

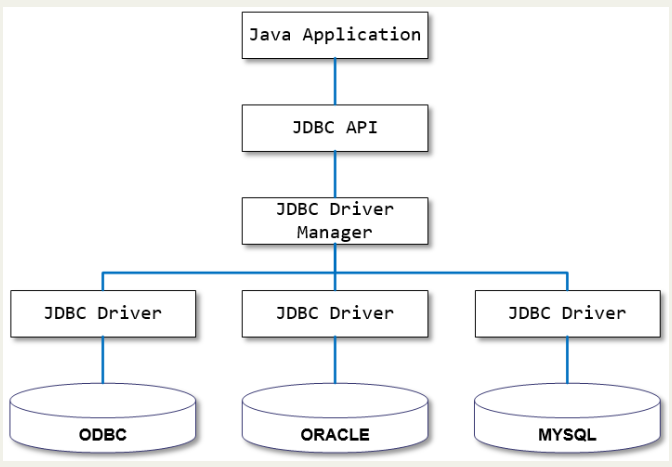
객체지향프로그래밍 중간고사 대체 과제

2016112158 김희수



<JDBC 프로그램의 구조>

JDBC(Java Database Connectivity)란 자바 프로그램에서 다른 기종간의 데이터베이스를 표준화된 방법으로 접속할 수 있도록 만든 API 규격이다. 즉, 데이터베이스가 다양함에 따라 이로 인해 어플리케이션에서 데이터베이스에 접속/데이터를 처리하는 방법이 각기 다른 것을 표준화 한것이다. JDBC API, JDBC Driver, JDBC Driver Manager로 구성되어 있다. 요약하자면 JDBC란 데이터베이스에 연결 및 작업을 하기 위한 자바 표준 인터페이스이다.



- Java Application : 네이티브 자바, 자바 서블릿, 자바 응용

- JDBC API : 자바 응용프로그램에서 데이터베이스를 연결하고 데이터를 제어할 수 있도록 데이터베이스 연결 및 제어를 위한 인터페이스와 클래스들을 제공

- JDBC 드라이버 매니저 : 자바 응용프로그램이 사용하는 데이터베이스에 맞는 드라이버를 찾아서 JDBC를 초기화한다.

- JDBC 드라이버 : 각 데이터베이스 제작업체에서 만든 데이터베이스 드라이버. JDBC 드라이버를 사용하면 JDBC를 지원하는 어떤 데이터베이스에도 접속하는 것이 가능하다

<JDBC 구조와 역할>

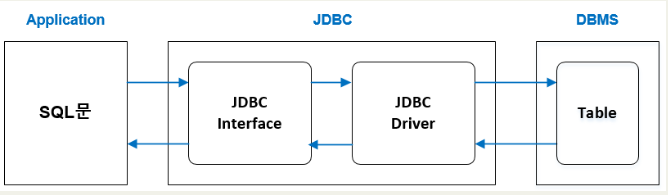
- JDBC는 크게 JDBC 인터페이스와 JDBC 드라이버로 구성되어 있다.

- 응용프로그램에서는 SQL 문을 만들어 JDBC Interface를 통해 전송하면 실제 구현 클래스인 JDBC 드라이버에서 DBMS에 접속을 시도하여 SQL문을 전송하게 된다

- DBMS의 결과는 JDBC Driver와 JDBC Interface에게 전달되고 이를 다시 응용프로그램으로 전달되어 SQL문의 결과를 볼 수 있다.

- 이처럼 JDBC는 Applicatio과 DBMS의 Bridge역할을 하게 된다

<어플리케이션, JDBC, DBMS의 관계도>



<Sysetm Architecture>

JDBC API는 2-tier와 3-tier를 모두 지원한다.

cf) tier: 일련의 유사한 객체가 나열된 상태에서 계층 또는 열을 의미한다. 프로그램의 일부가 여러 객체에 나뉘어 존재할 수 있으며, 그 계층 또한 네트워크 상의 서로 다른 컴퓨터에 위치할 수 있다.

