

세션 관리 안내서

JEUS 9

TMAXSOFT

저작권 공지

Copyright 2024. TmaxSoft Co., Ltd. All Rights Reserved.

제한된 권리

이 소프트웨어(JEUS®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

Java, Solaris는 Oracle Corporation 및 그 자회사, 관계회사의 등록 상표입니다.

Microsoft, Windows, Windows NT는 Microsoft Corporation의 등록 상표 또는 상표입니다.

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상호, 상표, 또는 등록 상표입니다.

본 사용설명서에 기재된 회사, 시스템, 제품 이름 등에 반드시 상표 표시(™, ®)를 하지는 않습니다.

오픈 소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다.: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :

\${INSTALL_PATH}/license/oss_licenses

기술 지원

홈페이지: <https://www.tmaxsoft.com>

기술 서비스 센터: 1544-8629

안내서 이력

제품 버전	안내서 버전	발행일	비고
JEUS 9	3.1.1	2024-09-27	-

목차

1. 세션 트래킹	1
1.1. 개요	1
1.2. 세션 트래킹 구조	1
1.3. 세션 트래킹 동작	3
1.3.1. 웹 엔진에서 동작	3
1.3.2. 클러스터 환경에서 동작	4
1.4. 세션 트래킹 모드	11
1.5. 컨텍스트간 세션 공유	12
1.6. 세션 트래킹 설정	12
1.6.1. 세션 설정	13
1.6.2. 세션 서버 설정	16
1.7. 세션 트래킹 튜닝	16
1.8. 세션 모니터링	17
2. 세션 서버	18
2.1. 개요	18
2.2. 기본 개념	18
2.2.1. 중앙 세션 서버	18
2.2.2. 분산 세션 서버	19
2.3. 서버 구조	19
2.3.1. 중앙 세션 서버 구조	19
2.3.2. 분산 세션 서버 구조	20
2.4. 동작 방식	22
2.4.1. 중앙 세션 서버	22
2.4.2. 분산 세션 서버	22
2.5. 주요 기능	26
2.5.1. 중복 로그인 방지 기능	26
2.5.2. Failback 기능	27
2.6. 세션 클러스터 모드	30
2.6.1. 기본 세션 클러스터 모드	30
2.6.2. 도메인 와이드 세션 클러스터 모드	30
2.6.3. 세션 스토리지 스코프 클러스터 모드	31
2.7. 세션 서버 설정	31

1. 세션 트래킹

본 장에서는 세션 트래킹에 관련된 기본적인 개념과 세션, 세션 ID, 세션 쿠키, URL Rewriting 등에 대한 정의를 설명한다. 또한 JEUS 웹 엔진과 보다 복잡한 분산 환경인 서버 클러스터링 환경에서의 세션 트래킹 구현 및 설정 방법에 대해 설명한다.

1.1. 개요

세션 트래킹(Session Tracking)은 좁은 의미로 요청된 세션을 찾아 주는 동작이다.

클러스터링 환경에서 세션 트래킹의 지원 범위(Scope)에 따라 라우팅(Routing)에 의한 세션 트래킹과 세션 서버(Session Server)를 통한 세션 트래킹으로 구분된다.

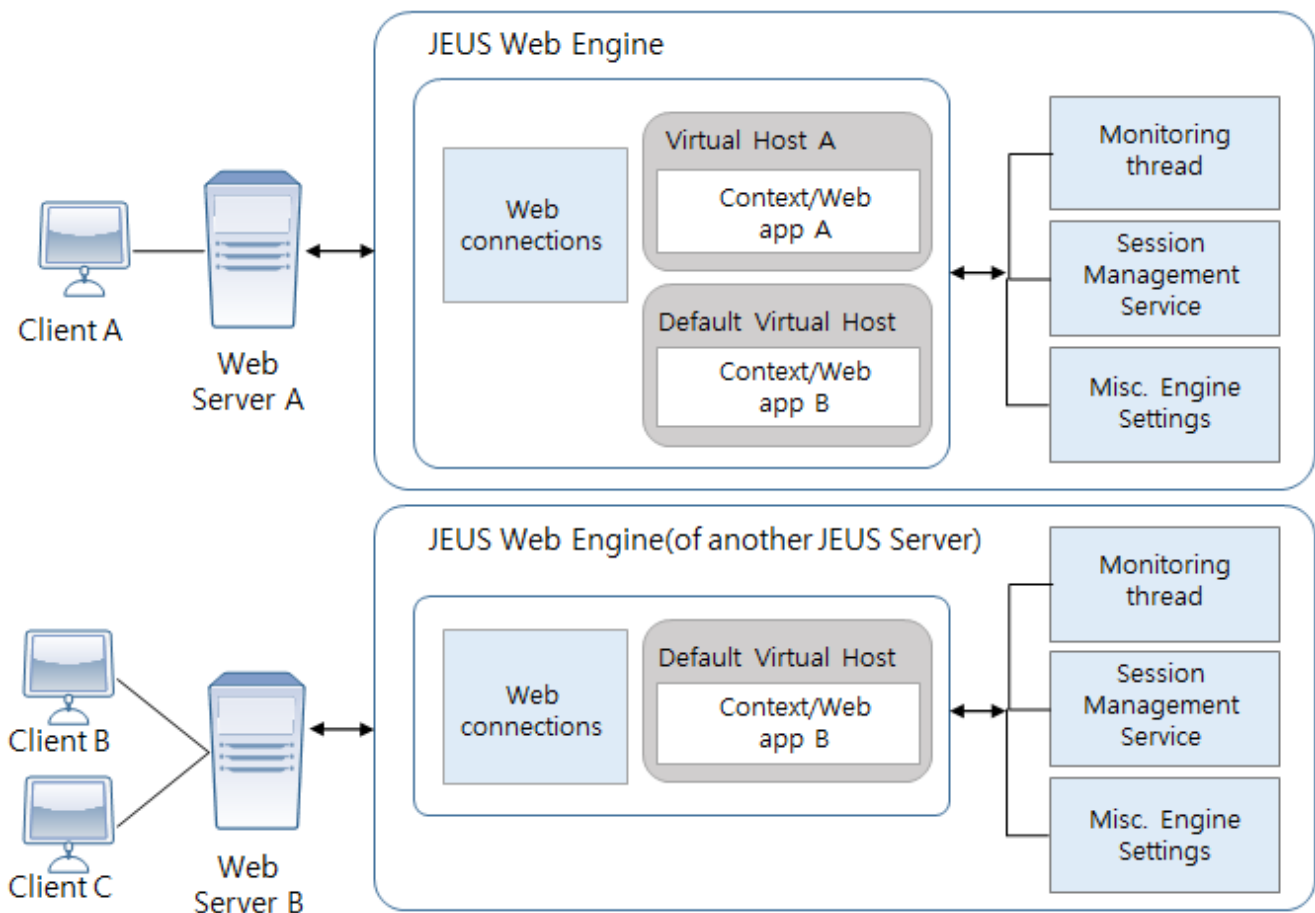
1.2. 세션 트래킹 구조

본 절에서는 기본적인 세션 트래킹의 구조에 대해 설명한다. 본 절에서는 매우 간단한 설명만이 제공되기 때문에 익숙하지 않은 사용자들은 서블릿과 세션 트래킹에 대한 내용을 참고한다.



본 절에서는 매우 간단한 설명만이 제공되기 때문에 익숙하지 않은 사용자들은 서블릿과 세션 트래킹에 대한 내용을 참고한다.

다음 그림은 세션 트래킹과 세션 관리에 관련된 웹 엔진의 컴포넌트의 구조이다.



JEUS 웹 엔진의 구조 중 세션 관련 부분

HTTP 세션은 동일한 클라이언트(예를 들면 웹 브라우저)의 HTTP 요청과 연관된 일련의 작업들이다. 여러 클라이언트가 이런 요청을 표준 HTTP를 통하여 보낼 때 기본적으로 HTTP 헤더에 유일한 "Client ID"가 없기 때문에 웹 서버는 이 클라이언트들을 구별할 수 없는 문제가 발생한다. 따라서 웹 서버는 관련된 사용자 요청을 하나의 세션으로 트래킹할 수 없다. 이것은 HTTP가 무상태(stateless)와 무연결(connectionless) 프로토콜이기 때문이다.



웹 서버도 세션 트래킹에 관련되어 있음을 주의한다.

HTTP 요청은 다음과 같이 작동한다.

1. 클라이언트는 웹 서버에 연결한다.
2. 클라이언트는 무상태 HTTP 요청을 생성한다.
3. 클라이언트는 응답을 받는다.
4. HTTP 연결이 끊어진다.

"Client ID"나 지속적인 세션의 개념은 HTTP 프로토콜에 포함되어 있지 않다. 따라서 웹 서버는 HTTP 연결이 끊어지거나 요청에 대한 응답 수신을 종료한 순간, 각 요청에 대한 사항들을 잃어버린다(단일 요청의 경우 발생). 그렇기 때문에 복잡한 웹 애플리케이션에서 사용자가 지속적으로 서로 연관된 요청을 수행하는 경우에는 사용이 불가능하다.

이 문제를 극복하기 위해 세션 ID라는 특수 string을 각 HTTP 요청에 추가한다. 이 유일한 ID는 클라이언트가 최초 요청을 할 때 필요에 따라 생성되어 클라이언트에 전달된다. 이후 클라이언트의 요청에 이 세션 ID가 지속적으로 붙여진다. 이러한 과정을 통해서 웹 엔진은 각 요청의 근원을 파악할 수 있기 때문에 사용자가 거래를 하는 동안에

대화성 상태(Conversational state)를 유지할 수 있고, 세션의 기능이 없는 HTTP 프로토콜을 이용하여 세션의 개념을 지원할 수 있다.

세션 ID는 쿠키 또는 클라이언트에게 반환되는 HTML 페이지의 각 URL 링크의 파라미터로 자동 추가되고 이것을 URL Rewriting이라고 한다. 또 다른 옵션은 hidden field로 HTML 폼에 세션 ID를 저장하는 방법이 있다. 물론 서블릿 프로그래머의 관점에서 이 논점들은 강력한 Servlet API에 의해 모두 구현되는 것들이다.

1.3. 세션 트래킹 동작

본 절에서는 JEUS 웹 엔진에서의 세션 트래킹 동작 방식과 클러스터 환경에서의 세션 트래킹 동작 방식에 대해 설명한다.

1.3.1. 웹 엔진에서 동작

JEUS 웹 엔진은 세션 트래킹을 가능하게 하기 위해 URL Rewriting과 쿠키를 지원하고, 그 중에서 기본적으로 쿠키가 사용된다. 세션 ID를 운반할 때 사용되는 쿠키를 세션 쿠키(Session Cookie)라고 한다.

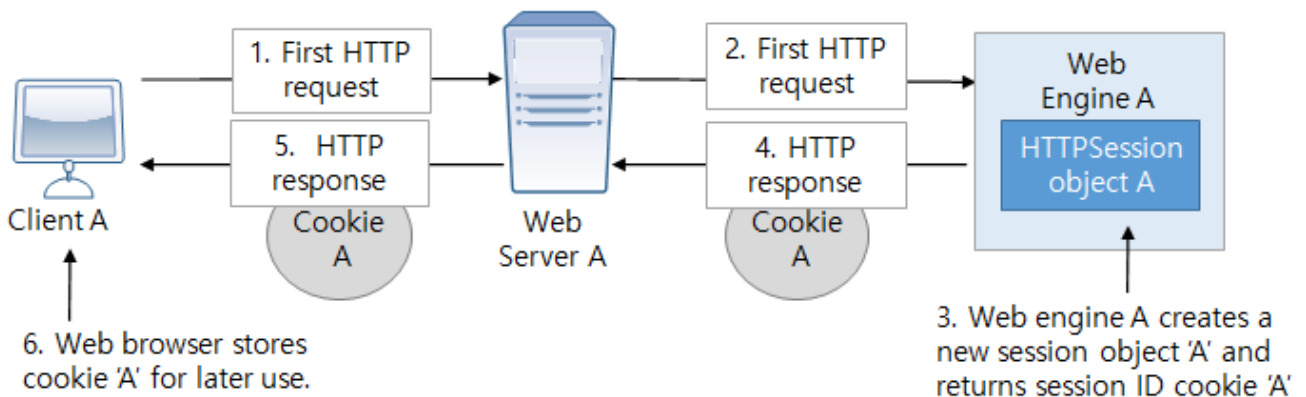
웹 엔진에서 하나의 세션은 하나의 Servlet API인 HTTP 세션 클래스의 인스턴스로 표현된다. 이 인스턴스는 세션 쿠키(또는 URL Rewriting의 결과로 생성된 URL 파라미터)의 세션 ID와 연관되어 있다. HTTP 세션 객체는 기본적으로 그것을 생성하는 웹 엔진에 존재한다. 또한 사용자 선호 성향이나 사용자가 전자 상거래 사이트에서 구매하고 싶은 물품의 목록 등과 같은 사용자에 대한 데이터를 가지고 있다.



URL Rewriting은 여기서 설명한 기술의 개념과 유사한 것이다. 단, 세션 ID가 HTML 페이지의 URL 링크에 포함된다는 것이 세션 쿠키가 별도의 쿠키 헤더(Cookie Header)에 포함된다는 것과 다른 점이다.

세션 쿠키는 하나의 클라이언트가 JEUS 웹 엔진이 관리하는 리소스를 요청하는 과정에서 사용한다.

다음은 웹 엔진이 세션 쿠키를 발급하는 과정에 대한 설명이다.



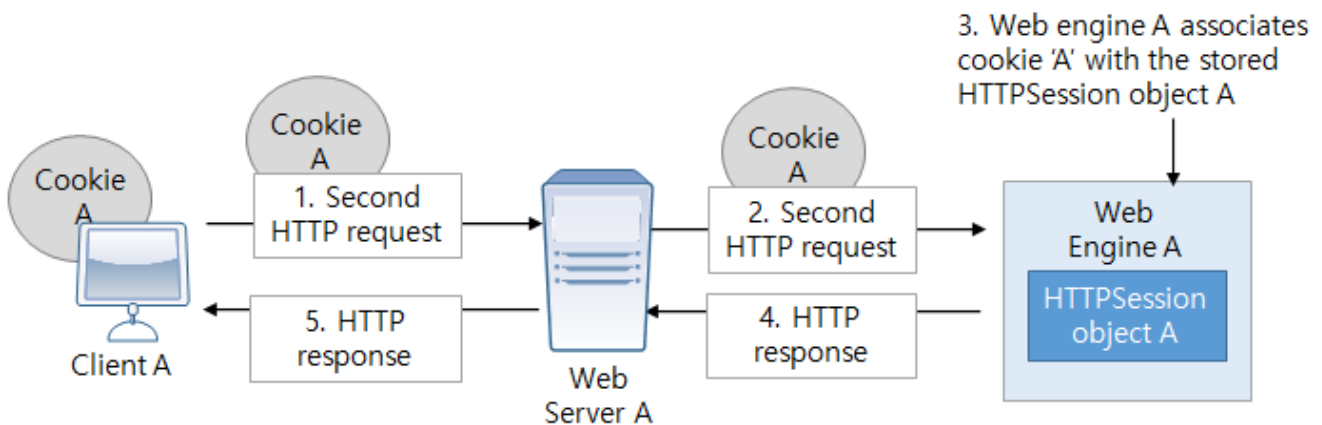
웹 엔진의 세션 쿠키 발급 과정

1. 클라이언트는 웹 서버에 초기 요청을 한다.
2. 웹 서버는 웹 엔진에 요청을 전달한다.

3. 웹 엔진은 해당 클라이언트를 위한 HTTP 세션 객체와 해당 HTTP 세션의 세션 ID를 지닌 세션 쿠키를 생성한다. 이렇게 생성한 세션 ID는 동일한 클라이언트가 지속적인 요청을 할 때 메모리에서 생성한 HTTP 세션 객체를 가져오기 위해 사용된다.
4. 응답 데이터와 세션 쿠키가 웹 서버에 전달된다.
5. 세션 쿠키는 클라이언트의 웹 브라우저에 응답과 함께 전달되고 HTTP 연결은 끊어진다.
6. 세션 ID를 포함하는 세션 쿠키는 클라이언트의 웹 브라우저에 저장된다.

이제 클라이언트는 세션 쿠키를 갖게 되었고, 동일한 웹 엔진에 또 다른 요청할 경우에 쿠키를 포함해서 보낼 수 있다. 웹 엔진은 쿠키의 세션 ID를 통해 클라이언트를 인식할 수 있고, 클라이언트의 HTTP 세션 객체를 가져올 수 있다.

다음은 이 과정에 대한 설명이다.



