WebSphere. software

WebSphere eXtreme Scale v8.6

WXS 를 활용한 글로벌 캐시 구성 가이드

(2015. 01.)

IBM SWG WebSphere

IBM

0) IBM WebSphere eXtreme Scale(WXS) 을 활용한 글로벌 캐시란?

이전 강좌에서 IBM WXS 를 이용한 HTTP 세션 관리에 대해서 살펴봤었습니다. WXS 의 Grid 를 활용하여 HTTP 세션 데이터를 캐시 해두고 사용하는 방안을 보았는데 이번 강좌에서는 이와 비슷하게 Grid 에 원하는 Java 객체를 캐시해두고 여러 클라이언트(WAS)가 접속해서 해당 정보를 활용할 수 있는 방안(글로벌 캐시)에 대해서 간단하게 샘플을 활용하여 설명 드리도록 하겠습니다.

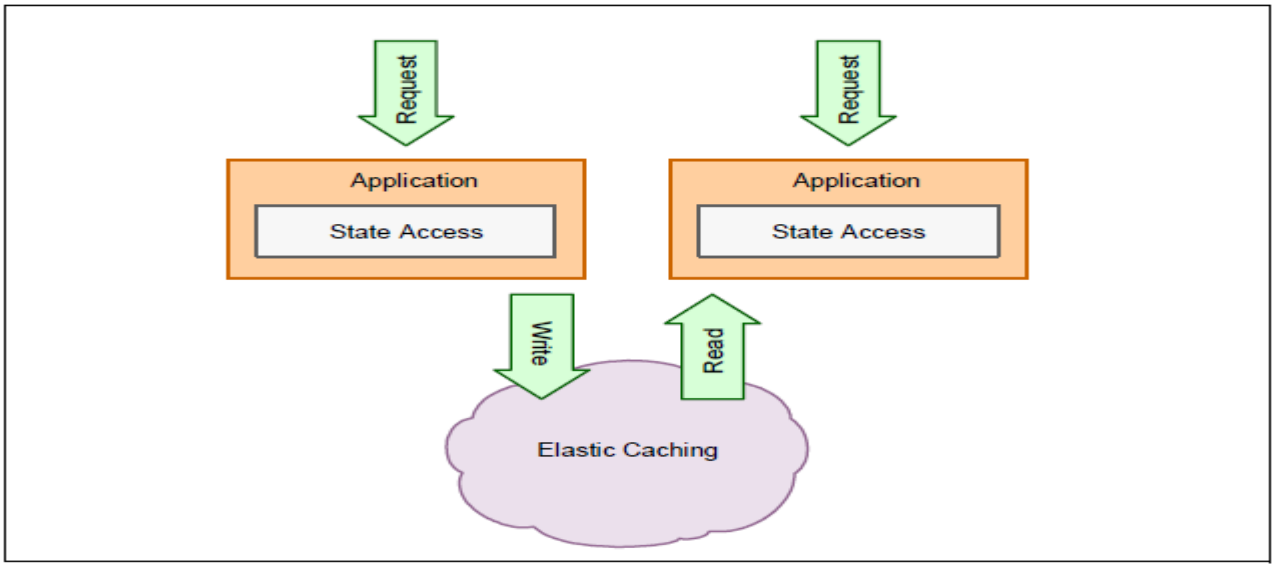
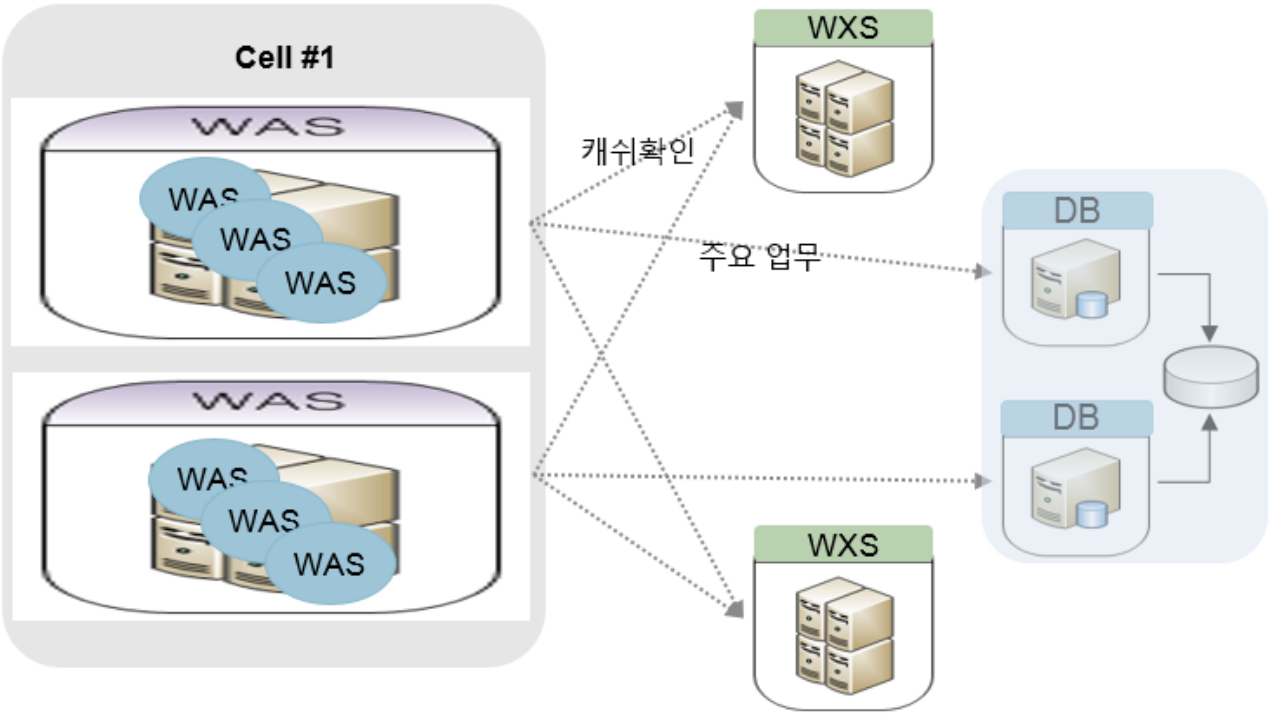