**1) 2-tier**

- 자바 애플릿이나 애플리케이션이 JDBC를 이용하여 DBMS 데이터베이스에 직접 접근

- 가장 대표적인 C/S(Client/Server) 구조

- 프로그래밍이 간단하다는 장점

- 보안, 로드밸런싱, 확장성(오브젝트 재사용) 등의 문제점

- 2-tier 디자인이 적합한 경우

· 애플리케이션이 하나의 데이터베이스만을 사용

· 데이터베이스 엔진이 하나의 CPU에서 동작

· 데이터베이스가 계속 거의 같은 크기로 유지

· 사용자 기반이 같은 크기로 유지

· 요구가 확정되어 변화 가능성이 극히 적거나 없는 경우

· 애플리케이션을 종료한 후에도 최소한의 지속성을 요구

**2)3-tier**

- 자바 애플릿이나 애플리케이션이 DBMS를 직접 접근하는 것이 아니라  
  중간에 미들웨어(미들티어)를 거쳐 데이터베이스로 접근

- 데이터베이스와의 연동 부분을 분리시킴으로써

  Presentation Layer가 데이터 저장 방법에 신경을 쓰지 않아도 된다.

- 클라이언트는 단지 미들티어를 참조

- 미들티어 서버는 DBMS와 같이 특정한 작업만 수행하는

  최종 서버와 통신을 하여 결과를 얻은 후 이 결과를 클라이언트에게 전달

- 2-tier 모델보다 안정적이고 유연하며 보안이 강화

· tier 1 : 사용자 인터페이스를 담당하는 클라이언트

· tier 2 : http, 코바 등을 지원하는 응용 처리 서버

· tier 3 : DBMS와 같이 사용자가 최종적으로 원하는 기능을 수행할 서버

< JDBC Driver 유형>

**1) Type 1**

- JDBC-ODBC Bridge

- 특징

· 데이터베이스를 연동하기 위해 브릿지 기술을 사용

· ODBC API로의 게이트웨이를 제공하여 실제로는 ODBC의 API를 구현함으로써 데이버이스를 만듦.

· 브릿지 솔루션은 보통 클라이언트에 소프트웨어가 설치될 것을 요구

· JDBC-ODBC 드라이버는 ODBC 드라이버가 풍부하기 때문에 거의 대부분 데이터베이스 시스템에서 사용할 수 있으며 JDBC-ODBC 드라이버를 사용하는 클라이언트에 사전에 ODBC 드라이버가 설치되어 있는 겨우 매우 유용하게 사용할 수 있다.  
 · 속도와 관련한 가장 큰 문제: JDBC를 통해 호출된 명령이 다시 ODBC를 통해 나가야 하기 때문에 두 개의 브릿지를 거치며, 이로 인해 빠른 속도를 기대하기 어렵다.  빠른 성능을 요구하는 애플리케이션의 경우에는 Type 1, JDBC-ODBC 브릿지는 적당하지 않다.

· JDBC-ODBC 브릿지를 사용하는 시스템에는 반드시 해당 데이터베이스에 연결하기 위한 ODBC 드라이버가 설치되어야 하며, 이 단점은 애플릿에서 JDBC를 사용하여 프로그래밍 하는 경우 많은 문제가 된다.  애플릿을 이용하여 JDBC-ODBC를 사용할 경우, 애플릿을 다운받은 클라이언트에 미리 ODBC 드라이버가 설치되어 있어야 하기 때문에 배포 등에 많은 문제가 발생하게 된다.

**2) Type 2**

Native-API / Partly Java Driver(사용 일부 자바)

- 특징

· 각각의 데이터베이스 제조업체들이 제공한 C 혹은 C++ 메소드를 자바 코드가 호출하는 방식

· 부분적으로 자바 드라이버인 원시 API 라고 일컬어짐  
· 데이터베이스와 연결되는 부분이 Native Code로 구현되어있는 만큼 JDBC-ODBC 브릿지에 비해 빠른 속도를 제공한다.