세션 ID 쿠키로 웹 엔진에 두 번째 요청을 보내는 과정

1. 클라이언트는 동일한 웹 서버에게 또 다른 요청을 보낸다. 이번에는 이전에 받은 세션 쿠키를 웹 브라우저에서 받아 요청에 첨부한다.
2. 웹 서버는 세션 쿠키가 포함된 요청을 받아서 처음의 요청과 같이 동일한 웹 엔진에 전달한다.
3. 웹 엔진은 요청과 세션 쿠키를 받는다. 세션 쿠키에서 발견된 세션 ID에 해당하는 HTTP 세션 객체를 자신의 메모리 영역에서 찾아온다. 웹 엔진은 찾아온 HTTP 세션 데이터를 사용하여 요청을 처리한다.
4. 응답 데이터와 세션 쿠키는 웹 서버에 전달된다.
5. HTTP 응답이 웹 브라우저에게 전달되고 HTTP 연결은 끊어진다. 세션 쿠키는 연결이 처음 생성될 때에만 응답에 포함되어 전달되면 되고, 그 이후의 응답에는 함께 전달될 필요는 없으므로 이 응답에 쿠키가 함께 전달될 필요는 없다.

1.3.2. 클러스터 환경에서 동작

웹 엔진에서 동작에서는 클라이언트, 웹 서버, 웹 엔진이 각각 하나씩 연계된 간단한 상황에서의 세션 트래킹 과정에 대해 살펴보았다. 그러나 실제 운영 환경에서는 이러한 간단한 구조는 충분하지 않은 경우가 많다. 다수의 사용자 요청을 수용하기 위해서는 부하 분산과 웹 서버 클러스터링이 구현되어야 한다. 클러스터의 구성에 대한 자세한 내용은 JEUS Web Engine 안내서의 **부하 분산을 위한 웹 서버 설정**을 참고한다.

웹 서버 클러스터에서 세션 트래킹 메커니즘을 구성하고 설정할 때에는 특별한 주의가 필요하다. 분산된 클러스터에서 세션을 관리할 때에는 다음 3가지의 주요 사항들이 쟁점이 된다.

1. 세션 쿠키를 포함한 요청을 처음 요청했던 웹 엔진에게 어떻게 전달할까?
2. 세션을 이용하여 모든 웹 엔진이 제한적인 요청을 처리하기 위해 하나의 엔진에서 생성된 HTTP 세션 객체를 어떻게 다른 웹 엔진에서도 사용할 수 있게 할까?
3. 내부 또는 외부적인 장애로 웹 엔진이 다운되었을 때 어떻게 세션 데이터를 백업할까?

첫 번째 논점은 **스티키 세션 라우팅(Sticky Session Routing)**이라는 기능으로 대처가 가능하고, 나머지 2가지 논점들은 **세션 서버(Session Server)**로 대처 가능하다. 본 절에서는 위 3가지의 문제와 해결 방법에 대해 상세히 설명한다.

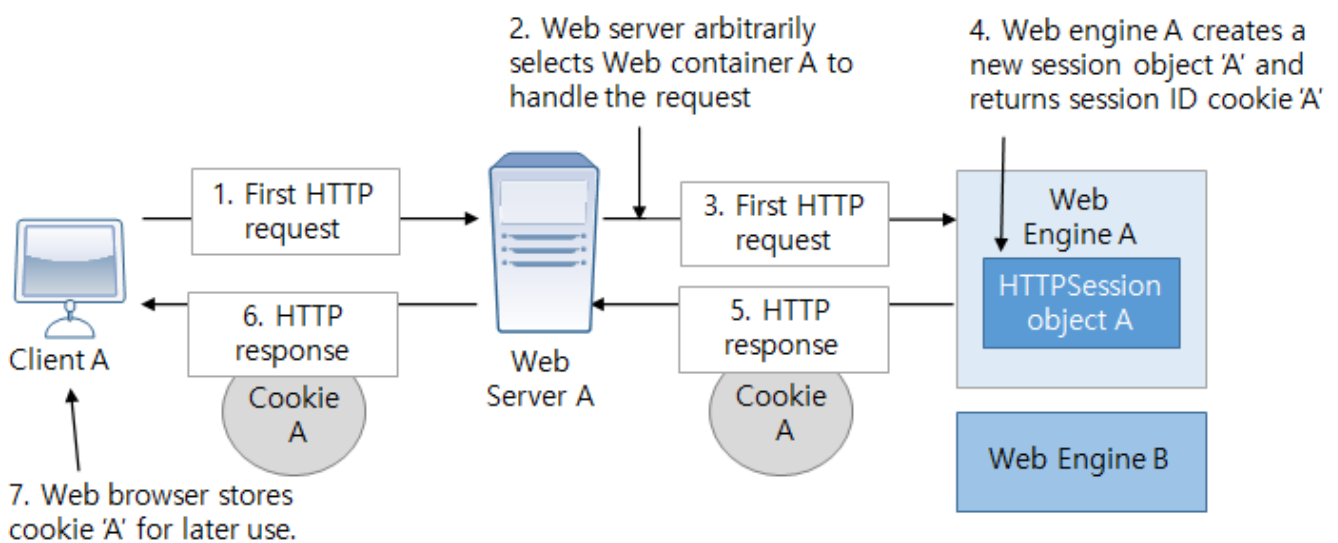


위의 첫 번째와 두 번째 논점은 근본적으로는 같은 문제이지만, 스티키 세션 라우팅과 세션 서버라는 서로 다른 2가지 방법으로 해결한다.

스티키 세션 라우팅(Sticky Session Routing)

스티키 세션 라우팅은 클러스터된 환경에서 해당 세션에 자신이 생성 또는 저장된 엔진의 ID를 부여하는 기능을 의미한다. 해당 엔진 ID가 쿠키에 포함되어 있기 때문에 웹 서버에서는 해당 엔진 ID를 확인하여 요청을 할 수 있다. 그로 인해 웹 엔진의 메모리에 저장되어 있는 세션의 효율을 증가시킬 수 있다.

예를 들어 2개의 웹 엔진 A, B가 하나의 웹 서버에 연결된 클라이언트 요청 상황은 다음과 같이 나타낼 수 있다.

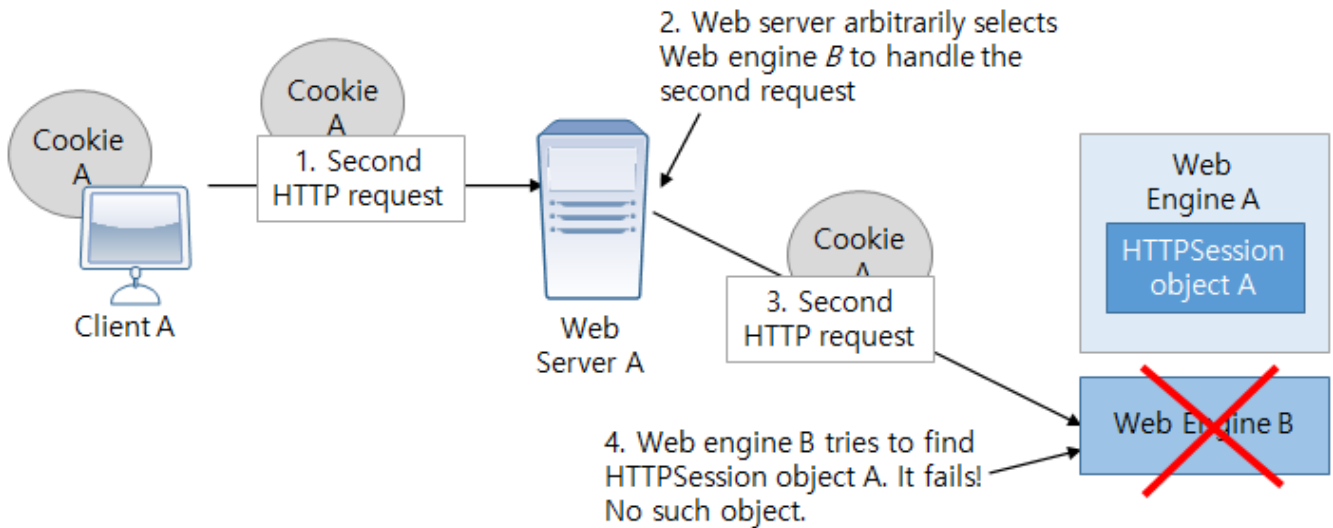


세션 ID 쿠키를 이용하여 2개의 웹 엔진과 세션을 시작하는 경우

1. 클라이언트는 웹 서버에 초기 요청을 한다.
2. 웹 서버는 임의적으로 요청 전달을 위해 1개의 웹 엔진을 선택한다. 예제에서는 웹 엔진 A가 선택되었다.
3. 요청은 웹 엔진 A로 전달된다.
4. 웹 엔진 A는 HTTP 세션 객체를 생성하고 응답과 함께 세션 ID 쿠키를 돌려보낸다. 이 ID는 다음에 오는 동일한 클라이언트의 요청을 처리할 HTTP 세션 객체를 메모리에서 불러올 때 사용된다.
5. 웹 엔진은 응답을 하고 웹 서버에 세션 쿠키가 반환된다.
6. 세션 쿠키는 응답과 함께 클라이언트의 웹 브라우저로 전달되고, HTTP 연결은 끊어진다.

첫 번째 요청은 문제없이 정상적으로 종료되었다. 그러나 두 번째 요청이 동일한 클라이언트로부터 생성될 때에는

심각한 문제가 발생한다.

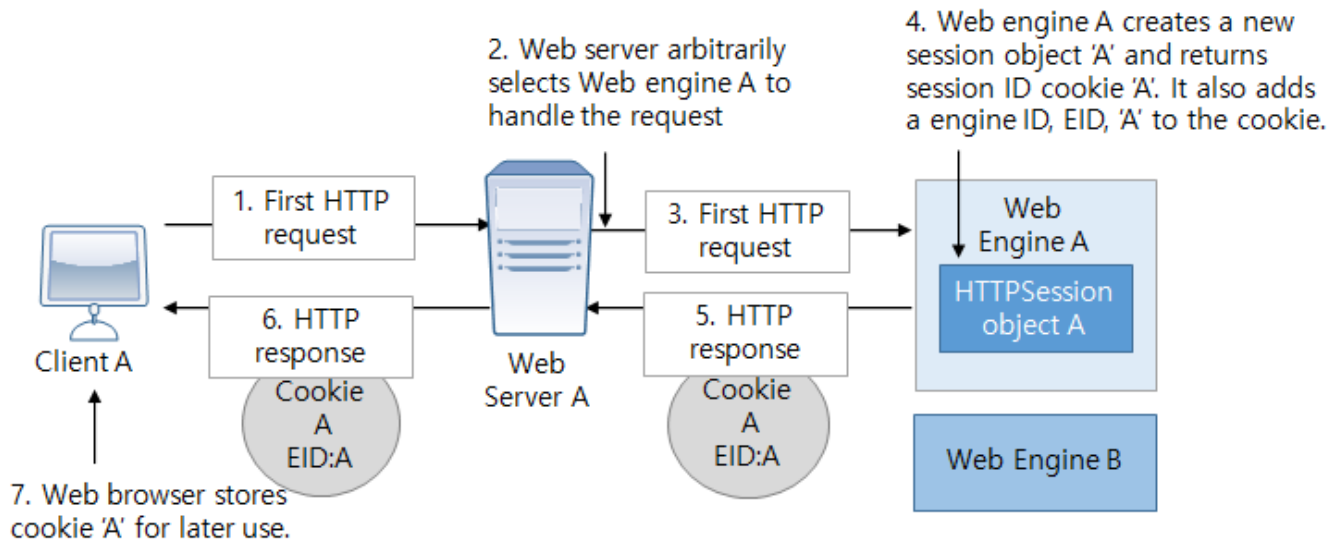


스티키 세션 라우팅이 없는 경우 클라이언트 세션 삭제

1. 클라이언트는 동일한 웹 서버에 또 다른 요청을 한다. 이번에는 첫 번째 요청 과정에서 반환된 세션 쿠키가 웹 브라우저로부터 가져와 요청에 붙여진다.
2. 웹 서버는 요청을 받아들이고 2개의 웹 엔진 중 임의로 1개의 웹 엔진을 선택한다. 이번 예제에서는 웹 엔진 B가 선택되었다.
3. 요청과 세션 쿠키는 웹 엔진 B에 전달된다.
4. 웹 엔진 B는 요청과 세션 쿠키를 받는다. 자신의 메모리 영역에서 쿠키에 대응하는 HTTP 세션을 가져오려고 시도하지만 대응하는 HTTP 세션 객체가 없기 때문에 가져올 수 없다. 따라서 웹 엔진 B는 클라이언트 세션을 유지할 수 없고 새로운 세션을 강제로 생성하거나 또는 오류 메시지를 반환한다(물론 두 옵션 모두 권장하지 않는다).

위의 설명과 같이 엔진의 수를 늘리고 동일한 서비스를 수행해서 트래픽을 분산시킬 수는 있다. 그러나 실제로 갖고 있는 정보를 찾을 수 없거나 전달할 수 없기 때문에 문제가 발생한다. 이 문제를 해결하기 위해 처음에 HTTP 세션 객체를 생성했던 동일한 웹 엔진에 세션으로 제한된 요청을 제대로 라우팅해줄 수 있도록 세션 쿠키에 웹 엔진 ID를 추가로 부여한다.

다음은 이 해결 과정을 그림으로 나타낸 것이다.



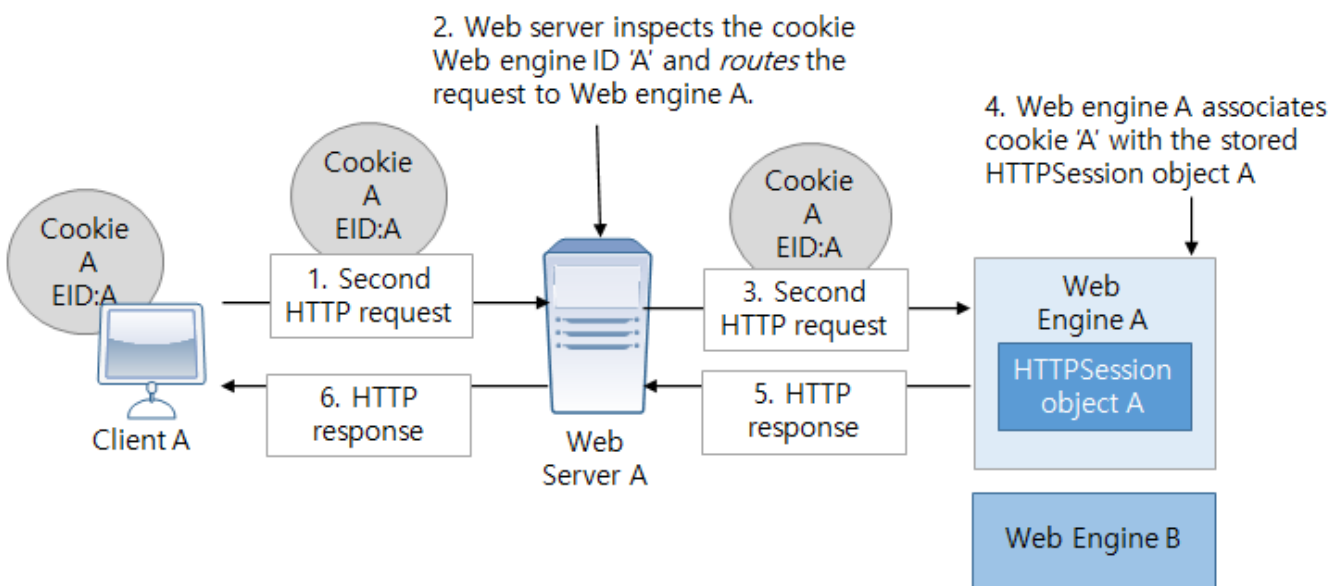
세션 쿠키에 추가의 웹 엔진 ID 부여

1. 클라이언트는 웹 서버에 초기 요청을 한다.
2. 웹 서버는 임의의 웹 엔진을 선택한다. 예제에서는 웹 엔진 A가 선택된다.
3. 요청은 웹 엔진 A에 전달된다.
4. 웹 엔진 A는 HTTP 세션, 세션 ID 쿠키를 생성하고 웹 엔진 ID(EID)를 쿠키에 삽입한다.
5. 웹 서버에 응답과 세션 쿠키를 반환한다.
6. 웹 브라우저에 응답과 세션 쿠키를 반환한다.
7. 웹 엔진 ID(EID)를 포함하는 세션 쿠키는 웹 브라우저에 저장된다.



JEUS 6에서는 해당 엔진의 ID를 직접 입력해서 운용했지만, JEUS 9에서는 해당 정보를 인코딩해서 운용한다.

다음은 스티키 세션 라우팅의 동작 과정이다.



스티키 세션 라우팅의 동작 과정

두 번째 요청에 웹 서버는 세션 쿠키와 엔진 ID 값(EID)을 찾아낸다. 웹 서버는 해당 HTTP 세션이 존재하는 원래의 웹 엔진으로 요청을 라우팅한다.



웹 서버에서 표준이 아닌 엔진 ID를 식별하기 위해서 Apache, IIS, SunOne(Iplanet)과 같은 다른 웹 서버의 경우에는 mod_jk 모듈을 설치해야 한다. WebtoB 웹 서버에 이 기능은 내장되어 있다. mod_jk 모듈 설치에 대한 자세한 내용은 JEUS Web Engine 안내서의 "부하 분산을 위한 웹 서버 설정"을 참고한다.