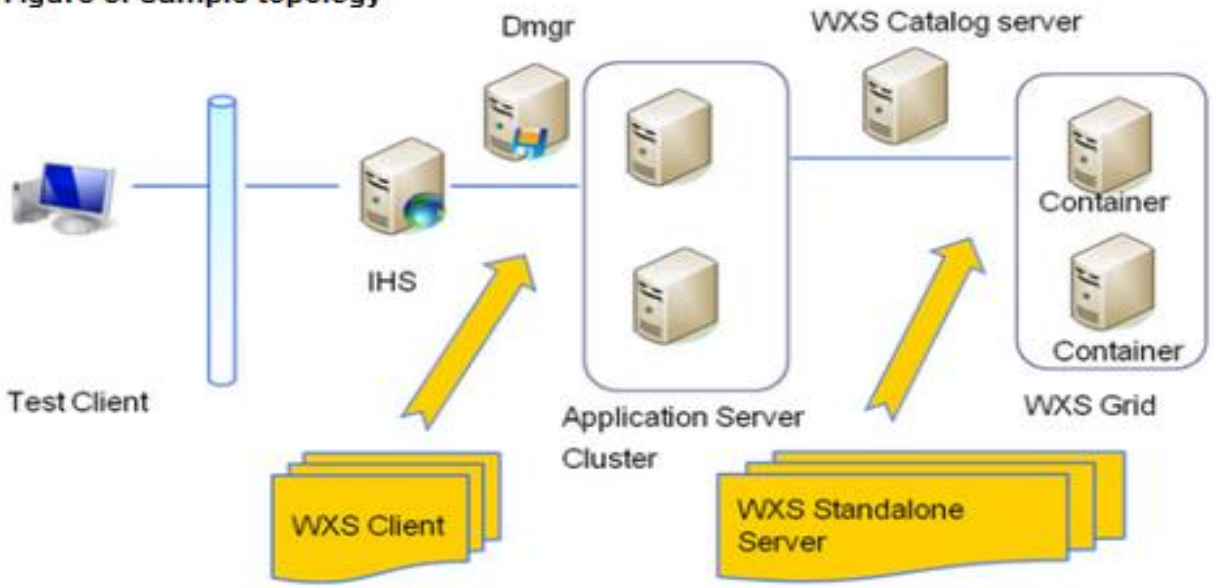
Figure 3-1 Application state store scenario



