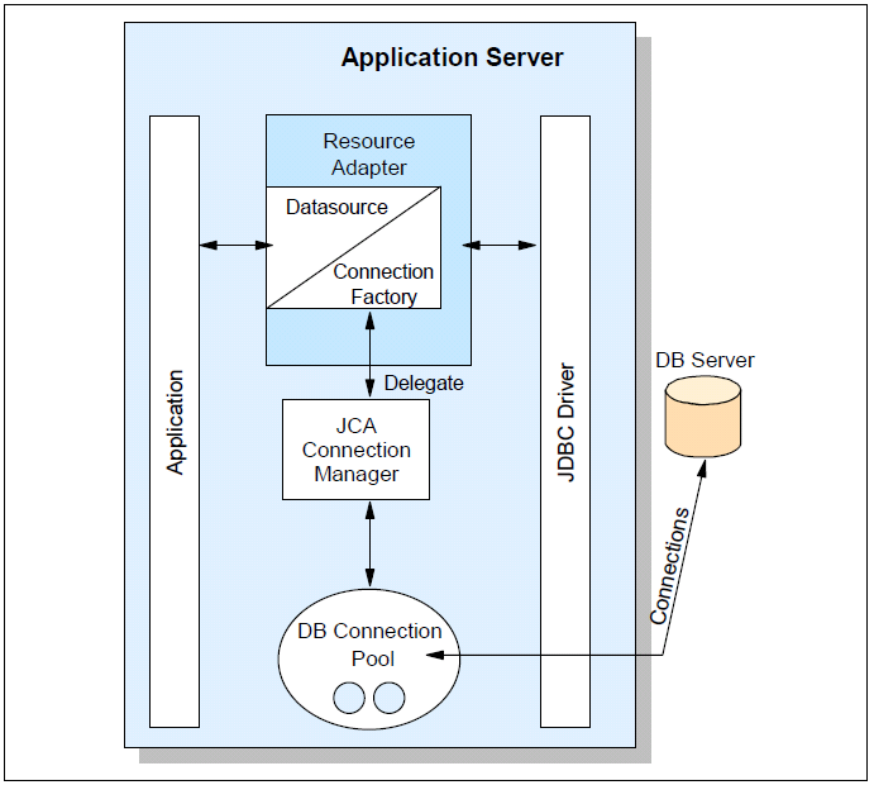
하나씩 쉽게 따라 해보는 IBM WebSphere Application Server(WAS) v7 – 23

이정운 (juwlee@kr.ibm.com)

하나씩 쉽게 따라 해보는 IBM WAS v7 스물 세 번째 이야기를 시작합니다. 스물 세 번째 이야기는 3부 강의의 두번째 로서 DB Connection 에 대해서 조금 더 자세한 이야기를 해보도록 하겠습니다.

이전 강좌에서는 DB Connection 을 받아오기 위하여 WAS에서 DB 자원을 지칭하는 JDBC provider 를 만들고, Data source 설정을 한 후, 이를 이용해 DB 와의 connection 을 받아온다라고 간단하게만 언급했는데 실제적으로는 하단의 구조와 같은 작업들이 이루어집니다.



참조: IBM WAS v7 Redbook

어떻게 이해가 되시나요? 사실적으로 그림만 보고는 바로 이해하기 어려울 수 있습니다. 따라서 이미 언급한 것 처럼 이것을 좀 더 분석해 보는 시간을 가져보도록 하겠습니다.

IBM WAS 에서 EIS(Enterprise Information System) 에 대한 Connection 관리 아키텍처는 Java EE 표준의 JCA 표준을 통해서 이루어집니다.(EIS 라니까 조금 어렵게 느껴지시는 분들이 있는데, 대부분 Database 를 많이 의미하니 그냥 그림에서 표현된 것처럼 DB 같은 자원 관리자라고 생각하셔도 무방합니다.) Connection 관리자(CM) 는 JCA 표준에서 정의된 자원어뎁터(Resource Adapter) 나 특정 벤더의 DB와 JDBC 연결을 제공하는 드라이버 구현 클래스를 제공하는 JDBC 프로바이더와 조합되는 Data source 를 통해서 얻어진 Connection 을 관리할 수 있습니다. 즉, WAS 서버 내에서는 실제적으로 CM 을 통해서 만들어진 Connection 을 관리하고 풀(pool)링, 로컬 트랜잭션,

보안 등을 지원합니다.

