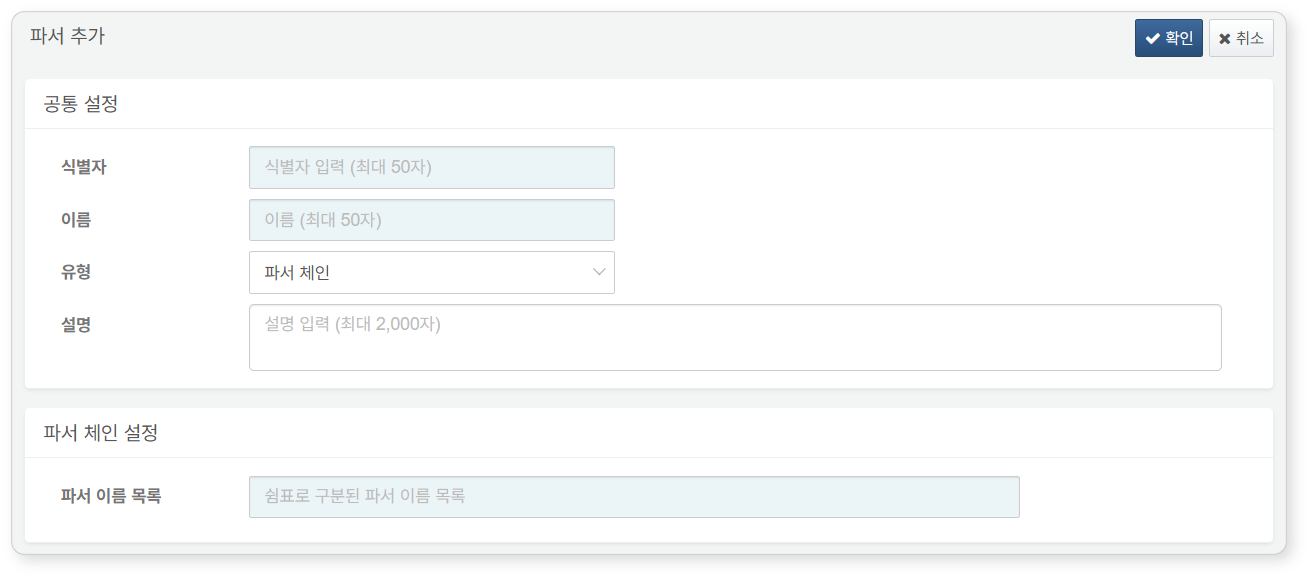
* **파서 선택 조건식**: 불리언 값을 반환하는 쿼리문으로 작성되는 **조건식**, 조건식을 평가한 값이 true일 때 적용할 파서의 식별자 정보
* **조건**: 불리언 값(true, false)으로 평가되는 쿼리 조건식(최대 1,000자). 입력 레코드가 여러 조건을 만족할 경우, 가장 첫 번째 조건에 해당하는 파서를 선택합니다.
* **반환값**: 조건을 만족할 경우 사용할 파서의 식별자(최대 50자)

파서 선택기는 여러가지 형식의 데이터가 혼합되어 있는 입력을 레코드 단위로 조건식에 따라 평가합니다. 다음 그림은 파서 선택기에 파서 선택 조건식을 입력한 예시입니다. 조건식은 서로 중복되지 않으면서 모든 가능한 경우를 포괄할 수 있게 작성하는 것이 좋습니다.



파서 체인

입력 데이터에 여러 개를 조합한 파서 체인을 적용해 파싱합니다. 파서 체인이란 입력 데이터를 파서로 처리한 다음, 또 다른 파서가 이전 파서의 출력을 입력으로 받아 처리하는 방식을 의미합니다.