· JDBC 드라이버를 사용하고자 하는 각각의 클라이언트에 DBMS Vendor의 데이터베이스 라이브러리가 로드되어야 하기 때문에 인터넷이나 CS 환경에서는 사용하기 적합하지 않다.  또한, Type3, Type4 드라이버에 비해 낮은 성능

**3) Type 3**

- Net-Protocol / All-Java Driver(순수 자바)  
- 클라이언트에서 일반적인 Network API를 이용해 보낸 정보를 서버가 Database에 독점적인 형태로 변환하는 방식  
- 특징

· 클라이언트에 존재하는 JDBC 드라이버는 소켓을 사용하여 서버에 존재하는 미들웨어(Middleware) 애플리케이션에 연결

· 애플리케이션은 클라이언트의 요청을 사용하고자 하는 데이터베이스에 독점적인 APU로 전환

· 하나의 드라이버로 여러 개의 데이터베이스를 연동

· 클라이언트에 소프트웨어를 설치할 필요가 없음

**4) Type 4**

- Native-Protocol / All-Java Driver

- Database Engine에 사용되는 Network Protocol을 자바 소켓으로 직접Database에 교신하는 방식

- 특징

· 가장 직접적인 순수 자바 솔루션

· 거의 대부분 데이터베이스의 제조업체가 제공한다.

· ODBC나 Native Lib 형태로 request를 변환하지 않기 때문에 Performance가 매우 좋다. 또한, 특별하게 Driver나 Lib, Middleware 등을 설치할 필요가 없기 때문에 배포 등이 매우 용이하다.

<JDBC API>

JDBC API는 데이터베이스의 데이터를 액세스할 수 있도록 제공되는 표준 자바 API 클래스와 인터페이스로 구성되며 다음 기능을 실행하기 위한 클래스와 인터페이스를 제공한다.

- Java 프로그램에서 데이터베이스에 서버 접속

- SQL 문을 구성, 데이터베이스 서버에서 실행

- 데이터베이스 서버가 처리한 결과 가져오기

- 데이터베이스의 정보, 처리 결과에 대한 정보 등을 가져오기

※**java.sql.\* 패키지(package)**

- Java Application 으로부터 Database를 조작하는 API는 JDK의 코어 API로 『java.sql.\*』 패키지에 설정되어 있다.

- JDBC는 데이터베이스에 접속하기 위해 한 개의 클래스(java.sql.DriverManager)와 두 개의 인터페이스(java.sql.Driver 와 java.sql.Connection)를 사용한다.

<Java Socekt Programming>

소켓이란…

소켓이란 두 호스트 사이의 연결을 말한다. 소켓은 아래와 같은 기본적인 7가지 기능을 수행한다.

- 원격 장비에 연결하기(connect)

- 데이터 보내기(send)

- 데이터 받기(recv)

- 연결 닫기(close)

- 포트 지정하기(bind)

- 수신 대기하기(listen)

- 수정된 포트에 대해 원격 장비의 연결 받아들이기(accept)

자바의 Socket 클래스는 위에 나열된 기능 각각에 해당하는 메소드를 제공하며, 앞의 네 개는 서버와 클라이언트 모두에서 사용되고, 나머지 세 개는 클라이언트의 접속을 대기하는 기능으로 서버에서만 사용된다.

자바 프로그램은 보통 다음과 같은 방법으로 클라이언트 소켓을 사용한다.

- 프로그램은 Socket 클래스의 생성자를 사용하여 새로운 소켓을 만든다.

- 생성된 소켓은 원격 호스트에 연결을 시도한다.

TCP / UDP?

**TCP**

TCP란 소켓 프로그래밍 중의 하나로 스트림 통신 프로토콜이라고 부르며, 양쪽의 소켓이 연결된 상태여야만 가능하기 때문에 연결지향 프로토콜이라고도 한다.

이 TCP 프로토콜은 신뢰성 있는 프로토콜이기 때문에, 송신한 쪽의 데이터가 수신 측에 차례대로, 중간에 유실되는 일 없이 도착 하는 것을 의미합니다. 그러려면 수신 측과 송신 측이 미리 연결을 맺고 연결된 순서대로 데이터를 교환해야 한다.