스티키 세션 라우팅 기능을 위해서는 모든 웹 서버가 모든 웹 엔진과 연결을 맺어야 한다. 부하 분산기를 사용할 경우 여러 대의 웹 서버 중에서 요청을 받은 웹 서버가 해당 웹 엔진에 접속하지 못하면 세션이 끊어질 수 있기 때문이다. 그러나 부하 분산기가 스티키 세션 라우팅을 자체적으로 지원하고 있다면 모든 웹 서버와 웹 엔진이 연결될 필요는 없다. 예를 들어 WebtoB를 부하 분산기로 사용하고 있다면, 스티키 세션 라우팅은 클러스터링이 완벽하게 연결되어 있지 않아도 사용할 수 있다.

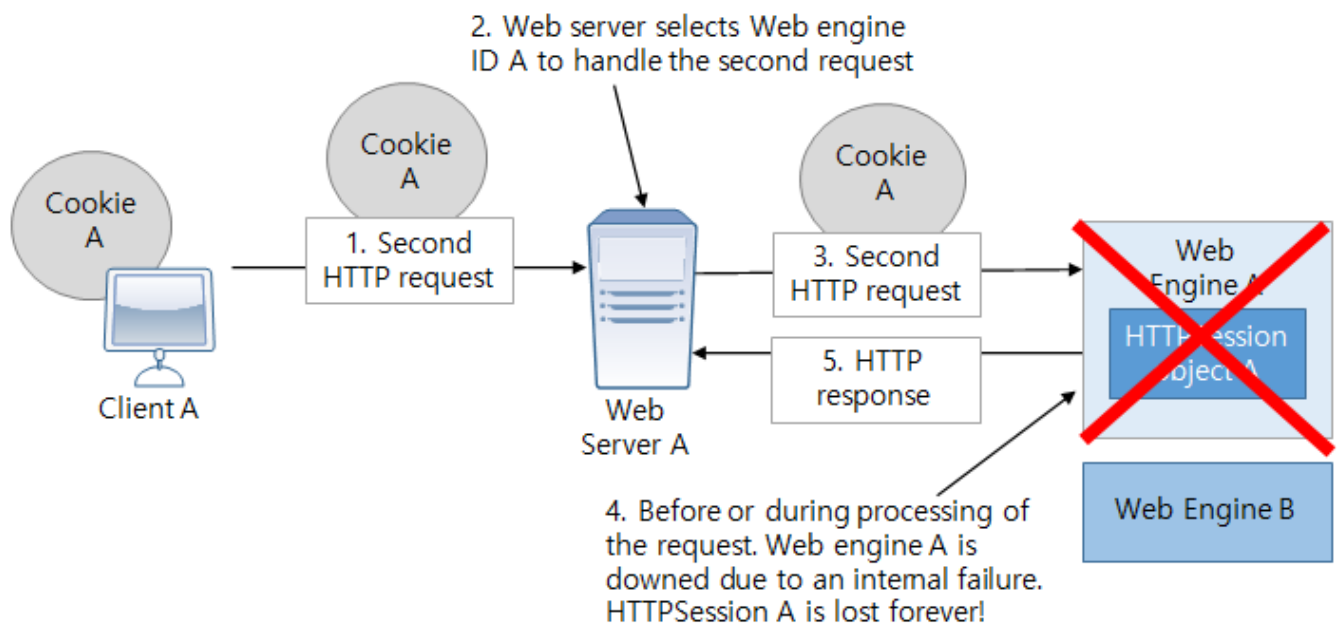
스티키 세션 라우팅 기술은 다른 동작에는 영향을 주지 않고 오직 요청을 전달하는 라우팅에만 관련이 있다. 스티키 세션 라우팅이 효율적으로 수행되면 세션의 중복 생성 및 중복 저장을 방지할 수 있기 때문에 성능이 많이 향상된다. 하지만 스티키 세션 라우팅을 사용해도 요청의 전달을 강제하지 않음을 상기할 필요가 있다. 효율을 높이는 부분이므로 요청을 보내야 할 엔진이 부하가 걸리거나 장애 상황에 있다면 요청은 다른 엔진으로 전달된다. 즉, 스티키 세션 라우팅 기술만으로는 완벽한 클러스터링 환경을 구성할 수 없다.

세션 서버(Session Server)

간단한 스티키 세션 라우팅을 사용하는 것보다 한 층 더 강력한 것이 세션 서버를 사용하는 것이다.

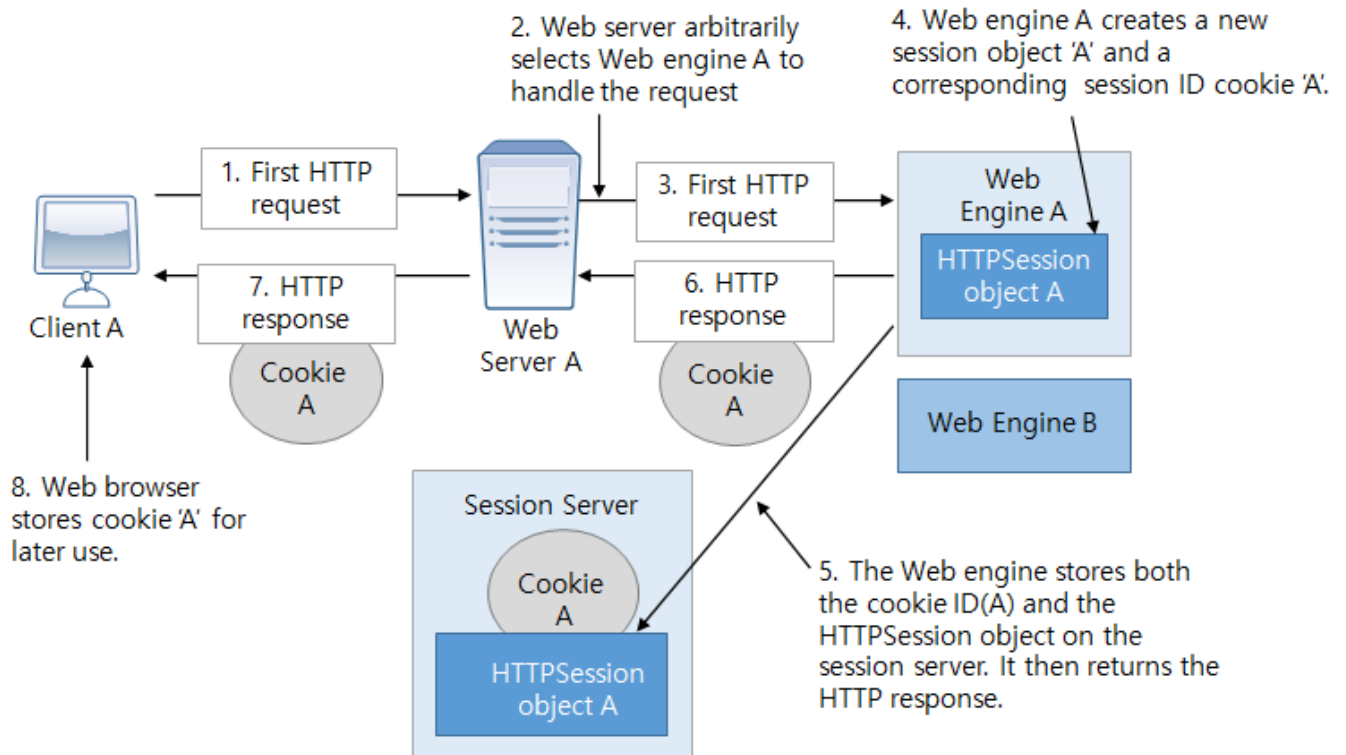
세션 서버를 사용할 때에는 모든 웹 엔진이 클러스터 내의 모든 웹 서버에 연결될 필요는 없다. 또한 세션 서버는 클러스터 내의 모든 세션 데이터에 대해 신뢰적인 백업 기능을 제공한다. 따라서 특정 웹 엔진에 장애가 발생해도 세션 데이터는 저장되고, 다른 정상적인 웹 엔진이 장애가 발생한 웹 엔진의 요청을 대신 처리한다.

다음은 웹 엔진에 존재하는 모든 사용자의 세션 데이터가 소멸되는 문제가 발생한 웹 엔진의 요청 처리 과정이다. 이런 상황은 운영 환경에서는 반드시 피해야 하는 상황이다.



장애가 발생한 웹 엔진의 문제

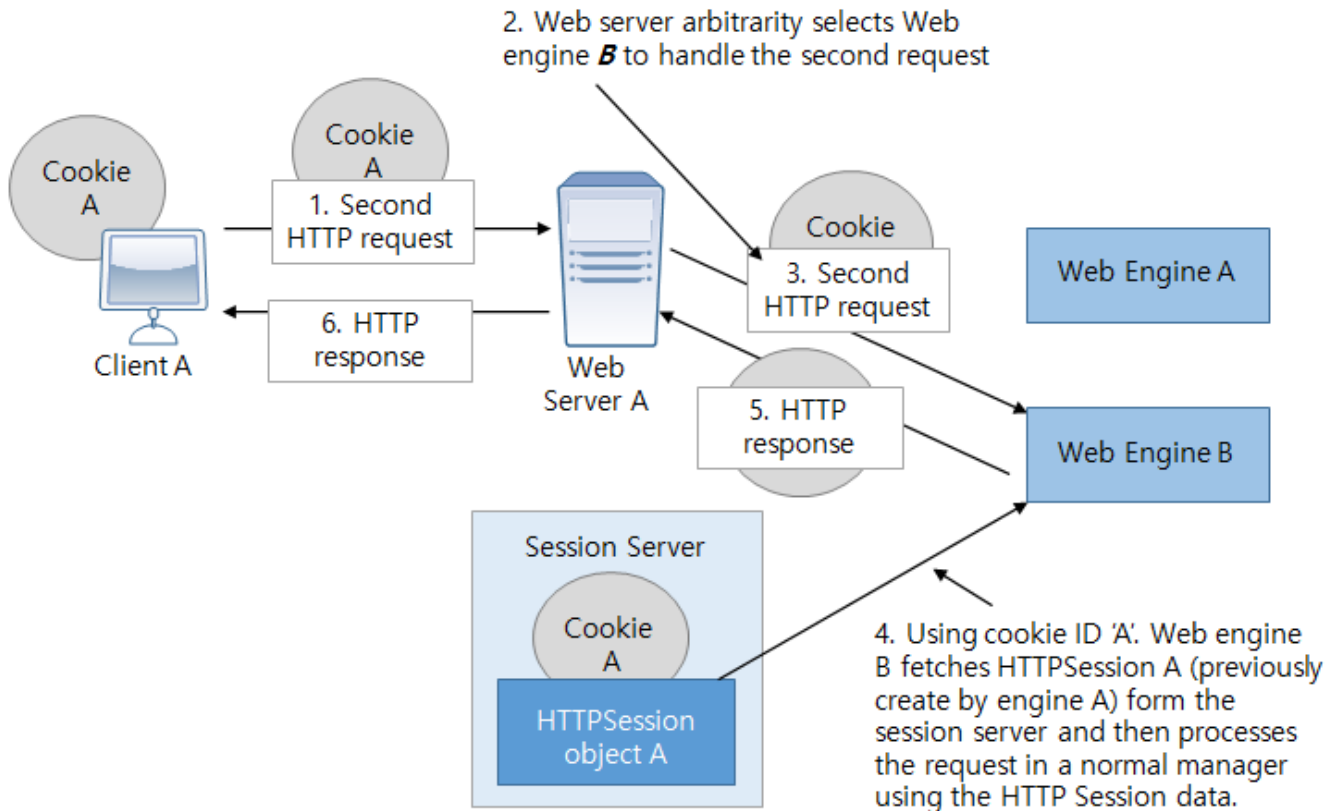
이와 같이 특정 웹 엔진에 장애가 발생하는 상황에서도 세션을 지속시키기 위해서 세션 서버가 클러스터에 추가되었다. 세션 서버를 사용할 경우에는 클라이언트의 첫 HTTP 요청이 다음과 같은 방법으로 처리된다.



세션 서버가 사용될 경우 클라이언트의 첫 HTTP 요청 처리

1. 클라이언트는 웹 서버에 요청을 보낸다.
2. 웹 서버는 웹 엔진 A를 클러스터 내에서 선택하여 요청을 처리하게 한다.
3. 웹 서버는 웹 엔진 A에 요청을 전달한다.
4. 웹 엔진은 HTTP 세션 객체와 세션 쿠키를 생성한다. 이 ID는 다음에 동일한 클라이언트로부터 요청이 왔을 때 세션 서버로부터 생성된 HTTP 세션 객체를 꺼내기 위해 사용된다.
5. 요청에 대한 처리가 완료되면 웹 엔진은 HTTP 세션 객체와 세션 ID를 세션 서버에 저장한다.
6. 응답 데이터와 세션 쿠키는 웹 서버로 전달된다.
7. 세션 ID 쿠키는 응답과 함께 웹 브라우저로 전달되고 HTTP 연결이 끊어진다.
8. 웹 브라우저는 이 세션 쿠키를 저장한다.

이후에 웹 엔진 B가 클라이언트 A의 요청을 처리하도록 웹 서버에 의해 임의로 선택되거나 웹 엔진 A에 장애가 발생하여도 세션 데이터는 세션 서버에서 가져올 수 있다.



세션 데이터 요청 처리

세션 서버는 위 그림과 같이 공용 저장소의 역할을 수행한다. 웹 엔진 A, 웹 엔진 B는 각각 별도의 저장 매체이며 서로의 내용을 참조할 수 없게 구성되어 있다. 그러나 세션 서버를 마치 둘의 공용 저장소처럼 두고 세션에 대한 정보를 저장하고, 또 불러오도록 하여 세션의 공유를 돕는다.

여기에서는 개념적으로 엔진 A, 엔진 B를 별도의 공간, 세션 서버를 공유되는 공간으로 비유하고 있다. 그러나 웹 엔진 A, 웹 엔진 B 또한 세션 서버처럼 공유되는 공간일 수 있고, 여기에서의 공유는 네트워크, 즉 다른 인터페이스 등을 통해 접근이 가능하다는 의미이다.

또한 좀 더 안정적인 세션 데이터 저장소를 위해 백업 세션 서버가 운용되고, 그로 인해 장애 사항에 대비할 수 있다. 백업 세션 서버는 내부적으로 운용되는 서버 중에서 자동 선택되며, 서버를 추가하거나 서버가 다운 및 fail되는 경우에도 자동으로 업데이트된다.

기존 버전에서 제공하였던 중앙 세션 서버 방식처럼 별도의 저장 공간을 두고, 어느 웹 엔진에서나 접근이 가능한 인터페이스를 제공함으로써 특정 웹 엔진에 장애가 발생하더라도 지속적으로 세션을 유지할 수 있는 장점이 있다.

그러나 중앙식 세션 서버의 방식은 대규모의 클러스터링 환경에서는 좋은 성능을 유지할 수가 없다. 그러한 이유는 중앙식 세션 서버에 모든 웹 엔진의 세션이 집중되기 때문에 부담이 가중되어 선형적인 성능 향상을 얻을 수 없는 단점이 있다. 또한 여러 대의 서버가 동작함에도 불구하고 단 하나의 서버만 동작하는 부분도 지금과 같은 분산 환경에서는 적합하지 않다.

이러한 흐름에 맞춰 분산식 세션 서버가 기본으로 운영된다. 분산 세션 서버는 대규모 클러스터링 환경에서 성능 향상을 위해 고안된 세션 서버이다.

분산 세션 서버는 각 웹 엔진마다 세션 서버를 두는 방식이다. 기본적으로 세션 라우팅 기술을 사용하며, 세션 서버와 마찬가지로 세션 데이터 백업을 설정할 수 있어 웹 엔진에 장애가 발생해도 지속적으로 세션을 유지할 수 있다.

분산 세션 서버에 대한 사용은 서버들 간의 클러스터에 포함되어 자동으로 제공되는 것을 원칙으로 하고 있다. 즉 서버 클러스터를 사용할 경우 해당 클러스터에 참여한 서버들의 컨텍스트들은 별도의 설정이 없어도 분산식 세션

서버를 통해 세션을 공유하고 유지하는 동작을 수행한다. 분산식 세션 서버에 대한 세부적인 설정에 대한 자세한 내용은 [분산 세션 서버 설정](#)을 참고한다.



JEUS 9에서는 세션 서버로 Jeus 자체 중앙식 세션서버와 Redis, Hazelcast를 지원한다. 설정 방법은 [세션 서버 설정](#)을 참고 한다.

혼합 모드(Mixed Mode)

혼합 모드는 위에서 설명한 세션 라우팅 방법과 세션 서버 방법이 혼합된 것이다. 각 방법의 장단점을 정리하면 다음과 같다.

	세션 라우팅 방식	세션 서버 방식
장점	웹 엔진의 세션 객체를 접근하므로 속도가 빠르다.	세션 서버에 모든 세션 객체가 저장되므로 특정 웹 엔진에 문제가 발생해도 세션 객체가 소멸되지 않고, 세션을 유지할 수 있다.
단점	문제가 발생할 경우 웹 엔진의 모든 세션 데이터는 소멸되고 복구가 불가능하다.	필요한 요청이 들어온 경우에 세션이 세션 객체를 세션 서버로부터 가져와야 하고 HTTP 세션 객체가 변경된 경우 세션 서버에 다시 저장해야 한다. 그러므로 세션 라우팅의 방법보다 처리 속도가 느리다.