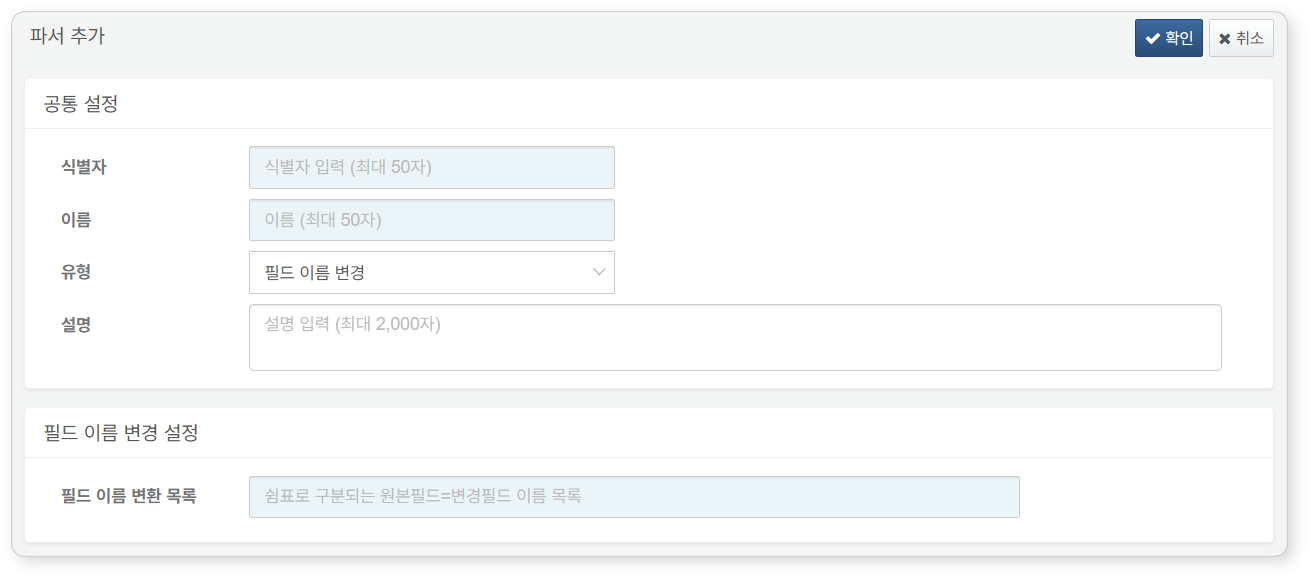


* **파서 이름 목록**: 쉼표(,)를 구분자로 하는 파서 식별자 목록(최대 1,000자). 체인의 식별자를 적용할 순서대로 입력하세요.

파서 체인이 너무 길어지면 로그 분석 성능에 영향을 줄 수 있습니다. 필요한 만큼만 파서를 연결하세요.

필드 이름 변경

입력 데이터의 특정한 필드 이름을 변경합니다.



* **필드 이름 변환 목록**: **원본필드=변경필드** 형식으로 쉼표(,)로 구분해서 입력(최대 10,000자)

#### 파서 수정

파서를 수정하려면,

목록에서 수정할 파서의 **식별자**나 **이름**을 클릭하세요.



**파서 수정** 화면에서 정보를 수정한 후 **확인**을 클릭하세요.

* 수정할 속성에 대한 설명은 [파서 추가](https://docs.logpresso.com/ko/sonar/4.0/ui/section-parsers)를 참조하세요.
* **식별자**는 수정할 수 없습니다. 식별자를 수정하려면 파서를 삭제하고 다시 생성하세요.

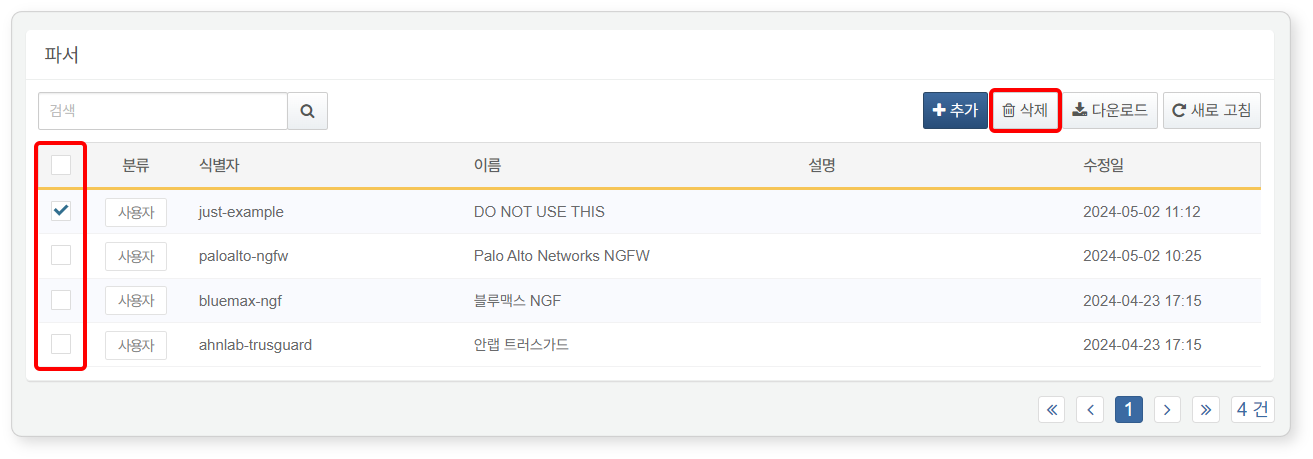
앱 파서를 수정하지 마세요. 앱의 동작에 문제를 일으킬 수 있습니다.