1) IBM WebSphere eXtreme Scale(WXS) 설치 및 기본 구성

설치 및 구성의 경우는 지난 강좌인 HTTP 세션 복제를 구성한 형태와 동일하게 서비스를 수행하는 WAS 에 client(jar 파일) 가 설치되어 WXS 는 standalone 형태의 Server 가 존재하는 아키텍처를 가정을 하고 진행하도록 하겠습니다.

Figure 5. Sample topology



2) IBM WebSphere eXtream Scale Server 구성

① IBM WXS 의 카탈로그 서버와 컨테이너 서버의 구성을 결정하는 설정 정보 파일 준비

일반적으로 설치되어 있는 [WXS 설치디렉토리]/ObjectGrid/wesb 디렉토리에 설정 샘플파일이 들어있으며 해당 샘플파일을(wesb_objectGrid.xml, wesb_objectGridDeployment.xml) 약간 변형해서 사용하는 형태로 가이드를 진행하도록 하겠습니다.

중요 부분 in wesb_objectGrid.xml

```
<objectGrids>
        <objectGrid name="Grid">
            <backingMap name="Customer" lockStrategy="PESSIMISTIC"
ttlEvictorType="CREATION_TIME" timeToLive="600"/>
        </objectGrid>
    </objectGrids>
```

샘플로 제공되는 objectGrid 설정 파일을 살펴보면 단순하게 Customer 라는 backingMap 을 하나 만드는 설정입니다. backingMap 은 실제 캐시된 객체가 저장되며 이전에 설명했던 Grid 내부의 파티션 안에 존재합니다. 그 뒤의 설정들을 계속 살펴보면 락 전략은 pessimistic 방식으로 수행하고 ttl Evictor 를 지정하는데 ttl Evictor 란 시간 기반으로 캐시를 버리는 형태의 모듈입니다. 설정을 읽어보면 바로 이해할 수 있는 것처럼 생성된 시간 이후로 600 초가 지나면 캐시가 무효하다고 보고 버리도록 설정된 것입니다.

변경 부분 in wesb_objectGrid2.xml

```
<objectGrids>
        <objectGrid name="Grid">
            <backingMap name="CacheData01" lockStrategy="PESSIMISTIC"
copyMode="COPY_TO_BYTES" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="600"/>
        </objectGrid>
    </objectGrids>
```

사용의 편의를 위해 이를 상단과 같이 변경합니다. 변경된 부분을 간단하게 설명드리면 이름을 CacheData01 로 변경했고 copyMode="COPY_TO_BYTES" 를 추가하여 eXtreme data format (XDF) 을 사용할수 있도록 하였습니다. XDF 는 IBM 이 만든 WXS 고유의 포멧으로 Java 뿐만이 아니라 C# 이나 .NET 에서도 사용될 수 있는 직렬화 포멧으로 성능 및 메모리 사용율이 우수한 포멧입니다. ttl Evictor 는 "LAST_ACCESS_TIME" 으로 변경하여 생성 시간 대신에 마지막 접속 시간을 기준으로 캐시된 데이터를 만료시키도록 설정을 변경하였습니다.

다음으로 Grid Container 배치관련 설정파일을 살펴보도록 하겠습니다.

중요 부분 in wesb_objectGridDeployment.xml

```
<objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0" maxSyncReplicas="1"
>
            <map ref="Customer"/>
        </mapSet>
    </objectgridDeployment>
```

샘플로 제공되는 objectGridDeployment 설정 파일을 살펴보면 objectgrid 이름은 Grid 이며 mapSet 이라는 이름의 mapSet 을 하나 만들고 파티션의 개수는 13개, 동기화 형태로 복제를 수행해서 가지고 있는 복제복의 최대 개수는 1 로 설정하였습니다. 이렇게 만든 mapSet 안에는 Customer 라는 map 이 하나 들어 있는 형태입니다..