혼합 방식을 사용함으로써 2가지 방식의 장점을 모두 살릴 수 있다. 2가지 방법이 혼합되면 세션 객체는 세션 서버와 세션 객체를 생성한 웹 엔진에 모두 존재하고, 웹 엔진은 필요할 때만 세션 서버에서 HTTP 세션 객체를 꺼내서 변경한다. 따라서, 혼합 방식을 사용할 경우에는 사용되는 네트워크 대역폭도 세션 서버만을 사용하는 방식에 비해 거의 절반 정도 줄이고 모든 세션 데이터의 안전한 백업도 보장받을 수 있다. 이 때문에 클러스터 환경에서는 혼합 방식의 세션 관리의 사용을 권장한다.

1.4. 세션 트래킹 모드

Servlet 4.0에서는 기존의 세션 트래킹의 여러 가지 방법을 선택할 수 있는 기능을 제공한다.

- 쿠키 사용 모드

기존의 기본적인 세션 트래킹 모드 중 하나는 쿠키를 활용하는 부분이다.

쿠키는 브라우저의 동작과 밀접한 관계를 가지고 있으며, 대부분은 이러한 쿠키를 통해 세션을 유지하고, 동작한다. 그러나 쿠키도 브라우저에 의존하기 때문에 규약이 다른 곳에서는 쿠키가 사용되지 않아서 세션이 유지되지 않을 수도 있다. 즉, 쿠키의 도메인과 경로에 대한 기본적인 지식이 요구되는 것으로 이러한 규약으로 인해 원하는 세션에 대한 트래킹이 가능한 것이다.

- URL 모드

URL Rewriting 모드는 브라우저가 쿠키를 지원하지 않을 경우에는 쿠키 사용 모드를 사용할 수 없기 때문에 존재하는 모드이다.

기존의 JEUS 6의 URL Rewriting과 동일한 방법이다. 즉, 쿠키를 사용하지 않아도 Target URL 자체에 원하는

세션에 대한 정보를 기록하는 것이다. 이러한 기능으로 웹 브라우저가 쿠키를 지원하지 않더라도 세션을 유지할 수 있다.

이 모드를 사용하려면, jeus-web-dd.xml DD(Deployment Descriptor)에 특수한 태그가 필요하다. <tracking-mode><url> 태그를 사용하면, 세션 ID는 URL Rewriting을 사용하여 유지된다. 이런 방법으로 세션 트래킹은 다른 도메인 이름이 몇 개의 요청에 사용되어도 동작한다.

- SSL 사용 모드

SSL 사용 모드는 SSL 연결에 세션을 제한적으로 전송하기를 원할 때 설정한다. JEUS를 운용할 때 세션에 보안상 보호되어야 할 데이터를 세션에 담는 경우가 존재하는데 이 경우에 설정하는 모드이다. SSL 사용 모드를 적용하면 SSL 연결이 아닌 곳에서 세션은 전송되지 않는다.

1.5. 컨텍스트간 세션 공유

일반적으로 세션은 동일한 컨텍스트에서만 관리되지만, 다른 컨텍스트 간의 공유도 가능하다.

세션 서버를 사용한다고 해서 모든 세션이 공유되는 것은 아니며, 서로 다른 컨텍스트 간의 세션 데이터 공유를 원한다면 분산 세션 설정의 **scope**를 설정한다. scope 설정은 다른 컨텍스트 간의 세션 공유를 위해 반드시 설정해야 하고, 이와 관련된 자세한 설정은 [분산 세션 서버 설정](#)을 참고한다.

1.6. 세션 트래킹 설정

세션 트래킹을 위해 세션 라우팅이나 세션 서버 기능을 사용하려면 다음과 같은 설정이 필요하다.

- 스티키 세션 라우팅 설정

기본적으로 세션 라우팅을 지원하기 때문에 지원 여부에 대한 별도의 설정을 필요로 하지 않으며 자동으로 지원한다.

추가적으로 엔진 정보의 인코딩에 설정이 존재한다.

항목	설명
BASE64	설정의 기본값으로 스티키되는 정보를 BASE64 룰에 의해 인코딩하여 전달된다. 세션 아이디를 통해 엔진정보나 도메인 정보가 노출되는 것을 원치 않아 직관적으로 의미 없는 정보로 보이도록 인코딩하여 전달한다.
BASE64_WITHOUT_PADDING	설정의 기본값으로 스티키되는 정보를 BASE64 룰에 의해 인코딩 후 PADDING("=") 없이 전달된다. 세션 아이디를 통해 엔진정보나 도메인 정보가 노출되는 것을 원치 않아 직관적으로 의미 없는 정보로 보이도록 인코딩하여 전달한다.

항목	설명
RAW	<p>스티키되는 정보를 인코딩하지 않고 그대로 노출하여 전달한다.</p> <p>인코딩되는 정보는 내부에서도 직관적으로 어떤 엔진에서의 요청인지에 대한 판단이 어렵기에 디버깅을 위해 설정하거나, 엔진 이름 노출이 보안상 영향을 주지 않을 경우에 설정한다.</p>

• 세션 서버 설정

분산 세션 서버를 사용하려면 클러스터 설정에서 해당 서버를 클러스터에 참여시키면 된다.



1. 세션 서버 및 스티키 세션 라우팅에 대한 자세한 내용은 [분산 세션 서버 설정](#)을 참고한다.
2. 설정은 콘솔 툴(jeusadmin)을 사용할 수도 있다. 콘솔 툴에서 사용하는 설정 관련 명령어는 JEUS Reference 안내서의 "set-sessionstorage-scope-session-config"를 참고한다.

1.6.1. 세션 설정

세션을 관리하기 위해 세션 객체의 공유 여부, 세션 쿠키 설정, Timeout 등 웹 엔진의 세션과 관련된 모든 사항에 대해 설정한다.

domain.xml을 사용하여 세션을 설정하는 방법은 다음과 같다.

1. domain.xml 에서 **[domain] > [server] > [web-engine] > [session-config]** 로 이동한다.

설정 항목들은 웹 엔진에서 공통적으로 사용할 세션 설정을 정의한다. 컨텍스트별로 설정을 오버라이드(override)할 수 있으며 컨텍스트, 웹 엔진 순으로 우선순위를 갖는다.

```
<session-config>
  <timeout>10</timeout>
  <max-session-count>-1</max-session-count>
  <reload-persistent>false</reload-persistent>
  <tracking-mode>
    <cookie>true</cookie>
    <url>false</url>
    <ssl>false</ssl>
  </tracking-mode>
  <session-cookie>
    <cookie-name>JSESSIONID</cookie-name>
    <url-cookie-name>jsessionid</url-cookie-name>
    <version>0</version>
    <path>/</path>
    <max-age>-1</max-age>
    <secure>false</secure>
    <http-only>true</http-only>
    <same-site>Disable</same-site>
    <partitioned>false</partitioned>
  </session-cookie>
</session-config>
```

다음은 설정 태그에 대한 설명이다.

• Tracking Mode

세션을 전달하는 방법(세션 트래킹)을 설정한다. 다음은 세션 트래킹을 설정하는 3가지 방법으로 중복 설정이 가능하고 설정하지 않으면 쿠키에 의한 트래킹만을 사용하도록 설정된다.

• Session Cookie

사용자의 세션을 추적하는 기본 기술은 모든 클라이언트 응답에 반환되는 세션 쿠키를 이용하여 구현된다. 이는 웹 엔진에서 응답을 내보낼 때 HTTP 헤더의 세션 쿠키에 대한 설정이다. 일반적으로 엔진에서 쿠키를 구성하지만 특별한 쿠키 정보를 구성하는 경우에 사용한다.

다음과 같은 세부 항목을 설정할 수 있다.

항목	설명
Timeout	<p>생성된 세션 객체의 유효 주기를 설정한다.</p> <p>보통 웹 애플리케이션 설정인 web.xml에서 Session Timeout을 설정한다. 만약 web.xml에서 특별한 설정을 하지 않았다면 여기서 설정한 값이 적용된다. 즉, web.xml의 Session Timeout 설정이 존재한다면 현재 설정값은 적용되지 않는다(web.xml의 Session Timeout이 우선순위가 가장 높다).</p> <p>Int 형식의 설정이며 분 단위의 시간을 설정한다. 설정하지 않을 경우 기본값으로 설정된다.</p>
Max Session Count	<p>메모리에 유지하는 최대 세션 수를 설정한다. 설정한 개수 이상의 세션이 유지되고 있을 때 세션의 생성 요청이 들어올 경우 오류를 발생시킨다.</p> <p>특정 애플리케이션의 세션 갯수 증가가 다른 애플리케이션에 세션 생성에 영향을 주어서는 안되기에 애플리케이션 별로 해당 세션 갯수가 적용된다.</p> <p>기본값은 -1이며, 무제한의 세션 생성을 허용한다는 의미이다.</p>
Reload Persistent	<p>일반적으로 서블릿 컨텍스트가 변경되어 리로딩이 발생할 때 해당 컨텍스트 내의 세션 객체의 속성들은 모두 삭제된다. 설정하지 않을 경우 기본값으로 설정된다.</p> <ul style="list-style-type: none">◦ true : 리로드를 수행해도 세션 객체의 속성들을 유지시켜 준다.◦ false : 리로드를 수행하면 세션 객체의 모든 속성들은 삭제된다. (기본값)
Cookie	<p>세션 트래킹 방법으로 쿠키를 사용하는 경우의 설정이다.</p> <ul style="list-style-type: none">◦ true : 쿠키를 통한 세션 트래킹을 사용한다. (기본값, 사용 권장)◦ false : 쿠키를 통한 세션 트래킹을 사용하지 않는다. 세션 유지가 정상적으로 되지 않을 수 있다.
Url	<p>세션 트래킹 방법으로 URL Rewriting 방법을 사용할 경우의 설정이다.</p> <ul style="list-style-type: none">◦ true : 세션 트래킹 방법 중 URL Rewriting 방법을 사용한다.◦ false : 세션 트래킹 방법에서 URL Rewriting 기술을 사용하지 않는다. (기본값)

항목	설명
Ssl	<p>세션 트래킹 방법으로 SSL을 사용할 경우의 설정이다.</p> <ul style="list-style-type: none"> ◦ true : 세션 트래킹을 할 때 SSL을 통해서만 세션을 전달한다. 정상적으로 동작하지 않을 수 있다. ◦ false : 세션 트래킹을 할 때 SSL이 아닌 연결을 통해서도 세션을 전달한다. (기본값)
Cookie Name	<p>세션 쿠키의 이름으로 표준 이름인 "JSESSIONID"를 사용하지 않고 다른 이름을 사용할 경우에 설정한다. String 형식의 설정이며, 설정하지 않을 경우 기본값으로 설정된다. (기본값: JSESSIONID)</p>
Version	<p>쿠키 ID의 버전을 설정한다.</p> <p>Int 형식의 설정으로 다음의 값 중에 하나를 설정한다.</p> <ul style="list-style-type: none"> ◦ 0 : NS 쿠키 유형을 가진다. (기본값) ◦ 1 : RFC 스펙의 쿠키 유형을 가진다.
Domain	<p>세션 쿠키가 전달될 때 서버의 도메인 이름을 설정한다. 쿠키는 이 도메인 요청에 대해서만 되돌아온다. 하나의 적합한 도메인 이름은 "."으로 시작되어야 하며, <host_name>을 지정해서는 안 된다. 이에 대한 자세한 내용은 "RFC-2109 스펙"을 확인한다.</p> <p>String 형식의 설정이며, 설정을 하지 않았을 경우 쿠키에 도메인 정보를 포함하지 않는다.</p>
Path	<p>세션 쿠키가 보내질 도메인 내의 String 형식의 URL 경로를 설정한다. 쿠키는 도메인이 적합할 때 해당 URL의 어떤 요청과 함께 보내진다.</p> <p>예를 들어 만일 "/examples"라는 경로가 설정되고, 도메인은 ".foo.com"으로 설정되었다고 가정할 때 클라이언트의 요청들은 "www.foo.com/examples"의 형식에 맞을 경우에만 해당 쿠키를 포함하여 서버로 요청한다. 이 또한 위의 도메인 설정과 더불어 "RFC-2109 스펙"을 확인한다.</p> <ul style="list-style-type: none"> ◦ 설정하지 않은 경우 : 엔진 내부에서 적절한 경로를 선택한다. ◦ 설정한 경우 : 설정된 고정 경로 정보가 항상 쿠키 정보로 포함한다. <p>경로의 최상위 값인 "/"가 아닌 다른 값으로 설정할 경우에는 애플리케이션들의 세션 공유 특성들을 고려하여 주의 깊게 값을 설정한다.</p>
Max Age	<p>세션 쿠키의 expires 속성을 설정한다. 이 시간 주기가 되면 쿠키는 클라이언트로부터 제거되고 더 이상 보내지지 않는다. Int 형식의 설정이며, 설정하지 않을 경우 기본값으로 설정된다. (기본값: -1, 단위: 초)</p> <p>기본값으로 설정하면 쿠키의 "expires" 속성을 사용하지 않겠다는 것을 의미한다. 즉, 브라우저의 LifeCycle을 따르겠다는 의미로, 브라우저가 닫힐 때 쿠키는 사용자의 세션이 끝남과 동시에 끝난다.</p>
Secure	<p>세션 쿠키의 secure 속성을 Boolean 형식으로 설정하고 설정하지 않을 경우 기본값으로 설정된다.</p> <ul style="list-style-type: none"> ◦ true : 세션 쿠키는 오직 Secure http connection인 HTTPS 위에서만 보내진다. ◦ false : 다른 연결에도 세션 쿠키가 전달된다. (기본값: false)

항목	설명
Http Only	<p>세션 ID 쿠키의 HttpOnly 속성을 Boolean 형식으로 설정한다. 설정하지 않을 경우 기본값으로 설정된다.</p> <p>HttpOnly 속성은 Servlet 3.0에 추가된 기술로 HTTP 외의 스크립트 요청에 의해서 해당 쿠키가 사용되는 것을 방지하는 보안 기술이다.</p> <ul style="list-style-type: none"> ◦ true : HTTP Request를 통해 서만 세션 쿠키가 사용된다. (기본값) ◦ false : 스크립트 코드 등에서도 세션 쿠키가 사용된다.
SameSaite	<p>세션 ID 쿠키가 사용자 의도와 상관없는 요청에 쓰이는 공격(사이트 간 요청위조)을 방지하는 보안기술을 설정한다.</p> <ul style="list-style-type: none"> ◦ Disable : same-site 쿠키를 삽입하지 않고, 브라우저의 기본 정책을 따른다. ◦ None : cross-site간 모든 쿠키를 허용한다. ◦ Strict : 쿠키가 cross-site로 보내지는 것(사이트 간 요청위조)을 방지한다. ◦ Lax : 사이트 간 요청위조에 비교적 안전한 cross-site 요청만 허용한다. 자세한 내용은 관련 스펙을 참고한다.
Comment	<p>사용자가 해당 쿠키에 대한 정보를 쉽게 알 수 있도록 하기 위해 해당 쿠키에 대한 목적 또는 설명을 기록한다.</p> <p>Netscape Version 0 쿠키에서는 지원되지 않는다.</p>



scope 설정을 통해 세션이 여러 컨텍스트들 간에 공유된다면 해당 Session Tmeout은 scope의 세션 타임아웃 설정을 따른다.



각 항목은 동적 설정 항목이 아니므로 서버를 재시작해야 변경 내용이 서버에 반영된다.

1.6.2. 세션 서버 설정

도메인 구조에서 세션 서버를 사용하려면 클러스터링에 참여해야 한다 . domain.xml에서 클러스터에 참여하는 서버들을 설정만 하면 된다. 이러한 설정으로 인해 컨텍스트별 설정 없이 세션 서버를 사용할 수 있다. 클러스터링 참여 방법 및 분산 세션 서버의 설정에 대한 자세한 내용은 각각 [세션 클러스터 모드](#)와 [분산 세션 서버 설정](#)을 참고한다.

1.7. 세션 트래킹 튜닝

클러스터된 환경에서 최적의 성능을 위해 다음과 같이 수행한다.

- 좋은 성능과 안정적인 운영을 보장하기 위해 항상 스티키 세션 라우팅과 세션 서버를 혼합하여 사용하도록 한다.
- 웹 서버들이 제대로 연결되서 설정되고 튜닝되어 있어야 한다.

- 사용자의 요청이 폭주하는 사이트에서는 분산 세션 서버를 사용할 것을 권장한다.

1.8. 세션 모니터링

생성된 세션들에 대한 기본적인 모니터링을 통해 현재 서버가 가지고 있는 세션의 개수를 확인할 수 있다.

세션의 모니터링은 다음과 같은 방법을 통해 가능하다.

- **콘솔 툴 사용**

콘솔 툴을 사용한 세션 모니터링 방법은 JEUS Reference 안내서의 "show-web-statistics"와 "list-session"을 참고한다.