연결지향 방식은, 한번 연결되면 연결이 끊어질 때까지는 송신한 데이터가 차례대로 목적지의 소켓에 전달되는 신뢰성 있는 통신이 가능하다.

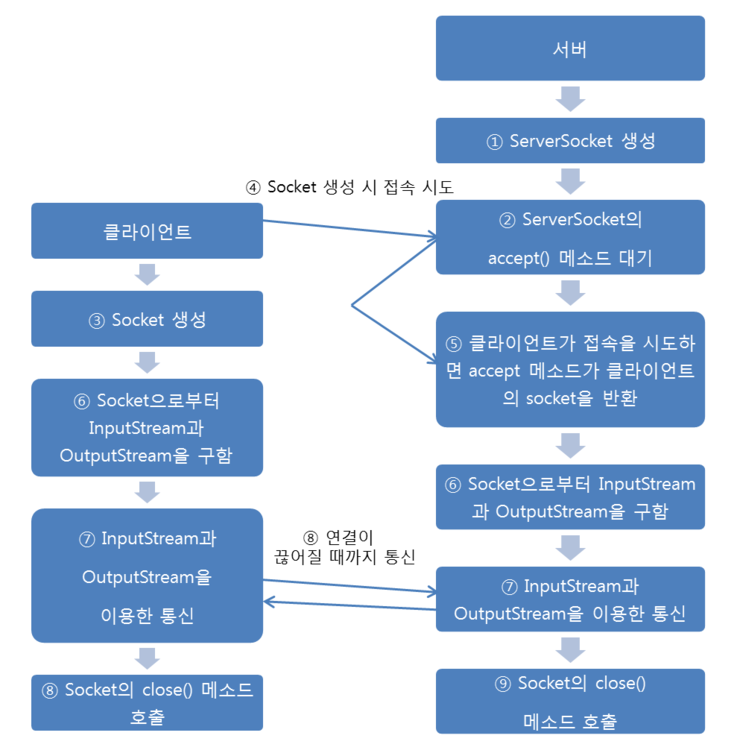
이렇게 신뢰성 있는 TCP 연결을 하기 위해서는 TCP 프로그래밍에서 사용하는 라이브러리의 사용 방법과 동작 순서를 정확하게 숙지, 이해하고 있어야한다. 히 자바는 TCP 프로그래밍을 쉽게 할 수 있도록 java.net 패키지에 관련 클래스들을 미리 준비해놓고 있기 때문에, 해당 패키지의 클래스들을 알맞게 사용할 수 있어야 한다