중요 부분 in wesb_objectGridDeployment2.xml

```
<objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="1" >
            <map ref="CacheData01"/>
        </mapSet>
    </objectgridDeployment>
```

간단하게 테스하는게 목적이라 해당 설정의 파티션 개수를 줄이고 map 이름만 objectGrid 와 동일하게 변경하고 수정을 마칩니다.

나머지 설정은 지난 강좌와 동일하게 사용하도록 하며 여기까지 하셨다면 WXS Server 에 대한 준비가 완료된 것 입니다. 참고로 말씀드린 것처럼 나머지 부분은 지난 강좌와 동일하게 사용해도 되지만 간단한 테스트를 위해서 제가 테스트시에 간단하게 만들었던 시작 스크립트를 첨부드리오니 참고하시기 바라겠습니다.

카탈로그 서버

startXsServer cat1 -listenerPort 4809 -catalogServiceEndPoints cat1:kr050578:6600:6601

컨테이너 서버

startXsServer c1 -objectGridFile C:₩IBM₩WebSphere85₩eXtremeScale₩ObjectGrid₩wesb₩wesb_objectGrid2.xml -deploymentPolicyFile C:₩IBM₩WebSphere85₩eXtremeScale₩ObjectGrid₩wesb₩wesb_objectGridDeployment2.xml -catalogServiceEndPoints kr050578:4809

이를 이용해서 WXS 의 카탈로그 서버와 컨테이너 서버를 구동하면 하단과 같은 결과를 확인할 수 잇습니다. (참고로 복제본은 그리드 설정이 minSyncReplicas="0" maxSyncReplicas="1" 이기 때문에 컨테이너 서버를 하나 더 두게 되는 경우에 확인하실 수 있습니다.)

xscmd -c showMapSizes -cep kr050578:4809

```
C:\IBM\WebSphere85\eXtremeScale\ObjectGrid\bin>xscmd -c showMapSizes -cep kr0505
78:4809
시작 시간 :2015-01-07 17:14:21.387

CWXSI0068I: 명령 실행 중: showMapSizes

*** Grid 데이터 그리드 및 mapSet 맵 세트에 대한 결과 표시.

*** c1에 대한 맵 나열 ***
맵 이름     파티션 맵 항목 사용한 바이트 샤드 유형 컨테이너
-------     ------ ------- ------------ --------- --------
CacheData01 0      0       0            Primary   c1_C-1
CacheData01 1      0       0            Primary   c1_C-1
CacheData01 2      0       0            Primary   c1_C-1
CacheData01 3      0       0            Primary   c1_C-1
CacheData01 4      0       0            Primary   c1_C-1
서버 총계: 0 (0 B)

총 카탈로그 서비스 도메인 수: 0 (0 B)
(사용된 바이트 통계는 단순 오브젝트 또는 COPY_TO_BYTES 복사 모드를 사용 중인 경
우에만 정확합니다.)

CWXSI0040I: showMapSizes 명령이 완료되었습니다.
```

3) WebSphere eXtream Scale Client 애플리케이션 개발

당연히 WXS 를 활용한 글로벌 캐시에 접속하고 데이터를 input 하거나 get 하기 위해서는 WXS 에서 제공되는 API 를 사용해야 합니다. 이전 강좌처럼 WXS client 가 설치된 IBM WAS 제품이라면 별도의 라이브러리 추가나 설정이 필요하지 않습니다. 그러나 WXS client 가 설치되지 않은 경우라면 ogclient.jar 파일이나 wsogclient.jar 파일만 애플리케이션에서 참조하면 됩니다.