CM 에서 관리 가능한 데이터소스 connection 을 만들기 위해서는 JDBC data source 를 가능하게 하는 RA 를 제공하며, 이는 WAS 에서 JCA connection 을 관리하는 CM 과 동일한 녀석입니다. 다시 말씀 드려서 CM 의 입장에서는 JDBC 데이터소스와 JCA connection factory 가 동일하게 보입니다. 그러므로 여기서는 우선 간단하게 JDBC 데이터소스부분만 다루도록 하겠습니다. – 사실적으로 JDBC 데이터소스가 일반적 환경에서 가장 많이 사용되기 때문입니다..

Part 1. Connection 이해하기

위에서 간단하게 나마 Connection 을 실제 관리하기 위한 내부 아키텍처 그림을 통해서 사전 설명드렸지만 아직은 부족한 부분이 많습니다. 제가 언제나 강조하지만, Connection 을 이해하는 것도 백문이 불여일견 이라고 했으니 한번 간단한 예를 가지고 확인을 해보도록 하겠습니다.

EJB 의 경우를 예로 들어보면

1. EJB 가 data source, connection factory 에 대해서 JNDI 룩업(lookup) 을 수행하고 getConnection() 을 요청을 발행합니다.
2. Connection factory 는 요청을 CM 에게 위임합니다.
3. CM 은 WAS 내의 Connection pool 인스턴스(instance) 를 찾습니다. 만약 Connection pool 이 가용하지 않다면 CM은 ManagedConnectionFactory 를 사용하여 물리적이며 풀링 되지 않은 connection 을 생성합니다.

위와 같은 단계를 통해서 Connection 객체를 받아오며, 해당 Connection 객체를 이용해서 DB 와의 실제 연결 작업을 진행합니다. 그러나 언제나 사실은 저 먼 곳에 있다고 이를 좀 더 들여다보면, 실제로는 만들어진 Connection 객체를 받아오는 것이 아니라 Connection 핸들(handle)을 받아오는 것입니다.

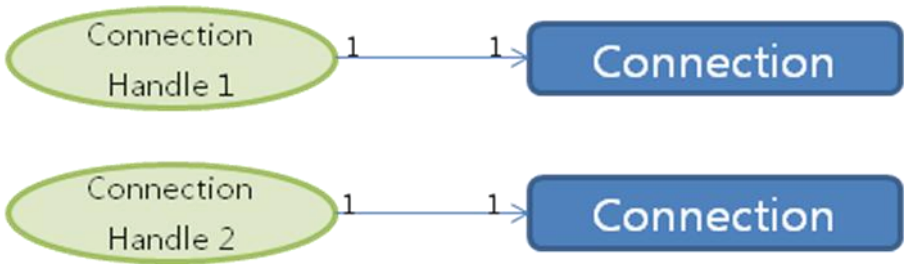
Connection 핸들은 J2EE 어플리케이션에 사용되며 뒷(back) 단의 자원 관리자(예: DB) 와 물리적인 connection 을 의미하는 것이 아니라 물리적인 connection 의 proxy 입니다. Proxy 라고 하시니 조금 어려울 수도 있을 텐데, 물리적인 connection 이 JDBC driver 에 의해서 반환 받은 connection 객체라면, connection 핸들은 물리적인 connection 의 랩퍼(wrapper) 객체로 생각하시면 됩니다. 내부 적으로 들여다 봐도 getConnecion() 메소드를 호출하면 connection 핸들을 반환 받게 되고 물리적 connection 은 CM 에 의해 별도 관리됩니다. 다시 강좌의 처음 부분에서 언급한 내용을 참고하시면 아시겠지만 WAS 에서는 CM 을 통해서 Connection 을 관리합니다. 그러므로 getConnection() 메소드를 통해서 Connection 을 받아온다고 하여도 실제적으로는 CM 에 의해서 Connection 은 관리되고 CM 에 의해 관리되는 Connection 객체의 핸들만을 받아와서 실제 프로그램에서 사용하는 것입니다.(중요한 점은 관리와 사용이 분리된다는 것 입니다.)