TCP 프로그래밍에서 중요한 클래스는 다음과 같다

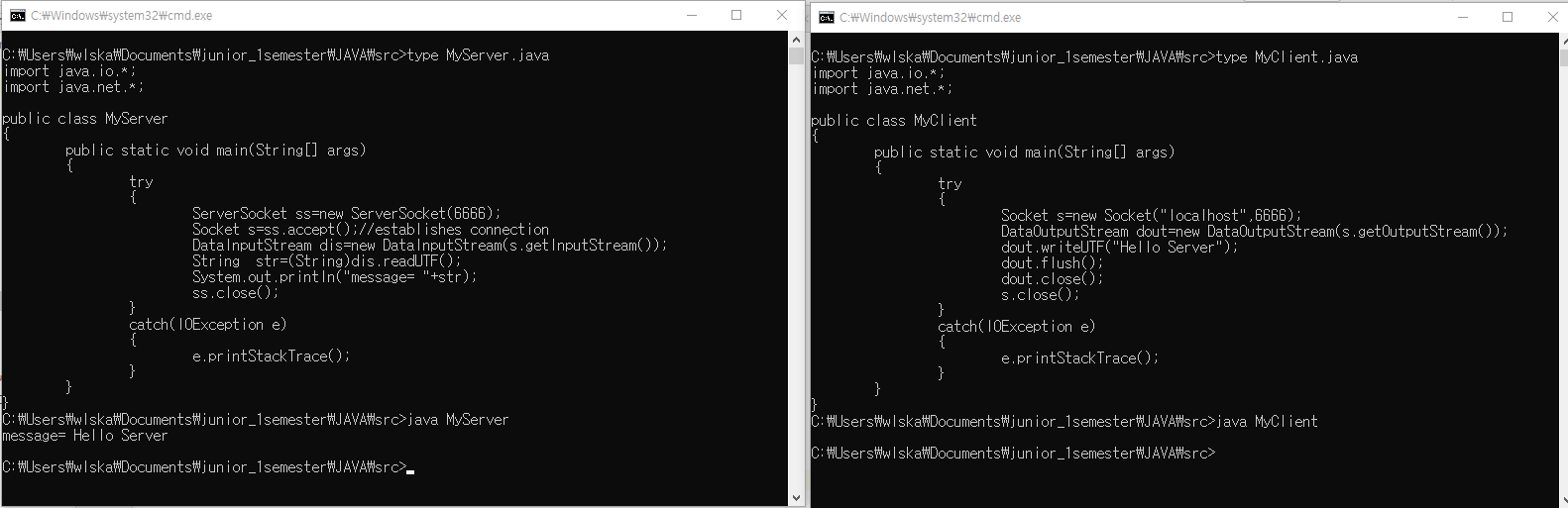
- Java.net.ServerSocekt : 서버쪽에서 클라이언트의 접속을 대기하기 위해서 반드시 필요한 클래스

- Java.net.Socket : 서버와 클라이언트가 통신하기 위해서 반드시 필요한 클래스

각각 서버와 클라이언트를 사용하는 클래스는 위와 같으며, 통신순서는 아래와 같다



(프로그래밍 예제)



왼쪽은 서버, 오른쪽은 클라이언트

서버에서 ServerSocket ss = new ServerSocet(6666)은 Server를 생성한다. Socket s = ss.accept()는 클라이언트와의 연결을 만들때까지 기다린다.

클라이언트에서 Socket s = new Socket(“localhost”,6666)은 클라이언트를 생성한다.

클라이언트에서 DataOutputStream 객체 dout을 생성한 후, dout.write로 쓰기를 수행하면 서버에서 생성한 DataInputStream 객체(dis)에서 그 데이터를 받아와 str에 읽어온다(String str = (String)dis.readUTF(). 그리고 그 데이터를 출력한다. 여기서 입력된 데이터는 “Hello Server”이다.

**UDP**

UDP는 User Datagram Protocol의 약자이며, 데이터그램 통신 프로토콜이라고도 합니다. TCP와는 다르게 비연결성(connectionless) 프로토콜이기도 하다.

UDP의 특징은 다음과 같다.

①    패킷을 보낼 때마다 수신 측의 주소와 로컬 파일 설명자를 함께 전송해야 한다.

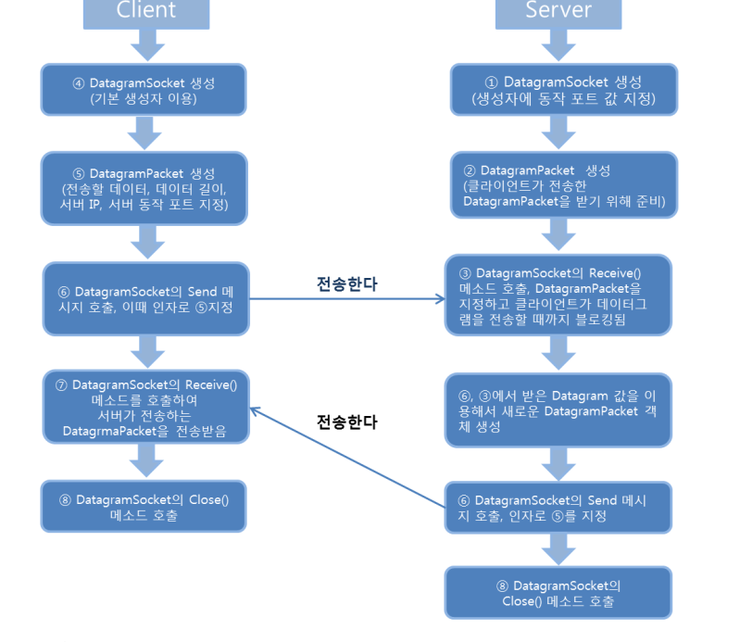
②    비연결성이기 때문에 TCP보다 신뢰성이 떨어진다.