그럼 이러한 환경에서 간단한 샘플 애플리케이션을 개발해 보도록 하겠습니다. 우선은 WXS 에서 제공되는 API 를 활용하여 카탈로그 서버에 접속하는 코드를 하단과 같이 작성합니다. 참고적으로 해당 부분은 계속 재사용되기 때문에 싱글톤 형태로 구성하는 것도 하나의 좋은 방법입니다.

```
// Retrieve an ObjectGridManager instance.
ogm = ObjectGridManagerFactory.getObjectGridManager();

// Obtain a ClientClusterContext by connecting to a catalog
// service domain, manually suppling the catalog service endpoints,
// and optionally specifying the ClientSecurityConfiguration and
// client ObjectGrid override XML file URL.
String catalogServiceEndpoints = endpoint;

try {
    ccc = ogm.connect(catalogServiceEndpoints,(ClientSecurityConfiguration) null, (URL) null);
    // Obtain a distributed ObjectGrid using ObjectGridManager and providing
    // the ClientClusterContext.
    og = ogm.getObjectGrid(ccc, gridName);
    System.out.println("CatalogServer connection!");
} catch (ConnectException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

보시면 아시겠지만 ObjectGridManagerFactory 를 사용하여 ObjectGridManager 를 받아와서 연결을 수행하여 ObjectGrid 를 가지고 오는 형태로 간단하게 구성가능 합니다. (참고로 endpoint 는 카탈로그 서버 주소:포트 형태로 "kr050578:4809" 을 gridName 은 Gird 이름으로 "Grid" 를 입력합니다.)

다음으로 서블릿 페이지를 하나 만들어서 하단의 코드를 이용해서 WXS 캐시에 데이터를 input 하는 코드를 작성합니다.

```
PrintWriter out = response.getWriter();