이해하기가 조금 어렵나요? 다시 말씀 드린다고 하여도 동일한 이야기인데, 물리적인 connection 은 CM 에 의해 관리되고 실제로 getConnecion() 을 하게 되면 connection 객체의 랩퍼 객체인 Connection 핸들을 받아서 사용하는 것입니다. 따라서, Connection 핸들을 제어하면 마치 Connection 객체를 가지고 있는 듯이 작업이 가능합니다. (다시 한번 강조하지만 중요한 것은 지금 어플리케이션에서 받아와서 사용하고 있는 Connection 핸들과 물리적인 Connection 이 실제적으로는 분리된 다른 것이라는 이야기입니다.)

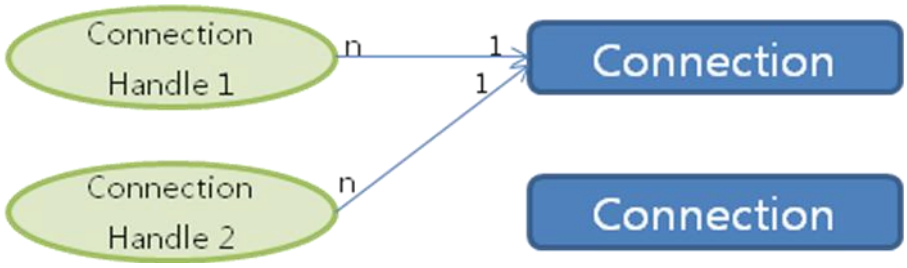
Connection 핸들을 이해했다면 한가지 또 알아두어야 할 사항이 있습니다. 그것은 바로 자원 공유를 위한 Shareable, Unshareable 모드입니다. Shareable 과 unshareable 로 지정 가능한 자원 공

유 범위지정은 J2EE 1.3 표준에서 소개되었으며 Connection 자원을 사용할 경우에 Connection 자원은 Shareable, Unshareable 이라는 두 가지 모드 중에 한가지로 표시됩니다.

Unshareable 로 표시된 connection 을 접근한다는 의미는 컴포넌트가 사용하는 connection 핸들과 핸들의 물리적 connection 이 일대일(one-to-one) 관계라는 것을 말합니다. 이를 좀 더 자세히 말하면 모든 getConnecion() 호출은 요청한 사용자가 독점하는 물리적인 connection 을 가진 connection 핸들을 반환합니다.



Shareable 로 표시된 자원은 높은 확장성을 제공할 수 있습니다. 모든 getConnection() 호출 마다 다른 물리적인 connection 을 반환하는 것 대신에 조건이 맞는다면 다수의 connection 핸들에 의해 하나의 물리적인 connection 은 공유 될 수 있습니다. (참고적으로 WAS 에서는 shareable 이 기본 값입니다.)



그렇다고 모든 경우에 shareable connection 을 사용할 수 있는 것은 아니고 global 트랜잭션일 경우에 그 범위에 한해서만 사용 가능합니다. (잘 생각해보시면 아시겠지만 로컬 트랜잭션일 경우에는 하나의 트랜잭션만 처리하므로 굳이 공유를 한다고 해서도 이점이 발생하는 경우가 적습니다.)

소스를 통해서 간단한 예를 살펴보면서 좀 더 이해해 볼 수 있는 시간을 가져보도록 하겠습니다.

```
//사용자 트랜잭션 시작
userTransaction.begin();
// 첫 번째 connection 가져오기
java.sql.Connection con1 = ds.getConnection();
java.sql.Statement stmt1 = conn1.createStatement();
…
// 두 번째 connection 가져오기
java.sql.conenction con2 = ds.getConnection();
java.sql.Statement stmt2 = conn2.createStatement();
…
// 사용자 트랜잭션 커밋(Commit)
userTransaction.commit();
```

위의 소스가 shareable 을 사용한다고 지정되어 있다면, 첫 번째 connection 을 가져온 후에 두 번째 connection 을 가져오는 것이 아니라 connection con1 과 con2 는 같은 데이터소스에서 획득한 같은 물리적 connection 을 가진 두 개의 connection 핸들을 사용합니다. 즉, 물리적인 connection 을 connection con1 과 con2 가 공유하는 것 입니다. 당연히 자원 활용률을 높일 수 있으며 connection 을 다시 만들기 위한, 불필요한 오버헤드를 줄일 수 있습니다.
(실제적으로 어플리케이션 소스의 배치 설명자(deployment descriptor)에 res-sharing-scope 를 shareable 로 설정함으로써 자원관리자(예:DB) 에 대한 connection 을 공유할 수 있습니다.))