2. 세션 서버

본 장에서는 클러스터링 환경에서 세션 트래킹을 위해 운용되는 분산 세션 서버의 구조, 동작 및 설정 방법 등에 대해서 설명한다.

2.1. 개요

세션 서버는 클라이언트의 세션 데이터를 관리하거나 백업하는데 사용된다. 그 중에서도 특히 여러 웹 서버들과 서블릿 엔진들이 클러스터링된 환경에서 세션 데이터를 관리할 때 유용하다. 세션 데이터의 관리 방식인 세션 서버로 중앙식 세션 서버와 분산식 세션 서버를 운용한다.



JEUS 6까지는 중앙식 세션 서버가 주로 사용되었으나, 시스템의 중앙 집중으로 인한 문제를 줄이기 위해 JEUS 7 이후부터는 분산식 세션 서버만을 사용했었다. 그러나 JEUS 9부터에는 Jeus 자체 중앙식 세션서버와 외부 Storage로 Redis와 Hazelcast를 지원하여 다시 중앙식 세션 서버도 사용할 수 있다.

2.2. 기본 개념

본 절에서는 분산 세션 서버의 기본 개념에 대해 이해를 돕기 위해서 중앙 세션 서버와 비교해서 설명한다.

2.2.1. 중앙 세션 서버

간단히 세션 서버는 JEUS에서 세션 객체를 저장하는 곳이다.

세션 객체는 서블릿 API의 HTTP 세션 객체를 나타낸다. HTTP 세션같은 객체는, 상태 유지가 안 되는 프로토콜인 HTTP 요청을 임시 데이터와 매핑한다. 그래서 이 임시 데이터를 통해서 클라이언트를 구별한다. 이렇게 함으로써 WAS는 작업 중인 클라이언트를 다른 클라이언트와 구분할 수 있으며, 이전에 작업한 내용을 기억할 수 있다. 이런 "Session Tracking"은 웹 사이트에서 가장 중요한 부분이다.

그러나 문제는 HTTP 세션 객체는 그 세션이 처음 생성된 서블릿 엔진에만 저장된다는 것이다. 이는 여러 서블릿 엔진들로 클러스터링 환경을 구성했을 경우 세션을 처음 생성한 서블릿 엔진에게 해당 클라이언트의 모든 요청을 보내야 한다는 의미이다. 그렇지만 항상 특정 서블릿 엔진으로 요청이 간다고는 볼 수 없다. 예를 들어서 웹 서버가 세션 라우팅을 지원하지 않거나 서블릿 엔진이 내부 에러로 인해 갑자기 종료(Down)될 경우 세션 객체를 잃어버리게 되기 때문이다.

중앙 세션 서버는 세션 데이터들을 한 곳에 모아서 관리한다. 즉, 어떤 클라이언트를 위한 세션 객체나 특정 서블릿 엔진에서 생성된 것도 모두 중앙 세션 서버를 이용하는 것이다. 이는 2가지의 이점이 있다.

- 첫째는 중앙 세션 서버를 사용함으로써 더 이상 웹 서버의 세션 라우팅이 필요없다.
- 둘째는 중앙 세션 서버를 사용하고, 백업용 중앙 세션 서버를 하나 더 사용하여 세션 데이터를 백업함으로써 시스템이 더욱 안정화된다.

이로써 서블릿 엔진이 다운이 되더라도 안전하게 세션을 유지할 수 있다.

2.2.2. 분산 세션 서버

분산 세션 서버는 세션 클러스터링 기능을 제공하면서 확장성 및 안정성을 개선한 방식이다. 따라서 대규모 클러스터링 환경에서 성능을 발휘할 수 있다.

분산 세션 서버는 기본적으로 세션 서버로서의 동작면에서는 기존의 중앙 세션 서버와 동일하다. 여러 서블릿 엔진들로 구성된 클러스터링 환경에서 동일한 클라이언트에서 요청된 일련의 요청들이 특정 서블릿 엔진에서 처리되지 않더라도 세션을 계속 유지해주는 기능을 제공한다.

분산식 세션 서버는 해당 세션들의 관리 주체들이 분산되어 있음을 의미하며, 이에 따라 높은 확장성을 제공한다.

분산 세션 서버의 특징은 다음과 같다.

- 여러 개의 서블릿 엔진으로 구성된 클러스터링 환경에서 지속적인 세션 유지가 가능하다.
- 바로 이전의 요청을 처리하던 서블릿 엔진이 다운되어도 다른 서블릿 엔진들이 이후의 요청을 처리할 때 세션이 끊기지 않도록 한다.
- 분산식 프로토콜을 사용하기 때문에 클러스터링 규모가 커지더라도 확장성이 용이하다.

2.3. 서버 구조

본 절에서는 중앙 세션 서버와 분산 세션의 구조를 비교해서 설명한다.

2.3.1. 중앙 세션 서버 구조

중앙 세션 서버는 JEUS 웹 컨테이너와 연결하여 운영되며, 웹 컨테이너에 있는 클라이언트의 세션을 라우팅하거나 백업하는 기능을 제공한다. 중앙 세션 서버는 Session Manager, Session Cache Memory, Session Storage 등으로 되어 있다. Session Storage로는 JEUS 자체 중앙식 Session Server와 Redis, Hazelcast를 지원하고 있다.

중앙 세션 서버의 서브 컴포넌트는 다음과 같다.

- Session Manager
 - 엔진의 세션 관리를 총괄하는 모듈이다. 로컬 메모리, 세션 서버로부터 세션을 얻어오거나 관리한다.
 - 로컬 웹 엔진은 getSession을 통해 세션 매니저로부터 사용할 세션 객체를 얻어온다. Cached Session, Session Storage들 순서로 세션을 얻어온다.
 - 세션에 대한 처리가 끝나면 Session Storage로 update한다.

- Cached session(Session Cache Memory)

활성화되어 있거나 한 번 사용된 세션 객체는 빠른 사용을 위해서 이곳에 저장된다.

- Session Storage

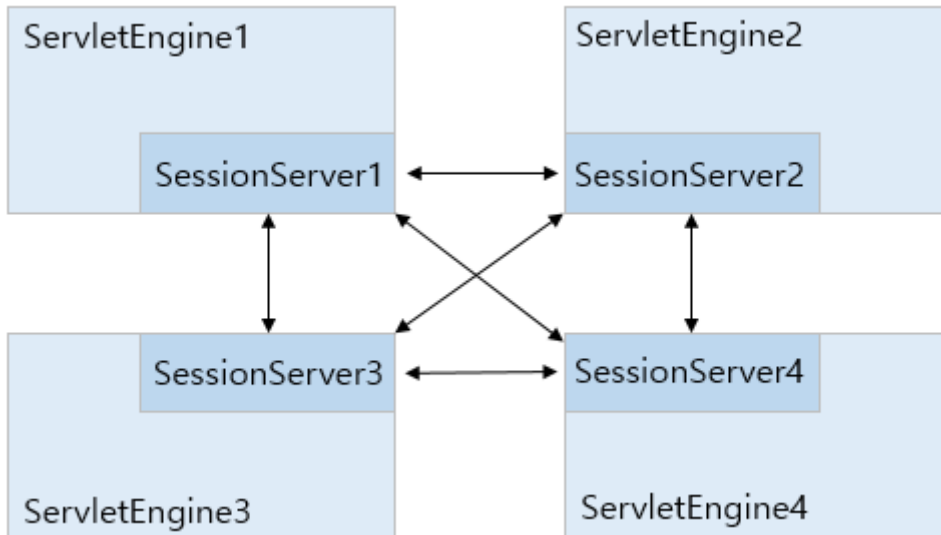
세션을 저장하는 Session Server 역할을 한다. Storage로는 JEUS 자체 중앙식 Session Server와 Redis, Hazelcast를 지원한다.

2.3.2. 분산 세션 서버 구조

분산 세션 서버는 세션 객체를 서비스하는 서버가 각각의 서블릿 엔진(웹 엔진) 또는 각각의 EJB 엔진에 분산되어 있는 분산식 구조이다.

분산 세션 서버 방식은 클러스터링에 참여하는 모든 엔진 내에 독립적인 분산 세션 서버가 존재하고, 이들 분산 세션 서버들이 Peer-to-Peer로 다른 엔진의 분산 세션 서버와 통신하여 지속적인 세션 서비스를 제공한다.

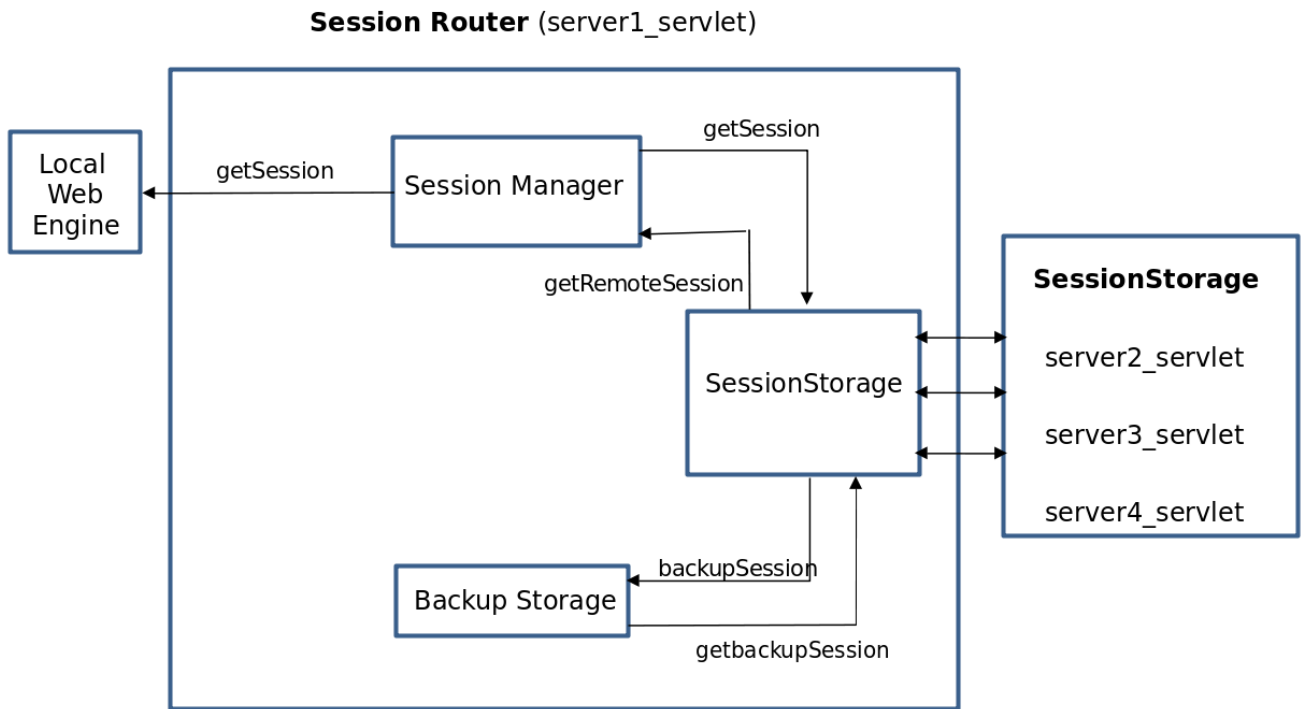
다음은 분산 세션 서버 방식을 사용하여 4개의 웹 엔진을 세션 클러스터링하는 구조이다.



분산 세션 서버를 이용한 세션 클러스터링 구조

위의 구조도에서 화살표는 분산 세션 서버 간의 소켓 연결을 나타낸 것이다. 항상 모든 연결을 맺는 것이 아니고, 연결에 대한 가능성을 의미한다. 일반적으로 연결이 필요없으며 세션 유지를 위해 다른 엔진과의 통신이 필요한 경우에만 연결을 맺고 유지한다. 장애 발생의 경우를 제외하고 연결은 보통 하나씩만 갖는다.

다음은 웹 엔진의 서브 컴포넌트로 동작하는 분산 세션 서버의 내부 구조이다.



분산 세션 서버 내부 구조

• Session Manager

- 엔진의 세션 관리를 총괄하는 모듈이다. 로컬 메모리, 파일 및 다른 리모트 분산 세션 서버로부터 세션을 얻어오거나 관리한다.
- 로컬 웹 엔진은 getSession을 통해 세션 매니저로부터 사용할 세션 객체를 얻어온다. session의 sticky server, 이전 backup server들 순서로 세션을 얻어온다.
- 수정된 세션은 리모트 백업이 있을 경우 Cluster Manager를 통해, 백업 서버로 정해진 다른 웹 엔진의 분산 세션 서버로 in-memory 백업된다(backupSession).

수정된 세션은 '**Backup Level**' 설정에 따라 다르게 적용된다. '**Backup Level**'이 'all'일 경우 항상 'get'일 경우 getter와 setter의 호출할 때, 'set'일 경우 setter호출할 때에만 (setAttribute, removeAttribute, setMaxInactiveInterval)의 세션 operation이 발생했을 때 수정된 세션으로 판단한다.

'**Backup Level**' 항목의 설정값에 대한 자세한 설명은 [세션 서버 설정](#)을 참조한다.

• Backup Storage

- 자신의 분산 세션 서버를 백업으로 선택한 다른 엔진의 분산 세션 서버가 주기적으로 전송하는 백업 세션 객체를 관리한다. 이 백업 세션 객체는 원본을 가지고 있는 엔진에 장애가 발생한 경우 대신하여 세션을 제공한다.
- 자신을 백업으로 지정한 리모트 웹 엔진이 전송하는 백업 세션을 받아서 저장 및 관리한다(backupSession, getBackupSession).
- 로컬에서 수정된 세션 객체는 조건이 만족되면 지정한 리모트 분산 세션 서버로 백업된다. 이러한 백업은 요청이 끝나면 바로 수행되어서 장애 상황에서의 Failover가 가능하도록 한다.

• Cluster Manager

- 리모트 웹 엔진에 있는 세션들에 대해서 특정 operation을 할 경우 중재 역할을 하는 모듈이다.

해당 엔진은 dynamic한 환경을 고려하여 운용된다. 즉, 기동환경 중에 운용하는 서버가 추가되거나 제거된 경우 해당 상황에 적절한 백업을 선택하여 운용되고, 이것은 환경설정 중심에서 벗어난 큰 특징 중 하나이다. 리모트 엔진(RemoteEngine)에 대한 연결은 지속적인 ping을 통한 확인이 아닌 SCF를 통한 방식으로 지속적인 장애 상황에서의 불필요한 오류 상황을 제거해준다.

리모트 웹 엔진로부터의 getSession(), removeSession() 등을 예로 들 수 있다.



기존 환경 중심의 구조에서는 설정으로 인해서 고정된 백업 엔진만을 바라보게 되었지만 dynamic하게 변화하는 리모트 엔진에 대해서 유동적인 변경이 가능하도록 설계되었다.

2.4. 동작 방식

본 절에서는 중앙 세션 서버와 분산 세션의 동작 방식을 비교해서 설명한다.

2.4.1. 중앙 세션 서버

웹 컨테이너는 중앙 세션 서버와 연결된다. 한 번 연결할 때 웹 컨테이너는 세션 객체가 새로 생성되거나 수정된 세션으로 판단되면 중앙 세션 서버에 세션 객체를 저장하거나 업데이트한다. 그리고 클라이언트의 요청이 들어올 때마다 저장소로부터 세션 객체를 가지고 온다.

2.4.2. 분산 세션 서버

분산 세션 서버 방식은 세션 라우팅 기능을 기본으로 동작한다. 세션 ID에 특정 서버의 이름을 붙여서 발급하여 기존에 생성된 서버의 위치를 알고 해당 정보로부터 세션을 유지한다. 그렇기 때문에 세션 라우팅 기술을 강제로 사용하지 않을 경우에는 성능의 효율이 떨어진다.

세션 라우팅을 지원하는 웹 서버를 앞단에 배치하면 정상적인 경우 세션 객체가 존재하는 웹 엔진으로 요청이 라우팅될 것이다. 따라서 이러한 경우의 동작 방식은 세션 라우팅에 의한 세션 클러스터링과 동일하다. 이에 대한 내용은 [클러스터 환경에서 동작](#)을 참고한다.

세션 라우팅 기능을 기본으로 동작하는 분산 세션 서버 방식에서는 SessionKey를 사용한다.

다음은 웹 엔진에서 사용되는 SessionKey의 예제이다.

```
<SessionID>.<primary-engine-name>  
예) XXX.domain1/server1
```

항목	설명
XXX	<SessionID>를 상징적으로 나타낸 것으로 실제 <SessionID>는 이보다 훨씬 긴 Random String 형태이다.