#### 파서 삭제

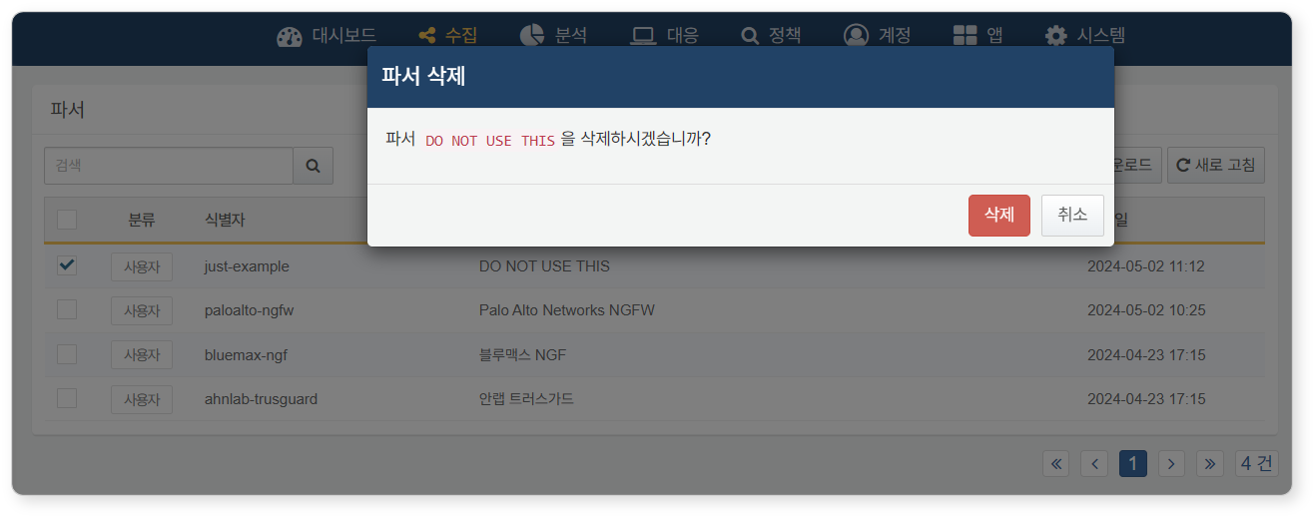
파서를 삭제하려면,

목록에서 삭제할 파서 정보가 있는 행의 체크박스를 선택하세요.

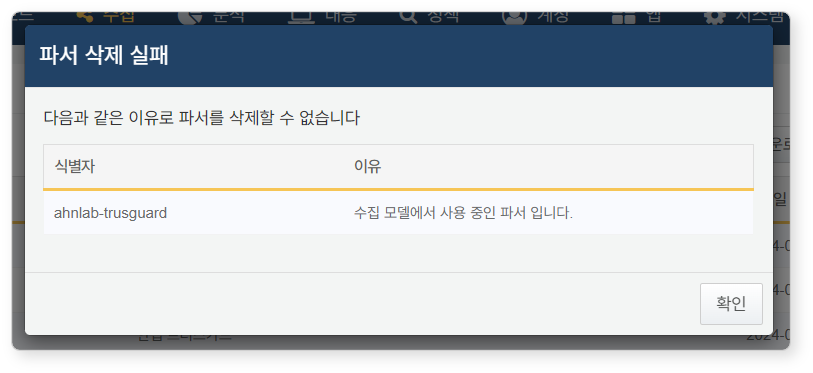
도구 모음에서 **삭제**를 클릭하세요.



**파서 삭제** 대화상자에서 삭제할 파서 목록을 확인한 후 **삭제**를 클릭하세요. 삭제하지 않으려면 **취소**를 클릭하세요.



파서를 삭제하지 못하는 경우, **파서 식제 실패** 대화상자에서 실패 원인을 확인할 수 있습니다.



앱 파서를 삭제하지 마세요. 앱의 동작에 문제를 일으킬 수 있습니다.