Ejb-jar.xml 예제

```
<ejb-local-ref>
  <description>Local ref</description>
  <ejb-ref-name>ejb/OrderLocalHome</ejb-ref-name>
  <ejb-ref-type>Entity</ejb-ref-type>
  <local-home>market.order.OrderLocalHome</local-home>
  <local>market.order.OrderLocal</local>
  <ejb-link>OrderEJB</ejb-link>
</ejb-local-ref>
<resource-ref>
  <res-ref-name>jdbc/cmp/UserDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
  <res-id>OracleConnector1</res-id>
</resource-ref>
```

Part 2. Connection 생명주기(Life Cycle)

지난 파트에서는 Connection 에 대한 기본 구조에 대해서 알아봤습니다. 이번 파트부터는 Connection 생명주기에 대한 강좌를 진행하도록 하겠습니다 관리되는 Connection 객체도 결국은 객체이기 때문에 생명주기를 가지며 이를 잘 이해하고 활용해야 효율적인 connection 활용이 가능합니다.

생명주기에 따라서 Connection 객체는 *DoesNotExist* 나 *InFreePool*, *InUse* 세가지 상태를 가질 수 있습니다. Connection 이 생성되기 전에는 DoesNotExist 상태이며, connection 이 생성되면 InFreePool 이나 InUse 상태가 될 수 있습니다. Application 에 의해서 사용을 위해 할당이 되면 InUse 상태가 되며 사용되지 않는 상태거나 사용 후에 Connection 풀에 머무를 경우에는 InFreePool 상태가 됩니다.

그러나, 단순하게 지금 말한 것 처럼 Application 의 사용에 의한 할당에 따라 InUse, InFreePool 상태로 전이되는 것은 아니며 좀 더 까다로운 전이 조건들이 있습니다. 이러한 전이 조건들을 경계조건(guarding condition) 이라고 부르며 각각의 경계 조건은 하단의 표와 같습니다.

조건	설명
ageTimeoutExpired	Connection이 ageTimeout 설정 값보다 오래 살아있는 경우
close	어플리케이션이 connection 객체에 대해 close 메소드를 호출한 경우
fatalErrorNotification	치명적 에러통지를 받은 경우
freeConnectionAvailable	속성이 일치하는 가용한 connection 이 free pool 내에 있는 경우
getConnection	어플리케이션에 의해 getConnecion 메소드가 호출된 경우
markedStale	Connection 에 Stale 표시가 된 경우(일반적으로 치명적 에러 통지 때문에 표시)
noOtherReferences	Managed connection 에 대한 connection handle 이 하나만 있으며 트랜잭션 서비스 가 managed connection 에 대한 참조를 잡고 있지 않는 경우
noTx	시행중인 트랜잭션이 없을 경우
poolSizeGTMin	Connection pool 사이즈가 minimum pool 사이즈보다 클 경우
poolSizeLTMax	Pool 사이즈가 maximum pool 사이즈보다 작을 경우
shareableConnectionAvailable	Shareable connection 에 대한 getConnection() 요청이 있으며 공유 가능한 일치하는 속성을 가진 connection 이 있을 경우
TxEnds	트랜잭션이 종료된 경우
unshareableConnectionRequest	Unshareable connection 에 대한 getConnection() 요청이 있을 경우
unusedTimeoutExpired	Free pool 에 있는 connection 이 사용되지 않은 채 unused timeout 기간을 넘은 경우

경계 조건의 사용에 대한 간단한 예를 먼저 살펴보자면, InFreePool 에서 다음의 조건이 만족되어야지만 InUse 상태로 전이가 가능합니다.