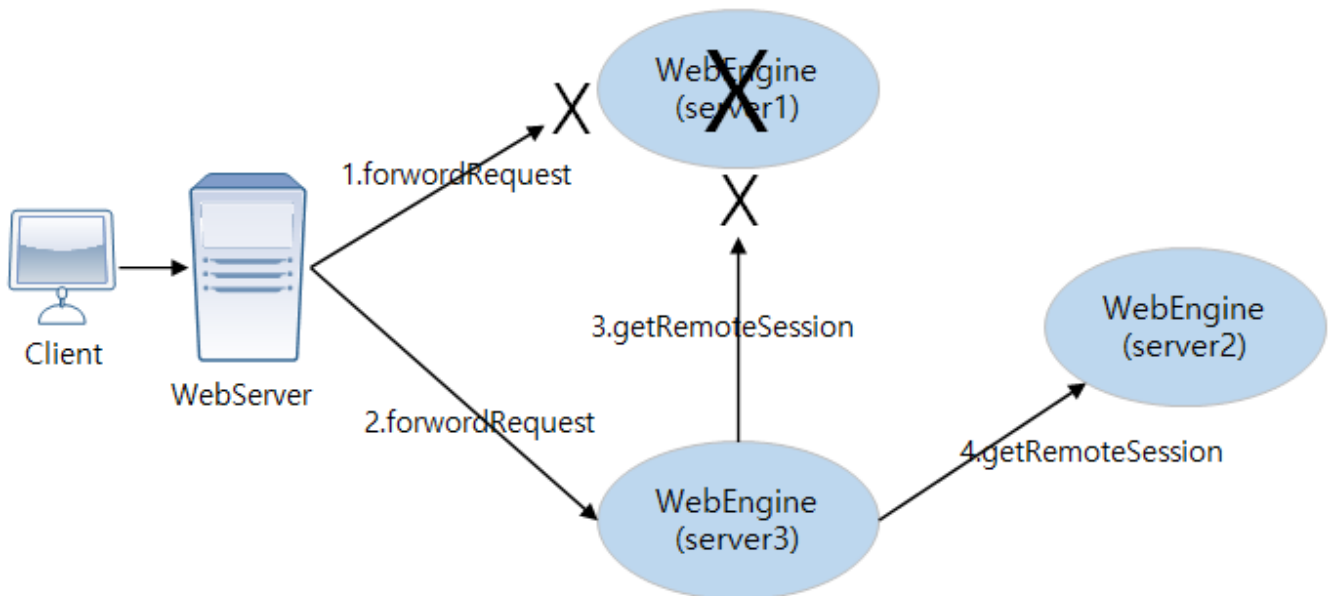
항목	설명
domain1/server1	라우팅 정보를 의미한다. domain1이라는 서버의 서블릿 엔진을 의미한다.

각 엔진은 세션 라우팅에 사용하는 세션 라우팅 ID를 하나씩 부여받는다. 이 ID는 웹 서버에 웹 엔진이 접속할 때 웹 엔진을 구분하는 구분자로 사용된다. 세션 라우팅 ID는 설정에 의해 자동 생성된다.

분산 세션 서버에 의한 Failover 구조의 3개의 웹 엔진은 다음과 같은 세션 라우팅 ID 및 백업 서버의 세션 라우팅 ID가 할당되었다고 가정한다.

- WebEngine(server1)
 - 세션 라우팅 ID : domain1/server1
 - 백업 서버의 세션 라우팅 ID : domain1/server2
- WebEngine(server2)
 - 세션 라우팅 ID : domain1/server2
 - 백업 서버의 세션 라우팅 ID : domain1/server3
- WebEngine(server3)
 - 세션 라우팅 ID : domain1/server3
 - 백업 서버의 세션 라우팅 ID : domain1/server1

세션 라우팅 ID가 할당된 분산 세션 서버에 의한 세션 트래킹의 Failover 구조는 다음과 같다.



- Request Session Key : XXX/domain1/server1
- Reponse Session Key : XXX/domain1/server3

분산 세션 서버에 의한 Failover 구조

다음은 분산 세션 서버에 의한 Failover 구조의 동작에 대한 설명이다.

1. 웹 서버는 세션 라우팅 ID를 분석하여 WebEngine(server1)으로 해당 Request의 전송을 시도하지만

WebEngine(server1)은 fail 상태이다.

2. 웹 서버는 다른 WebEngine 중에서 임의로 하나를 선택하여 요청을 보낸다. 본 예제에서는 WebEngine(server3)을 선택하였다.

웹 서버로부터 요청을 전달받은 WebEngine(server3)은 세션 라우팅 ID를 분석한다. 분석 결과 이 요청을 처리할 세션 객체는 WebEngine(server1)에 있으며, 해당 세션의 백업은 WebEngine(server2)에 존재한다는 것을 알게 된다.

3. 분산 세션 서버인 WebEngine(server3)은 요청 분석 결과에 따라 WebEngine(server1)에 접속하여 세션 객체를 가져오려고 시도한다. 그러나 WebEngine(server1)에 장애가 발생했으므로 해당 요청의 시도는 실패한다.
4. 요청이 실패로 끝났기 때문에 WebEngine(server3)은 WebEngine(server1)의 백업 서버인 WebEngine(server2)로 다시 시도한다(Backup Migration). WebEngine(server2)은 요청에 대해 백업해놓은 세션 객체를 WebEngine(server3)에게 넘긴다.
5. 성공적으로 세션 객체를 가져온 WebEngine(server3)은 이후 클라이언트의 요청을 처리하여 응답 메시지를 클라이언트에게 보낸다. 이때 새로운 SessionKey를 작성하여 보내기 때문에 클라이언트의 SessionKey가 변경되고 이후 요청이 WebEngine(server3)로 들어온다. 새로운 SessionKey를 작성할 때는 세션 라우팅 ID 부분만 자신의 것으로 치환한다.

웹 서버가 세션 라우팅을 지원해도 웹 서버에 세션 라우팅 ID에 해당하는 웹 엔진으로의 연결이 존재하지 않거나 해당 웹 엔진에 장애가 발생하여 요청을 전달할 수 없는 경우에 웹 서버는 세션 라우팅을 지원할 수 없다. 이러한 경우 웹 서버는 보통 임의로 선택된 다른 웹 엔진으로 요청을 전달한다.

다음은 세션 라우팅 기능을 사용하지 않는 상황을 가정할 경우의 [분산 세션 서버에 의한 Failover 구조](#)의 분산 세션 서버 동작에 대한 설명이다.

1. 세션 라우팅 기능을 사용하지 않는 상태라면 세션 라우팅 ID는 존재하지 않는다. 웹 서버는 임의의 WebEngine을 선택해서 요청을 보낸다. 본 예제에서는 WebEngine(server1)이 선택되었다.
2. 웹 서버는 WebEngine(server1)이 fail 상태임을 감지하고 다시 임의의 WebEngine을 선택하여 요청을 보낸다. WebEngine(server3)이 선택되었다.
3. 분산 세션 서버인 WebEngine(server3) 또한 해당 세션에 대한 라우팅 정보를 알 수 없기 때문에 현재 접속 중인 분산 세션 서버로 순서대로 요청한다. 해당 요청은 WebEngine(server1)에 요청을 보냈으나 이 시도는 실패하였다.
4. 분산 세션 서버는 해당 요청을 다시 WebEngine(server2)에 보내고 WebEngine(server2)은 해당 세션을 전달한다.
5. 세션을 전달받은 WebEngine(server3)은 SessionKey의 변경 없이 클라이언트로 보낸다. 이에 따라 다음 요청도 해당 세션을 가지고 있는 WebEngine(server3)으로 오지 않을 수 있다.



설명을 위해서 라우팅 정보를 domain1/server1과 같이 표기했으나 운용 중에는 해당 정보가 인코딩되어 동작한다.

세션은 한 리퀘스트 안에서 의미가 있는 객체로서 다른 JVM이나 다른 스레드에서 동시 조작등은 스펙상 보장을 하지 않는다. 즉, 애플리케이션의 책임이며 해당 환경은 지양이 되어야 하는 사항이다.

애플리케이션의 구성상 동시 요청을 보내야 한다고 한다면 해당 요청들은 각기 다른 세션을 바라보게 구성하거나 세션을 사용하지 않도록 해야 정상적으로 동작한다. 하지만 세션의 공유는 매우 매력적인 사항이며 해당 사항을

이용하고 애플리케이션을 구성하는 사례들이 증가 하고 있다.

이러한 동시 다발적인 요청이 세션과 연계가 된다면 제약 사항들이 존재하며 그것은 아래와 같다.

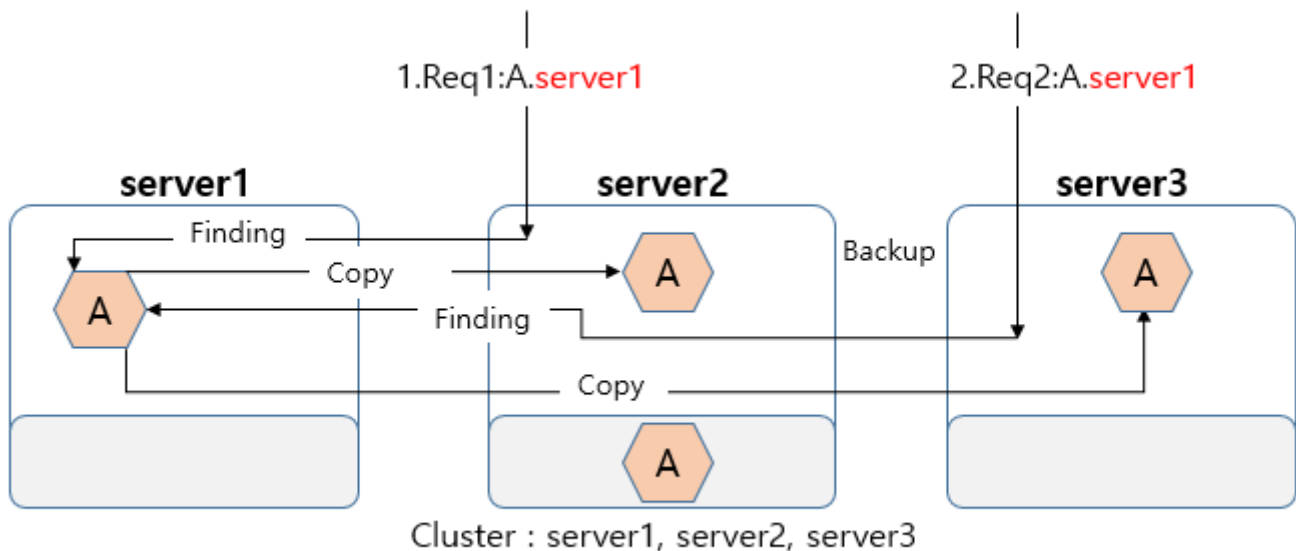
1. 새로운 세션이 동시에 생성될 가능성이 존재하며, 해당 세션 중 하나의 세션만 쿠키로 유지가 되기에 불필요한 메모리의 낭비가 발생할 수 있다.

각 요청이 다른 JVM으로 이동하거나, 동시에 처리될 경우에 모두 포함되는 행동이다. 세션이 존재하지 않을 경우 세션을 생성하는 것이 기본적인 동작이기에 동시에 전달된 요청은 각각 새로운 세션을 발급 받게 되는 것을 막을 수 없다. 이러한 세션들은 사용자의 브라우저의 쿠키에 저장되는데 해당 쿠키는 결국 하나의 스코프로 관리되어 마지막으로 발급 받게 되는 세션으로 덮어 씌워진다. 이로 인해 다른 요청에 의해 생성된 세션은 별도로 접근할 수 없으며 타임아웃에 의해 제거될 수 밖에 없다.

2. 세션이 동시에 다른 JVM에서 유지되기 위한 방식에 따라 세션을 유지하지 못할 수 있다. 이 제약 사항 스티키 세션 라우팅과 연계되어 있는 제약사항으로 동일한 서버나 웹 엔진으로 전달되었을 때는 해당되지 않는 문제이다. 애플리케이션의 구성이나 환경으로 인해 다른 JVM으로 요청이 동시에 전달될 때에는 세션 유지하는 방식에 의해 세션이 유지되지 않을 수 있다.
3. 세션에 동시에 업데이트를 수행할 경우 해당 세션에 반영이 무시될 수 있다.

2번의 사항과 마찬가지로 다른 서버나 웹엔진에 전달되었을 경우에 해당되는 문제이다. 세션의 업데이트는 하나의 요청에 의해서 처리되는 것을 가정하였기에 그 결과의 반영도 세션 생성과 동일하게 타이밍에 의해 반영이 무시될 수 있다.

제약사항에서 언급이 되었듯이 동시 요청이 동일한 서버나 웹 엔진에게 요청이 전달될 경우에는 특별한 문제가 발생되지 않는다. 하지만 애플리케이션 구성에 의해서 다른 서버로 요청이 동시에 전달될 경우에는 위와 같은 상황이 발생한다. 이를 막기 위해서 JEUS 9는 Copy&Update 방식으로 동작한다.



Session

동작 경로

동시 요청이 다른 서버로 가더라도 타겟이 되는 세션을 마이그레이션 하지 않고 카피하여 로컬에서 조작하고, 또한 그 결과를 다시 기존의 서버에게 업데이트 하는 방식을 수행한다. 이러한 동작으로 인해 세션의 유실은 방지할 수 있으며, 단순히 세션의 값을 확인하는 동작에서는 무리없이 서비스가 가능하다.



세션은 단순한 정보를 저장하고 관리하는 임시적인 객체이지 모든 시스템에 동시에 존재하는

데이터 베이스와 같이 사용해서는 안된다. 단순히 객체의 값을 참조하는 경우에만 사용하며, 다른 서버로의 동시 요청이 전달되는 구조 및 설정은 지양해야 한다.

2.5. 주요 기능

본 절에서는 분산 세션 서버의 주요 기능에 대해서 설명한다.

2.5.1. 중복 로그인 방지 기능

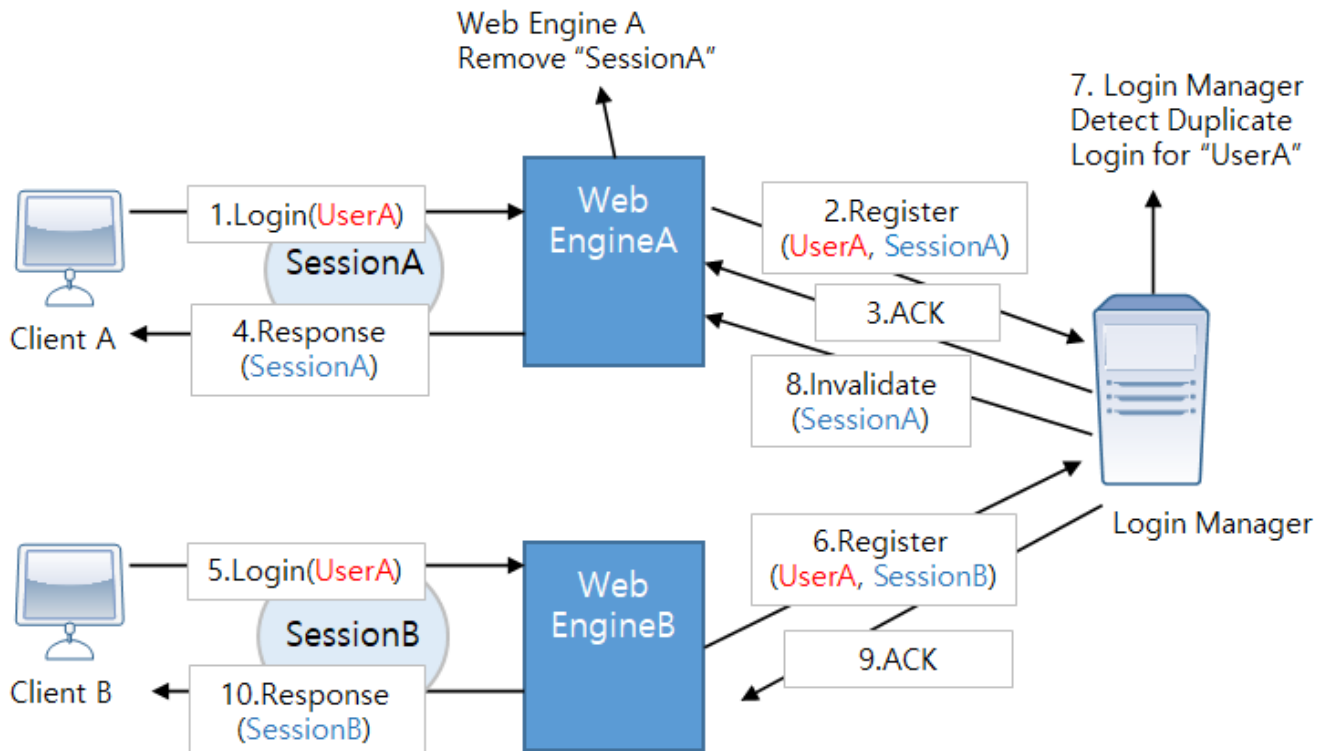
중복 로그인 방지 기능은 기본적으로 애플리케이션에서 관리 하는 중복 로그인 관리 기능 중 기본적인 기능을 분산식 세션 서버에서 제공하는 서비스이다.

분산식 세션 서버의 중복 로그인 방지 기능을 사용하게 되면 애플리케이션의 아이디에 대해 중복된 로그인을 허용하지 않음을 내부적으로 지원한다. 즉, 다른 세션에 대해 동일한 아이디로 로그인을 수행한 경우 기존의 로그인을 수행하였던 세션을 제거하여 근본적으로 중복 로그인을 방지한다.