③    TCP보다 빠르게 데이터를 주고받을 수 있다.

간략하게 요약하자면, 속도는 빠르지만 TCP보다 신뢰성이 떨어진다고 할 수 있다.

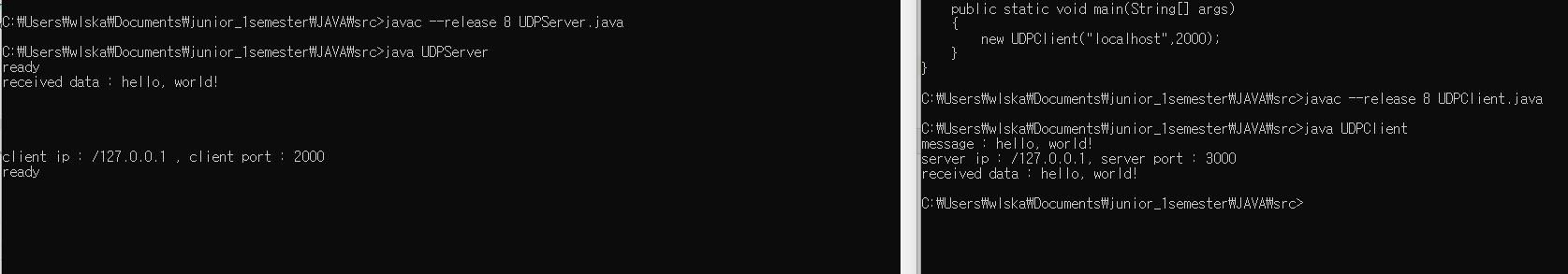
이러한 UDP를 이용한 프로그래밍을 하려면 클라이언트와 서버 모두 java.net 패키지 안에 있는 DatagramSocket 객체를 생성해야 하고, 데이터를 주고받기 위해서 DatagramPacket 객체를 이용해야 합니다.

UDP에서 클라이언트-서버 동작순서



클라이언트는 데이터를 전송하기 위해 DatagramSocket을 생성하고, DatagramPacket에 전송할 데이터, 데이터길이, 서버IP, 서버 동작 포트 지정등의 데이터를 싣고 서버로 전송하게 된다. 서버는 이렇게 전달된 Packet(Datagram)을 읽어 들인 후, 그 패킷의 값을 이용해서 전송할 패킷을 새롭게 구성하게 된다. 그리고 그 패킷을 이용하여 클라이언트에 다시 전송하게 된다. DatagramPacket을 이용하여 서버와 클라이언트가 packet을 주고 받고 하고 있는 것이다.

(프로그래밍예제)