try {
    // Get a session
    Session session = og.getSession();
    // Get the ObjectMap
    ObjectMap map1 = session.getMap(mapName);
    //transaction related with session begin
    session.begin();
    // Insert data into the map.This is implicitly transactional
    map1.insert(1, "TestTestTestTest");
    // Close the session so that it can be managed in a pool
    session.commit();
    out.println("Data Insert Completed.");
} catch (ObjectGridException e) {
    e.printStackTrace();
}
```

여기서는 GridObject 를 이용하여 session 을 맺고 ObjectMap 에 원하는 데이터를 insert 하는 형태로 진행됩니다. 하단의 Grid 의 내부 구조를 보시면 좀 더 이해에 도움이 됩니다.

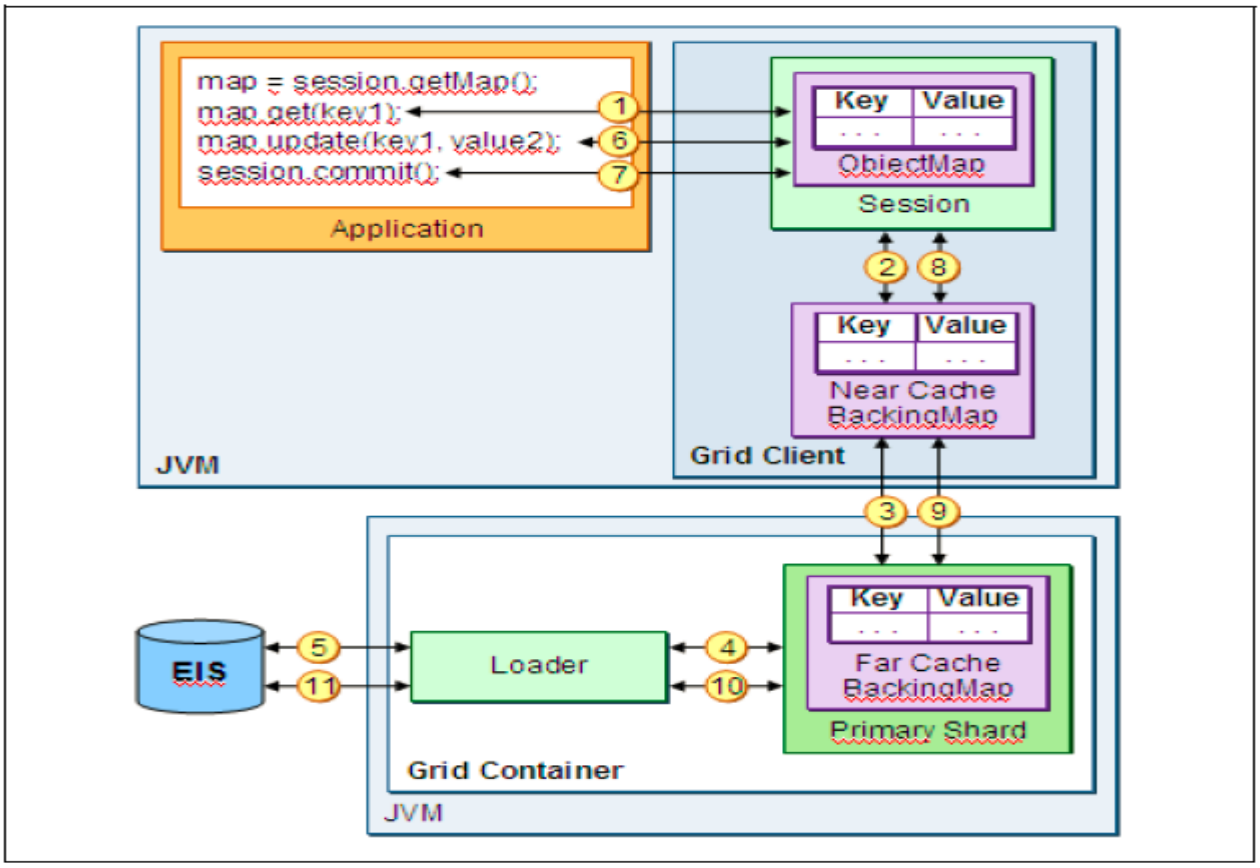


Figure 2-18 Component interactions for simple grid access

(참고로 클라이언트에서 보관하는 Near Cache 는 락 전략을 NONE이나 OPTIMISTIC 을 사용했을 때 같이 사용 가능해지며 현재 강좌에서는 락 전략을 PESSIMISTIC 을 사용하고 있기 때문에 설명은 생략하도록 하겠습니다.)

즉, ObjectMap 은 Grid 에 저장된 BackingMap 과 연결되어 실제로 가지고 오는 Map 객체이며 내부적으로 Key, Value 쌍을 가지고 있게 됩니다.

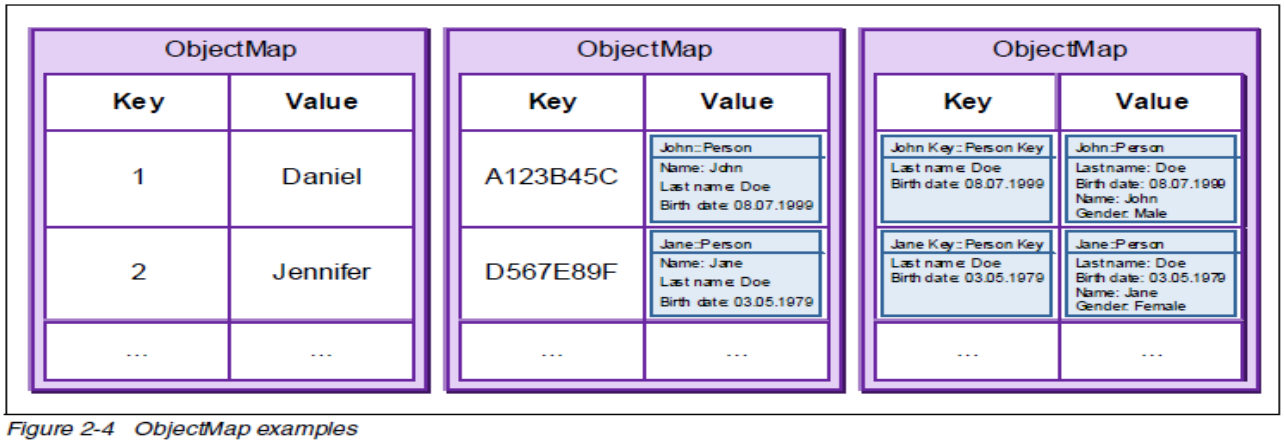


Figure 2-4 ObjectMap examples

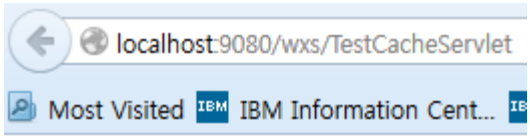
위의 작업과 동일하게 서블릿 페이지를 하나 더 만들어서 이번에는 WXS Grid 에 캐시된 정보를 가지고 오는 코드를 작성합니다.