InFreePool > InUse:
getConnection AND
freeConnectionAvailable AND
NOT(shareableConnectionAvailable)

위의 경우는 InFreePool 에서 InUse 상태로 전이되는 경계 조건을 이용해서 조건들을 표기화 한 것입니다. 이를 좀 더 자세히 풀어 쓴다면 하단 처럼 서술 가능합니다
- Application 이 getConnection() 을 이용해서 data source 나 Connection factory 를 호출한 경우
- 속성이 맞는 Free connection 이 풀에서 가용한 경우
- 다음 둘 중의 한가지 조건을 만족하는 경우
  > Unshareable 로 지정된 자원 참조를 위한 getConnection() 요청
  > 같은 공유 속성을 가진 shareable connection 이 없는 상태에서 shareable 로 지정된 자원 참조를 위한 getConnection() 요청

어떻게 위의 내용을 보시고 이해가 좀 되시나요? 말씀 드린 것처럼 위의 경계 조건을 만족하는 경우에 한해서만 Connection 객체는 InFreePool 상태에서 InUse 상태로 전이 됩니다. 그럼 이제 이 경계 조건에 따른 Connection 객체의 상태 전이를 좀 더 살펴보도록 하겠습니다.

먼저 Connection 객체를 받아오는 경우입니다. DoesNotExist 나 InFreePool 인 상태에서 InUse 로 전이되는 경우를 의미합니다. 먼저 상태가 DoesNotExist 일 때 입니다.

모든 connection 은 DoesNotExist 상태로 시작됩니다. WAS 가 시작 된다고 하여도 connection pool 은 존재하지 않습니다. 즉, connection 이 없다는 것이죠. 어플리케이션이 첫 번째 connection 을 요청해야만 connection 이 실제로 생성됩니다. (guarding condition 에 따라서 추가적인 connection 이 생성됩니다.)

```
getConnection AND
NOT(freeConnectionAvailable) AND
poolSizeLTMax AND
(NOT(shareableConnectionAvailable) OR
unshareableConnectionRequest)
```

위의 내용만으로는 이해하기가 좀 어려울 수 있으므로 이를 좀 더 자세히 설명 드리자면,
- 어플리케이션에 의해서 data source 나 connection factory 로 getConnection() 메소드를 호출 (getConnection)
- 풀에 가용한 connection 이 없음(NOT(freeConnectionAvailable))
- 풀의 사이즈가 Maximum 풀 사이즈보다 작음 (poolSizeLTMax)
- 같은 공유 속성을 사용하는 sharable connection 이 없는 sharable connection 에 대한 요청이거나 (NOT(shareableConnectionAvailable)), unsharable connection 에 대한 요청일 경우 (unshareableConnectionRequest)

위와 같은 조건을 모두 다 만족해야지만 connection 이 생성되며 InUse 상태로 전이됩니다. (여기서 중요한 것은 모든 조건이 만족되어야지만 connection 을 생성한다는 것 입니다.)

다음으로는 상태가 **InFreePool** 일 경우입니다. Pool 에 있는 connection 을 요청하여 사용되는 보통의 경우에 InFreePool 상태에서 InUse 로 상태 전이가 발생되며, 전이는 다음과 같은 경계 조건에 따라서 이루어집니다.

```
InFreePool>InUse:
getConnection AND
freeConnectionAvailable AND
(unshareableConnectionRequest OR
NOT(shareableConnectionAvailable))
```

- getConnecion() 이 호출 된 경우
- connection 풀에 가용한 connection 이 있을 경우(freeConnectionAvailable)
- 다음의 조건을 만족할 경우
  > unsharable connection 에 대한 요청일 경우(unsharableConnectionRequest)
  > sharable connection 에 대한 요청이라면, 해당 트랜잭션에서 이미 사용중인 같은 공유 속성

을 가진 connection 이 없을 경우 (NOT(sharableConnectionAvailable)).
위와 같은 조건을 만족해야지만 InFreePool 상태가 InUse 상태로 전이될 수 있습니다.