중복 로그인 방지 기능을 위해 Login Manager는 기존의 중앙 세션 서버와 유사하게 구성된다. 여러 서버들 간의 클러스터링 안에서 로그인 정보를 저장하는 서버를 지정하여 동작하며 장애 상황을 고려하여 secondary 서버를 구성하여 동작한다.

중복 로그인 방지 기능은 Session Storage 별로 설정 가능하며 로그인 정보를 Session Storage로 관리된다. 해당 설정은 Session Storage의 property에 login-manager를 true로 설정한다.

중복 로그인 방지에 대한 동작은 다음과 같다.



중복 로그인 방지 기능의 동작 방식

다음은 중복 로그인 방지 기능의 동작 방식의 동작에 대한 설명이다.

1. SessionA를 사용 중인 Client A는 Web EngineA에 ID:UserA로 로그인을 시도한다.
2. Web EngineA는 Login Manager에 로그인 정보를 등록한다.
3. Login Manager는 로그인에 대한 ACK를 Web EngineA에 전달한다.
4. Web EngineA는 요청에 대한 Response를 Client A로 전달한다.
5. SessionB를 사용 중인 서로 다른 Client B가 동일한 ID:UserA로 로그인을 시도한다.
6. Web EngineB는 Login Manager에 로그인 정보를 등록한다.
7. Login Manager는 동일한 ID:UserA에 대해 중복 로그인이 일어났음을 알게 된다.
8. Login Manager는 기존의 로그인을 수행한 SessionA를 지닌 Web EngineA에 SessionA를 제거하라는 명령을 보내며, Web EngineA는 해당 세션을 제거한다.
9. Login Manager는 로그인에 대한 ACK를 Web EngineB에 전달한다.
10. Web EngineB는 요청에 대한 Response를 Client B에 전달한다.



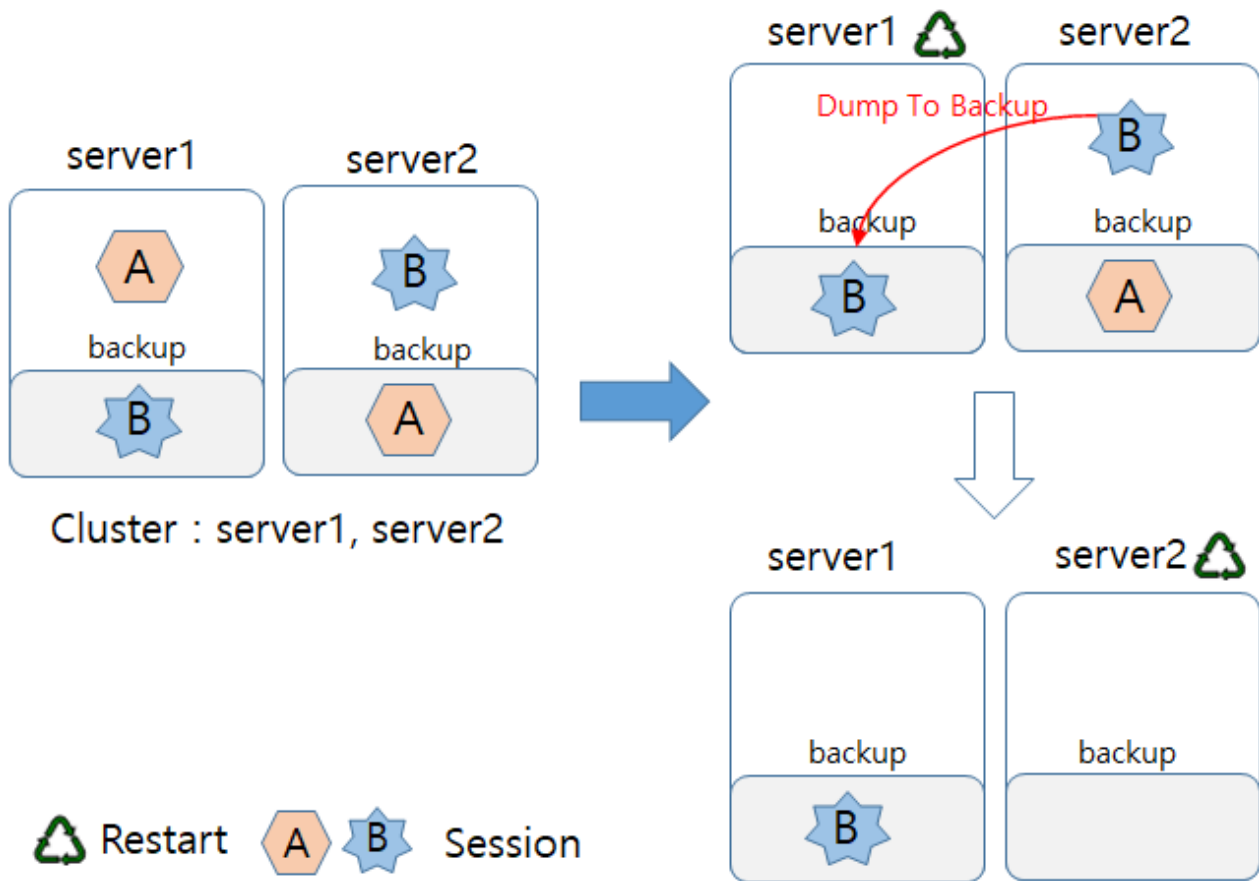
로그인은 애플리케이션에서 이벤트이므로 Jeus Login Manager에서 로그인 이벤트를 전달하는 방법에는 제약 사항이 존재한다. Jeus Login Manager에서는 기본적으로 Session Storage의 property에 user-info로 LoginID를 가지고 올 수 있는 Method를 설정한다. 동일한 키로 removeAttribute를 수행한 경우 로그아웃되며 invalidate된 세션은 자동으로 로그아웃된다.

2.5.2. Failback 기능

Failover 기능은 장애 상황에 대처 하여 기존의 서비스를 지속적으로 이뤄지도록 해주는 세션 서버의 기본적인 기능이다. 즉, 세션의 생성 이후 백업에 전달된 세션으로 인해 특정 서버가 장애가 발생하여도 세션의 유실이 발생하지 않으며, 기존 세션으로 지속적인 서비스를 하도록 하는 기능이다.

새롭게 제공하는 Failback 기능은 위의 1차적인 장애 상황에 대한 고려가 아닌, 연속적인 서버의 재시작 또는 장애 서버의 복구시에 제공되는 서비스이다. 해당 부분은 단순한 예로 순차적으로 서버를 재기동 하는 시나리오로 설명한다.

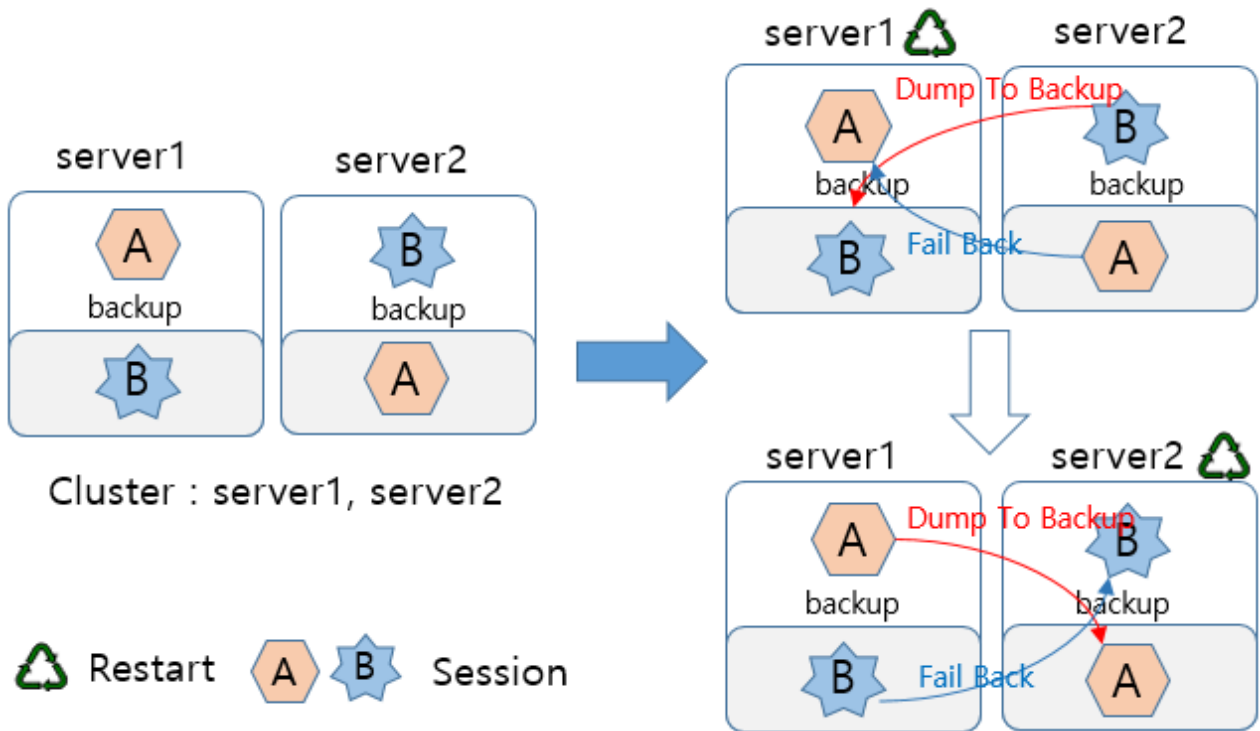
다음 그림은 Failback를 지원하지 않는 상태에서의 진행 상황에 대한 설명이다.



Failback을 지원하지 않는 경우

처음 Server1이 재기동되면 Server1이 가지고 있던 세션 "A"는 Server2의 백업에만 존재하게 된다. 다음으로 Server2가 재기동되면 Server2의 백업이 제거되기 때문에 세션 "A"는 전체 시스템에서 제거되게 된다. 만약 이후 세션 "A"의 요청이 온다면 세션 유지에 실패하게 되고 새로운 세션이 생성된다. 이는 기존의 분산식 세션 서버에서는 백업으로 받은 세션에 대한 처리를 수행하지 않았기 때문이다.

다음은 Failback를 지원하는 경우 결과이다.



Failback을 지원하는 경우

처음 Server1이 재기동되었을 경우 DumpToBackup에 의한 동작은 동일하다(세션 "B"가 유지되는 모습). 이후 추가적으로 Failback 기능에 의해 Server2가 지닌 세션 "A"가 다시 Server1로 전달되는 모습을 확인할 수 있다.

다음으로 Server2가 재기동되었을 경우에도 최초 환경과 동일하게 세션들이 유지되는 모습을 확인할 수 있다.



Failback을 지원하지 않더라도, 서버 장애가 발생하였을 때 특정 세션에 대한 액세스가 일어난다면 해당 세션은 정상적으로 유지 된다(Failover 기능). 이는 백업 세션의 존재로 가능하며, 이렇게 액세스된 세션은 다시 백업으로 전달되기 때문에(dumpToBackup) 추후에도 정상 동작이 가능해진다. 이를 위해 특정 웹 서버에서는 기존의 모든 세션에 대한 액세스를 수행하여 세션 유지를 위해 노력하기도 하였다.

Failback 기능은 기본적으로 설정에 의해 reply-response 방식으로 이뤄지게 된다.

기능의 시작은 자신이 백업 서버로 동작하였던 서버의 재기동을 감지함으로써 시작된다.

1. 특정 서버가 재기동되었으며 해당 서버의 세션을 자신이 백업 받았는지 파악한다.
2. Fail-back 설정이 true이면 기존에 서버로부터 받은 백업 세션들을 재기동된 서버에게 전송한다.



OOM으로 인한 장애 현상이 있을 경우 해당 기능은 권장하지 않는다. 세션의 과부하로 인해 많은 세션을 백업하였을 경우, Failback에 의해 지속적인 OOM을 유발할 수 있다. 해당 기능은 기존 세션의 끊임없이 서버를 순차적으로 재기동할 경우 롤링 패치 적용의 경우를 위해 제공된다.

2.6. 세션 클러스터 모드

본 절에서는 세션 클러스터 모드에 대해 설명한다.

세션 클러스터링은 단순히 생각하면 세션의 공유를 지원하는 기능이라고 할 수 있다. 세션의 공유는 동일한 스코일 경우 지원하는 것이 중요하며, 스코프가 다를 경우 공유를 막는 것도 중요하다.

JEUS에는 3가지 종류의 세션 클러스터 모드를 지원하며 각 모드에 의해 스코프가 결정된다.

- [기본 세션 클러스터 모드](#)
- [도메인 와이드 세션 클러스터 모드](#)
- [세션 스토리지 스코프 클러스터 모드](#)

2.6.1. 기본 세션 클러스터 모드

JEUS에서 기본적으로 제공하는 세션 클러스터 모드이다.

서버들을 클러스터링 설정하게 되면 자동적으로 제공되며, 세션 데이터를 클러스터링에 포함된 서버들에서 공유하여 사용할 수 있다. 서블릿의 기본 세션 스코프의 스펙을 준수하기 때문에 여러 개의 애플리케이션이 존재하더라도 서로간 독립적으로 세션관리가 가능하며, 또한 세션 클러스터링의 기본적인 기능들인 장애 상황 극복 및 로드 밸런싱의 처리를 지원한다. 스펙을 준수하여 기본적인 세션 스코프는 애플리케이션이지만, 웹 엔진 세션 매니저에서 제공하는 Session Storage의 Scope 설정을 이용하면 클러스터링 맷은 서버 스코프의 세션 클러스터링이 가능하다.

2.6.2. 도메인 와이드 세션 클러스터 모드

도메인 전체의 서버에 디플로이되는 모든 애플리케이션을 같은 스코프로 간주하는 세션 클러스터를 지원하는 모드이다.

기본 세션 클러스터 모드의 경우는 클러스터에 의존하기에 해당 클러스터에 의한 구조의 제약사항을 받고 있다. 그 중 서버 클러스터로의 디플로이 대상의 강제라고 볼 수 있다. JEUS 6에서는 각각 컨테이너에 별도의 애플리케이션을 디플로이하고 모든 컨테이너의 세션을 공유하여 사용할 수 있었으며 해당 구조를 직관적인 구조에 의해 많이 사용되었다.

다음은 좀더 자세한 예에 대한 설명이다.

- 2개의 노드가 존재하고 각각 2개의 컨테이너를 생성한다.
- 각 컨테이너에는 별도의 애플리케이션을 디플로이한다. 즉, 총 4개의 애플리케이션이 별도의 컨테이너에 존재한다.
- 결과적으로 4개의 각 다른 컨테이너의 모든 세션은 공유된다.

서버 클러스터의 기본 개념과 맞지 않다. 클러스터링을 맷을 경우 디플로이 대상이 클러스터로 강제되기 때문에 모든 서버에 동일한 애플리케이션이 디플로이된다. 정상적 상황 속에 안정적인 서버에서 이러한 강제된 디플로이는 불필요한 리소스 낭비라고 보는 시선이 존재한다. 하지만 클러스터링을 맷었음에도 디플로이된 애플리케이션이 물리적으로 특정 서버에만 존재하여 장애 상황에 서비스를 정상적으로 하지 못하는 것을 막아주는 올바른 방향이다. 이러한 상황속에서도 물리적으로 서버마다 다른 애플리케이션을 디플로이하기 위해서는 클러스터에 참여해서는 불가능하다.

서버 클러스터에 참여하지 않고 세션 클러스터링을 사용하고자 할 때 이 모드를 선택하면 된다.

DOMAIN_WIDE 모드는 도메인 구조의 자체 스펙에 어긋나는 사용자 편의를 위한 설정이며 가장 넓게 또한 편리하게 사용할 수 있지만 제약사항에 유의해야 한다.

- EJB와는 관계가 없고, 오직 웹 애플리케이션에만 영향을 미친다.
- 클러스터링 설정과 동시에 설정할 경우 클러스터의 설정은 무시될 수 있다.
- 설정할 경우 도메인 내(domain.xml)의 모든 서버에 대해 적용된다. 일부 서버만을 대상으로 설정하는 것은 불가능하다.



서버 클러스터링을 맺었더라도 도메인 와이드 모드를 사용하면 도메인 와이드 세션 클러스터 모드가 동작한다.

2.6.3. 세션 스토리지 스코프 클러스터 모드

스토리지에 포함된 애플리케이션이나 클러스터끼리 같은 스코프로 간주하는 세션 클러스터 모드이다.

세션 스토리지 스코프 클러스터 모드는 기본적인 클러스터링 구성과 관계없이 특정 애플리케이션이나 클러스터를 그룹화 하여 세션을 공유하며, 그룹이 여러개 존재할 경우 사용하는 세션 클러스터링이다. 대부분의 경우는 공유하는 그룹이 1개이기 때문에 위의 기본적인 스코프나 도메인 스코프의 모드 사용으로 대부분 해결이 가능하다. 지원되는 세션 클러스터링 자체는 스코프만 차이가 있을 뿐 내부적 동작은 중앙식/분산식 세션 서버와 동일하다. 세션 스토리지 스코프 기능으로 클러스터 간의 세션 공유도 가능하며, 한 서버 안에서 여러 개의 스코프를 구성하는 것이 가능하다. 기존의 특별 정의의 스코프 세션 클러스터를 대신하여 사용 할 수 있다.