```
PrintWriter out = response.getWriter();

try {
    // Get a session
    Session session = og.getSession();
    // Get the ObjectMap
    ObjectMap map1 = session.getMap(mapName);
    // Insert data into the map.This is implicitly transactional
    String data = (String)map1.get(1);
    // Close the session so that it can be managed in a pool
    session.close();
    out.println("Data get Completed : " + data);
} catch (ObjectGridException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

여기까지 하시면 단순하지만 WXS 로 구성된 글로벌 캐시에 데이터를 저장하고 가지고 오는 테스트 애플리케이션 작성이 완료된 것입니다.

4) 테스트

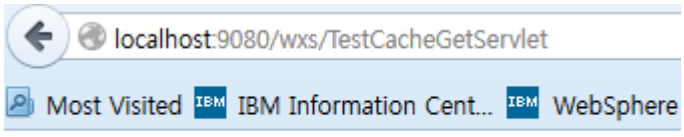
작성된 애플리케이션을 WAS 에 모두 배포한 후에 하단과 같이 WXS 캐시에 데이터를 input 하는 서블릿을 호출합니다.



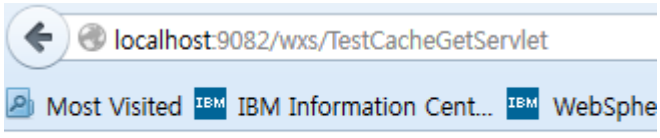
호출이 정상적으로 수행되면 하단과 같이 WXS Grid 컨테이너에 데이터가 들어온 것을 확인할 수 있습니다.

```
C:\IBM\WebSphere85\eXtremeScale\ObjectGrid\bin>xscmd -c showMapSizes -cep kr0505
78:4809
시작 시간 :2015-01-07 18:20:45.018

CWXSI0068I: 명령 실행 중: showMapSizes

*** Grid 데이터 그리드 및 mapSet 맵 세트에 대한 결과 표시.

*** c1에 대한 맵 나열 ***
맵 이름      파티션 맵 항목 사용한 바이트 샤드 유형 컨테이너
-------      ------ ------- -------------- --------- --------
CacheData01 0       0       0              Primary   c1_C-1
CacheData01 1       1       440 B          Primary   c1_C-1
CacheData01 2       0       0              Primary   c1_C-1
CacheData01 3       0       0              Primary   c1_C-1
CacheData01 4       0       0              Primary   c1_C-1
서버 총계: 1 <440 B>

총 카탈로그 서비스 도메인 수: 1 <440 B>
<사용된 바이트 통계는 단순 오브젝트 또는 COPY_TO_BYTES 복사 모드를 사용 중인 경
우에만 정확합니다.>

CWXSI0040I: showMapSizes 명령이 완료되었습니다.

종료 시간: 2015-01-07 18:20:48.574
```

다음으로 WXS 캐시에서 데이터를 가지고 오는 서블릿 페이지를 호출합니다.



정상적으로 캐시에 저장된 데이터를 화면으로 확인할 수 있습니다. 이와 마찬가지로 다른 WAS 서버로 해당 애플리케이션을 배포하여 데이터를 가지고 오는 서블릿 페이지를 호출해보면 글로벌 캐시 형태와 동일하게 이전에 1번 서버에서 저장한 데이터를 정상적으로 반환하는 것을 확인할 수 있습니다.



이를 좀더 쉽게 설명 드리면 최초로 언급한 것과 같은 형태의 글로벌 캐시 구성을 수행한 것이며 여러 클라이언트(WAS)가 공통된 글로벌 캐시에 접속하여 데이터를 작성하거나 읽는 작업을 수행할 수 있습니다.