마지막으로는 조금 색다를 수 있지만 InUse 상태일 경우입니다. 이 경우에도 역시 다음과 같이 InUse 상태가 다시 InUse 로 전이 될 수 있습니다. 이 경우는 결국 사용하는 connection 을 공유할 수 있다는 의미입니다.

```
InUse>InUse:
getConnection AND
ShareableConnectionAvailable
```

- getConnecion() 이 호출 된 경우
- 같은 공유 속성을 가진 connection 이 이미 사용 중에 있는 경우(ShareableConnectionAvailable)

와 같은 조건을 만족해야지만 사용되는 connection 을 공유할 수 있습니다.

위와 같은 조건을 만족해야지만 InUse 상태가 다시 InUse 상태로 전이될 수 있습니다. 또한 다시 한번 강조하지만 같은 사용자이면서 같은 트랜잭션 내에서 공유 속성이 모두 일치해야만, connection 을 공유할 수 있습니다. (단, 트랜잭션은 보통 단일 thread 에 연동되므로 thread 를 넘어서 connection 을 공유하려는 시도는 하지 말아야 합니다.)

반대로, InUse 상태인 Connection 객체를 상태가 InFreePool 이나 DoesNotExist 로 전이되는 경우도 당연히 존재합니다.

먼저 InUse 상태인 경우를 살펴보도록 하겠습니다. 처음은 InUse 상태에서 InFreePool 상태로 전이되는 경우입니다

```
InUse>InFreePool:
(close AND
noOtherReferences AND
NoTx AND
UnshareableConnection)
OR
(ShareableConnection AND
TxEnds)
```

- Unshareable connection 인 경우 어플리케이션 이나 컨테이너가 close() 를 호출했으며 어플리케이션이나(noOtherReferences) 트랜잭션 매니저에(NoTx) 의한 참조가 없을 경우
- Shareable connection 인 경우 트랜잭션 매니저가 트랜잭션을 종료된 경우(txEnds)

위의 조건 중에 하나만 만족하면 connection 객체는 InUse 상태에서 InFreePool 상태로 전이됩니다. 다시 말씀 드리자면 사용중인 Connection 이 free pool 로 반환됩니다.

다음으로 InUse 상태에서 DoesNotExist 상태로 전이되는 경우입니다

```
InUse>DoesNotExist:
close AND
markedStale AND
NoTx AND
noOtherReferences
```

- 어플리케이션에서 close() 가 호출된 경우
- pool cleansing step 에서 connection 에 stale 표시가 된 경우(markedStale)
- 어플리케이션이나(noOtherReferences) 트랜잭션 매니저에(NoTx) 의한 참조가 없을 경우

위의 조건을 모두 만족하는 경우, InUse 상태에서 DoesNotExist 상태로 전이되며 이 경우 Connection 은 Pool 로 돌아가지 않고 close 됩니다

마지막으로, 다음과 같은 경우 pool 에서 connection 이 제거되고 버려집니다
- RA 가 치명적인 에러(fatal error) 통지를 받으면 free pool 의 모든 connection 이 버려집니다
- pool 의 사이즈가 (poolSizeGTMin) minimum 사이즈보다 크며 free pool 안의 connection 이 지정된 unused timeout 보다 오랜 시간 존재하면(UnusedTimeoutExpired) connection 은 free pool 에서 제거되며 버려집니다.
- age timeout 이 설정되고 connection 이 그 보다 오래 살아남아 있으면 age 에 기반해서 connection 을 재활용하는 메커니즘을 제공합니다

조금은 어렵지만 지금까지 Connection 에 대한 좀 더 자세한 부분을 확인하였습니다. 굳이 Connection 의 밑단까지 다 이해를 해야 하나라는 질문이 있을 수도 있지만, WAS 에서 일어나는 가장 빈번한 일은 DB 를 접근하여 Data 를 받아오고 처리하는 일입니다. 그렇기 때문에, DB 와의 Connection 에 대해서 정확히 이해하는 것은 그 부분을 이해할 수 있는 중요한 일입니다. 가급적 꼭 숙지하시길 바라며, 이번 강좌도 여기서 마무리 하도록 하겠습니다. 이만~~~~~휘리릭... ^^&;

참고 1) IBM WebSphere Application Server v7.0 InfoCenter
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/welcome_nd.html

참고 2) IBM WebSphere Application Server v7.0 InfoCenter
- Connection management architecture
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.express.doc/info/exp/ae/cdat_wspra.html