2.7. 세션 서버 설정

본 절에서는 세션 클러스터링의 설정 중 기본 설정에 대해 설명한다.

세션 클러스터링은 단순히 생각하면 세션의 공유를 지원하는 기능이라고 할 수 있다. 세션 공유 지원을 위해서 웹 엔진에는 세션 서버가 운영되며 해당 세션 서버가 위에서 설명한 여러 가지 기능에 대한 동작을 수행한다. 해당 동작에 대한 설정을 세션 서버 설정이라고 한다. 해당 세션 서버의 운영에 대한 설정 외에 도메인 전체적으로 운영되는 기본 설정을 한다.

세션 서버에 대한 설정은 domain.xml과 console을 통해 설정할 수 있다.

```
<session-server>
  <cluster-mode>DEFAULT</cluster-mode>
  <session-storage>
    <name>jeus-session-storage</name>
    <session-manager-provider>JEUS</session-manager-provider>
    <scope>
      <name>jeus-distributed-scope</name>
      <jeus-session>DISTRIBUTED</jeus-session>
    </scope>
    <session-config>
      <timeout>30</timeout>
      <max-session-count>-1</max-session-count>
      <reload-persistent>false</reload-persistent>
    </session-config>
  </session-storage>
</session-server>
```

```

        <tracking-mode>
            <cookie>true</cookie>
            <url>false</url>
            <ssl>false</ssl>
        </tracking-mode>
        <session-cookie>
            <cookie-name>JSESSIONID</cookie-name>
            <url-cookie-name>jsessionid</url-cookie-name>
            <version>0</version>
            <max-age>-1</max-age>
            <path>/</path>
            <secure>false</secure>
            <http-only>true</http-only>
            <same-site>Disable</same-site>
            <partitioned>false</partitioned>
        </session-cookie>
    </session-config>
    <target-cluster>cluster1</target-cluster>
</scope>
<scope>
    <name>jeus-central-scope</name>
    <jeus-session>CENTRAL</jeus-session>
    <session-config>
        <timeout>30</timeout>
        <max-session-count>-1</max-session-count>
        <reload-persistent>false</reload-persistent>
        <tracking-mode>
            <cookie>true</cookie>
            <url>false</url>
            <ssl>false</ssl>
        </tracking-mode>
        <session-cookie>
            <cookie-name>JSESSIONID</cookie-name>
            <url-cookie-name>jsessionid</url-cookie-name>
            <version>0</version>
            <max-age>-1</max-age>
            <path>/</path>
            <secure>false</secure>
            <http-only>true</http-only>
            <same-site>Disable</same-site>
            <partitioned>false</partitioned>
        </session-cookie>
    </session-config>
    <target-application>sessionTest.war</target-application>
</scope>
</session-storage>
<property>
    <key>encoding-rule</key>
    <value>BASE64</value>
</property>
<jeus-login-manager>
    <login-manager-type>REDIS</login-manager-type>
    <primary>server1</primary>
    <secondary>server2</secondary>
    <property>
        <key>redis-nodes</key>
        <value>redis://localhost:6379</value>
    </property>
</jeus-login-manager>

```

```
<jeus-central-session-server>
  <primary>server1</primary>
  <secondary>server2</secondary>
</jeus-central-session-server>
</session-server>
```

• 기본 설정

다음은 도메인 전체에 적용되는 기본 설정 부분이다. **domain.xml**을 통해 항목 수정이 가능하다.

항목	설명
cluster-mode	<ul style="list-style-type: none"> • DEFAULT : 서버 클러스터를 맺을 경우 세션 클러스터를 지원하고 서버 클러스터를 맺지 않을 경우 세션 클러스터를 지원하지 않는다. 애플리케이션 단위의 세션 클러스터가 기본적으로 제공된다. (기본값) • DOMAIN_WIDE : 전체 도메인의 모든 서버의 모든 애플리케이션의 세션을 공유하는 세션 클러스터를 지원한다. <p>각 세션 클러스터링 모드는 스코프만 다를 뿐 동일한 형식의 분산식 세션 서버의 설정을 사용한다. 세션 스토리지 관련 설정하는 방법은 아래에서 설명한다.</p>
property	<p>property에 들어갈 수 있는 key들은 2가지가 있다.</p> <ul style="list-style-type: none"> • excluded-servers : 클러스터에 포함되지 않을 서버의 이름을 나열한다. 구분자는 콤마(,)로 한다. • encoding-rule : 스티키 라우팅의 대상이 되는 정보에 대해 인코딩 룰을 지원한다. 현재는 세 가지의 방식을 지원한다. <ul style="list-style-type: none"> ◦ BASE64 : 설정의 기본값으로 스티키되는 정보를 BASE64 룰에 의해 인코딩하여 전달된다. 세션 아이디를 통해 엔진정보나 도메인 정보가 노출되는 것을 원치 않아 직관적으로 의미 없는 정보로 보이도록 인코딩하여 전달한다. (기본값) ◦ BASE64_WITHOUT_PADDING : 설정의 기본값으로 스티키되는 정보를 BASE64 룰에 의해 인코딩 후 PADDING("=") 없이 전달된다. ◦ RAW : 스티키되는 정보를 인코딩하지 않고 그대로 노출하여 전달한다. 인코딩되는 정보는 내부에서도 직관적으로 어떤 엔진에서의 요청인지에 대한 판단이 어렵기에 디버깅을 위해 설정하거나, 엔진 이름 노출이 보안상 영향을 주지 않을 경우에 설정한다. <p>property는 console을 통해서도 설정이 가능하다. JEUS Reference 안내서의 "set-sessionserver-property"를 참고한다.</p>

• Session Storage 영역

domain.xml 또는 **console 명령어**를 이용하여 **추가/수정/삭제**가 가능하다.

JEUS Reference 안내서의 "세션 관련 명령어"를 참고한다.

항목	설명
Name	스토리지의 이름이다.

항목	설명
Session Manager Provider	세션 매니저로 사용할 구현체를 설정하거나 예약어를 설정한다. 예약어 외에는 package 전체 이름을 설정해야 한다. 설정 가능한 예약어는 4가지가 존재한다. 해당 예약어에 대한 설명은 표 아래의 [예약어] 설명을 참고한다.
Property	해당 스토리지에서 사용 할 프로퍼티를 설정한다. Property에서 유효한 키는 표 아래의 [Property 유효한 키] 설명을 참고한다.

[예약어]

다음은 Session Storage 설정의 'Session Manager Provider' 항목에 설정 가능한 예약어에 대한 설명이다.

• JEUS

JEUS에서 제공하는 다른 구현체가 존재하더라도 JEUS의 구현체를 사용한다. (기본값)

• REDIS

Session Storage로써 Redis를 사용한다. Redis는 5버전 이상을 지원한다. Redis에 대한 설정은 session-storage의 property에 입력한다.

다음은 유효한 키에 대한 설명이다.

- Redis의 아키텍처를 입력한다. 아키텍처에는 standalone과 cluster가 있다. (기본값: standalone)

예시

```
<key>redis-architecture</key>
<value>standalone</value>
```

- Redis의 노드를 입력한다. 아키텍처가 클러스터인 경우 콤마(,)로 클러스터 멤버를 기재한다. (기본값: redis://localhost:6379)

예시

```
<key>redis-nodes</key>
<value>redis://127.0.0.1:7006,redis://127.0.0.1:7007,redis://127.0.0.1:7008</value>
```

• HAZELCAST

Session Storage로써 Hazelcast를 사용한다. Hazelcast는 4.2 버전을 지원한다. Hazelcast에 대한 설정은 session-storage의 property에 입력한다.

다음은 유효한 키에 대한 설명이다.

- Hazelcast의 아키텍처를 입력한다. 아키텍처에는 standalone과 cluster가 있다. (기본값: standalone)

예시

```
<key>hazelcast-architecture</key>
<value>standalone</value>
```

- Hazelcast의 노드를 입력한다. (기본값: localhost:5701)

예시

```
<key>hazelcast-nodes</key>
<value>localhost:5702</value>
```

- Hazelcast의 cluster를 입력한다. (기본값: dev)

예시

```
<key>hazelcast-cluster</key>
<value>dev</value>
```

• RUNTIME

JEUS에서 제공하는 프로바이더 외 다른 구현체가 존재한다면 해당 구현체를 사용하도록 설정한다. 다른 구현체가 존재하지 않는다면 JEUS에서 제공하는 세가지 중 하나로 선택된다.

[Property 유효한 키]

다음은 Session Storage 설정의 '**Property**' 항목에 대한 추가 설명이다.

• Provider가 JEUS일 때

항목	설명
reserved-thread-num	분산식 세션 서버로 들어온 요청을 처리하기 위한 Thread Pool에 대해 부가적인 설정이다. 기본적으로 System Thread Pool을 사용하기 때문에 특별히 설정할 필요는 없다. 이 서비스를 위한 Thread를 미리 할당할 필요가 있을 경우에만 설정하기를 권장한다.
connection-timeout	웹 엔진에 존재하는 세션 서버 간의 소켓 커넥션을 생성할 때 적용되는 Timeout 값이다.
read-timeout	웹 엔진에 존재하는 세션 서버 간의 통신에 적용되는 Read Timeout 값이다. 데이터를 보낸 후 응답을 최대 이 시간만큼 기다린다.

항목	설명
backup-level	<p>사용된 세션을 리모트 웹 컨테이너 또는 로컬 파일 데이터베이스에 백업할 때 사용된 세션에 대한 기준을 설정한다.</p> <ul style="list-style-type: none"> ◦ access : 세션 Attribute 별로 update를 수행하며 해당 객체는 setAttribute/putValue/removeAttribute/removeValue/getAttribute/getValue 함수 호출한 특정 Attribute 들 만을 백업한다. (기본값) ◦ set : 해당 세션의 setAttribute/putValue/removeAttribute/removeValue 함수 호출이 발생한 경우에만 업데이트된 것으로 간주하여 해당 세션 객체를 백업한다. ◦ get : 해당 세션의 setAttribute/putValue/removeAttribute/removeValue/getAttribute/getValue 함수 호출이 발생한 경우에만 업데이트된 것으로 간주하여 해당 세션 객체를 백업한다. ◦ all : 사용된 세션은 모두 백업한다. 해당 세션 객체가 HttpServletRequest.getSession() API로 호출될 경우 업데이트된 것으로 간주하여 해당 세션 객체를 백업한다.
backup-unit-size	세션의 백업을 수행할 때의 단위에 대한 설정이다. 최대로 함께 보낼 갯수를 설정한다.
backup-queue-size	네트워크 지연으로 인해 백업이 정상적으로 전송되지 못할 경우 대기하는 큐 사이즈에 대한 설정이다. 분산식 세션 서버에서 백업이 존재하지 못할 경우 장애 상황에 세션 유지에 실패하기 때문에 해당 큐에 가득찰 경우 요청은 자연스럽게 지연되어 플로우 컨트롤된다.
backup-flowcontrol-enabled	백업 큐가 가득 차더라도 서비스를 지속적으로 수행하고자 할 때 설정하는 옵션으로 세션 유실보다 서비스가 우선시 될 경우에 사용된다.
failover-delay	<p>장애 상황에서 해당 엔진의 복구를 기다리는 시간을 설정한다.</p> <p>웹 엔진에 장애가 발생했을 때 해당 엔진을 제외한 나머지 엔진에서 다시 클러스터링 연결을 맺을 Timeout 값이다.</p>
restart-delay	웹 엔진을 정상적으로 다운시켰을 때 해당 엔진을 제외한 나머지 엔진에서 다시 클러스터링 연결을 맺을 Timeout 값이다. 다운되는 가장 많은 케이스인 재기동의 성능향상을 위한 설정이다.
user-info	user 정보를 가지고 올 수 있는 경로를 입력한다.
login-manager	<p>로그인 매니저의 사용여부를 선택한다.</p> <ul style="list-style-type: none"> ◦ false (기본값) ◦ true
login-manager-strategy	<p>로그인 매니저의 방법을 선택한다.</p> <ul style="list-style-type: none"> ◦ invalidate-before : 중복 로그인 시 이전 세션을 invalidate한다. (기본값) ◦ invalidate-after : 중복 로그인 시 나중 세션을 invalidate한다.

• Provider가 REDIS일 때

항목	설명
redis-nodes	Redis의 노드를 입력한다. 아키텍처가 클러스터인 경우 콤마(,)로 클러스터 멤버를 기재한다. (기본값: redis://localhost:6379)
redis-architecture	Redis의 아키텍처를 입력한다. 아키텍처에는 2가지가 있다. <ul style="list-style-type: none"> ◦ standalone (기본값) ◦ cluster

• Provider가 HAZELCAST일 때

항목	설명
hazelcast-nodes	Hazelcast의 노드를 입력한다. (기본값: localhost:5701)
hazelcast-cluster	Hazelcast의 cluster를 입력한다. (기본값: dev)
hazelcast-architecture	HAZELCAST의 아키텍처를 입력한다. 아키텍처에는 2가지가 있다. <ul style="list-style-type: none"> ◦ standalone (기본값) ◦ cluster

• Scope 영역

항목	설명
name	Scope의 이름이다.
jeus-session	Jeus session을 사용할 경우 분산식을 사용할 지, 중앙식을 사용할 지에 대한 설정이다. Storage의 provider가 Redis나 Hazelcast일 경우는 해당 설정은 무시된다. <ul style="list-style-type: none"> • DISTRIBUTED: 분산식 세션을 사용할 경우에 설정한다. • CENTRAL: 중앙식 세션을 사용할 경우에 설정한다.
target	scope에 포함될 applitaion 또는 cluster를 설정한다. (둘 중 하나만 설정) <ul style="list-style-type: none"> • target-application: 해당 스코프에 포함될 애플리케이션을 지정한다. • target-cluster: 해당 스코프에 포함될 클러스터를 지정한다.
session-config	해당 스코프에 포함된 컨텍스트에서 사용할 세션 설정이다. 상세 설정은 세션 설정 을 참고한다.

◦ jeus-central-session-server 영역

JEUS 자체 Session Server로 사용할 server를 설정한다. **domain.xml** 또는 **console 명령어**를 통해 항목 수정이 가능하다. JEUS Reference 안내서의 "set-jeus-central-session-server"를 참고한다.

항목	설명
primary	Primary Session Server를 설정한다. 동적 변경이 불가능하다.

항목	설명
secondary	Secondary Session Server를 설정한다. 동적 변경이 불가능하다.

◦ **jeus-login-manager** 영역

Jeus Login Manager를 사용할 경우 설정한다. **domain.xml**을 통해 항목 수정이 가능하다.

항목	설명
login-manager-type	<p>Login Manager로 사용할 구현체를 설정하거나 예약어를 설정한다.</p> <p>예약어는 다음과 같이 세 가지가 존재한다.</p> <ul style="list-style-type: none"> ◦ JEUS: JEUS에서 제공하는 다른 구현체가 존재하더라도 JEUS의 구현체를 사용한다. ◦ REDIS: Session Storage로써 Redis를 사용한다. Redis는 5버전 이상을 지원한다. Redis에 대한 설정은 property에 입력한다. ◦ HAZELCAST: Session Storage로써 Hazelcast를 사용한다. Hazelcast는 4.2버전을 지원한다. Hazelcast에 대한 설정은 property에 입력한다.
primary	JEUS Login Manager의 Primary 서버를 설정한다. 동적 변경이 불가능하다. 해당 설정은 ' Login Manager ' 항목이 'JEUS'일 때만 유효하다.
secondary	JEUS Login Manager의 Secondary 서버를 설정한다. 동적 변경이 불가능하다. 해당 설정은 ' Login Manager ' 항목이 'JEUS'일 때만 유효하다.

항목	설명
property	<p>JEUS Login Manager에서 사용할 프로퍼티를 설정한다.</p> <p>다음은 Property에서 유효한 키에 대한 설명이다.</p> <ul style="list-style-type: none"> ◦ 'Login Manager' 항목이 'REDIS'일 때 <ul style="list-style-type: none"> ◦ redis-nodes : Redis의 노드를 입력한다. 아키텍처가 클러스터인 경우 콤마 (,)로 클러스터 멤버를 기재한다. (기본값: redis://localhost:6379) <p>예시</p> <pre><key>redis-nodes</key> <value>redis://127.0.0.1:7006, redis://127.0.0.1:7007,redis://127.0.0.1:7008</value></pre> ◦ 'Login Manager' 항목이 'HAZELCAST'일 때 <ul style="list-style-type: none"> ◦ hazelcast-nodes : Hazelcast의 노드를 입력한다. (기본값: localhost:5701) <p>예시</p> <pre><key>hazelcast-nodes</key> <value>localhost:5702</value></pre> ◦ hazelcast-cluster : Hazelcast의 cluster를 입력한다. (기본값: dev) <p>예시</p> <pre><key>hazelcast-cluster</key> <value>dev</value></pre>