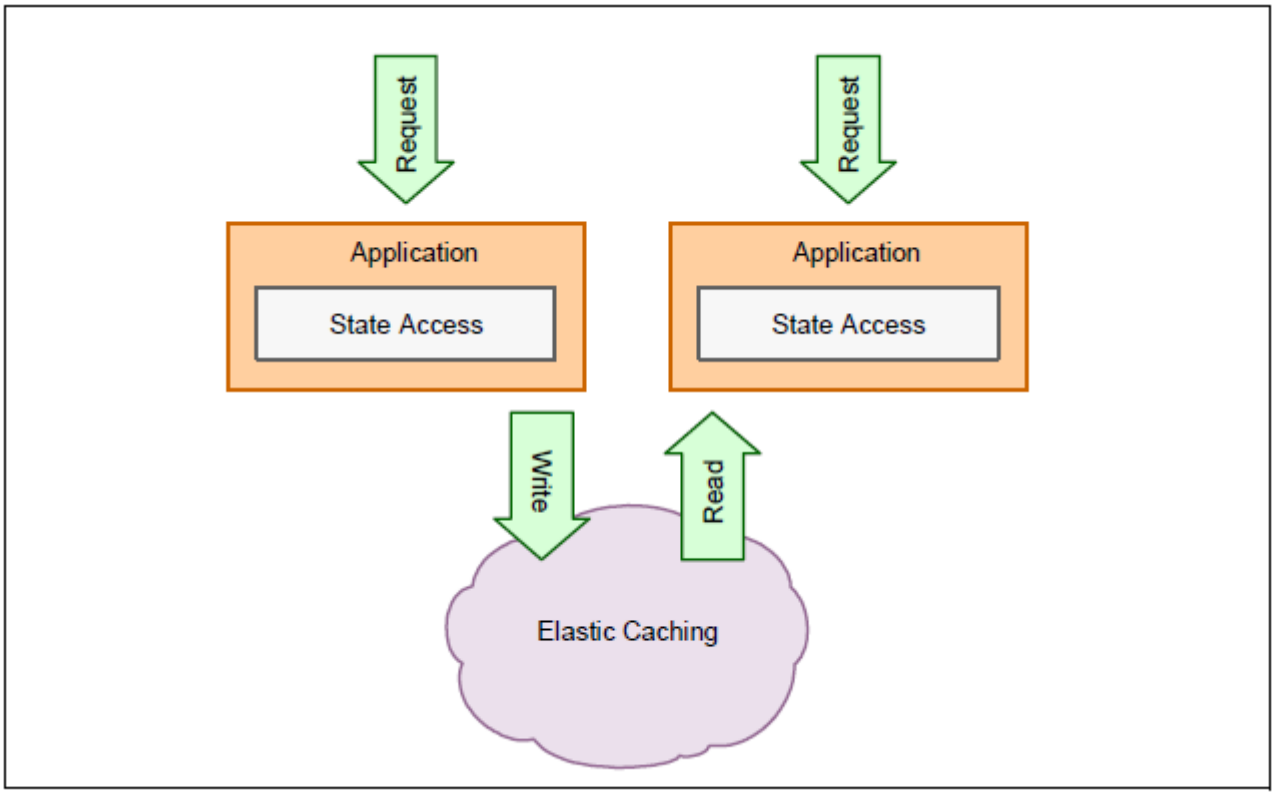
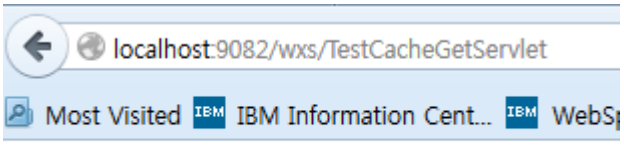


Figure 3-1 Application state store scenario

이렇게 하면 아주 간단하게 WXS 를 활용한 글로벌 캐시 구성이 완료된 것입니다.

마지막으로 아무 호출을 하지 않고 600 초를 기다린 후 조회 서블릿을 호출해보면 캐시된 데이터가 만료되어서 하단과 같이 null 을 반환하는 것을 확인할 수 있습니다.
(당연히 이전 objectGrid 설정에서 ttlEvictorType 를 NONE 으로 하면 제거되지 않습니다.)




```
Data get Completed : null
```

9) 참고 자료

IBM WebSphere eXtreme Scale Version 8.6 Information Center
http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fwelcome%2Fwelcome_xs.html