왼쪽은 서버 오른쪽은 클라이언트

-- UDPClient --

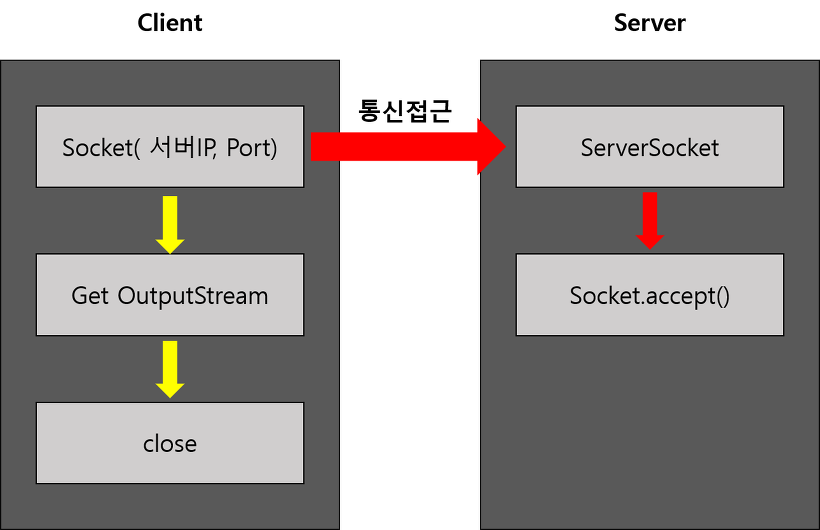
InetAddress객체 ia에서 ip주소를 저장. Datagram을 송수신하기위한 소켓객체 DatagramSocekt 객체 ds를 생성한다. 콘솔입력을 위한 BufferedReader객체 file을 생성하고 콘솔입력을 받아와 str에 저장한다. 버퍼를 생성해 str크기를 그 크기만큼 버퍼에 저장한다. Datagram을 송수신하기위한 패킷객체 dp를 생성하고 데이터그램을 송신한다. 서버로부터의 수신을 위한 buffer를 생성하고 buffer크기를 가지는 패킷객체 dp를 다시 만든다. 패킷을 수신받는다. 수신 받은 패킷의 ip주소와 port를 출력한다.

-- UDPServer--

데이터그램을 송수신하기 위한 소켓객체 ds를 생성한다. 수신을 위한 버퍼를 생성하고 그 버퍼크기만큼의 패킷객체 dp를 생성한다. 데이터그램을 받아서 dp에 저장한다. dp에 저장된 데이터를 String으로 바꾼 str을 만든다. 데이터그램에 저장된 데이터를 출력한다.

수신된 패킷(클라이언트로부터 온)의 IP주소가 저장된 객체를 ia에 저장한다. port에는 수신된 패킷의 port를 저장한다. 수신된 패킷의 ip주소와 port를 출력한다. 서버의 ip주소와 port정보가 포함된 패킷객체 dp를 재정의하여 송신한다.

<통신방식>



1. 먼저 Client class를 생성한다. 클래스의 파라미터는 위와 같다.

2. 해당 클래스에 서버의 ip주소와 port 번호를 넣고 출력 스트림으로 넘어간후 Server Socket에 접근한다.

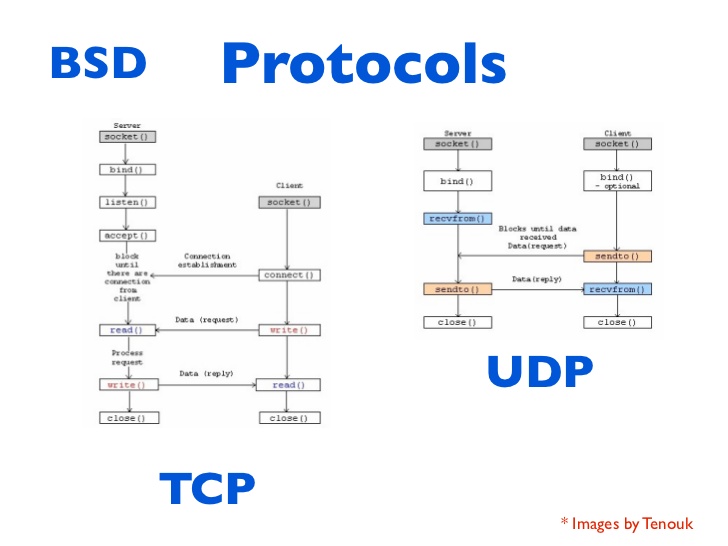
3. Server Socket class는 client가 접속을 했는지 체크만하는 용도이다.

4. 접근이 왔음이 인식되면 서버클래스는 재빨리 Socket.accpet()메소드를 실행한다

<예시>

<C로 작성한 BSD Socket 프로그래밍과 비교>

(C로 작성한 BSD Socket)

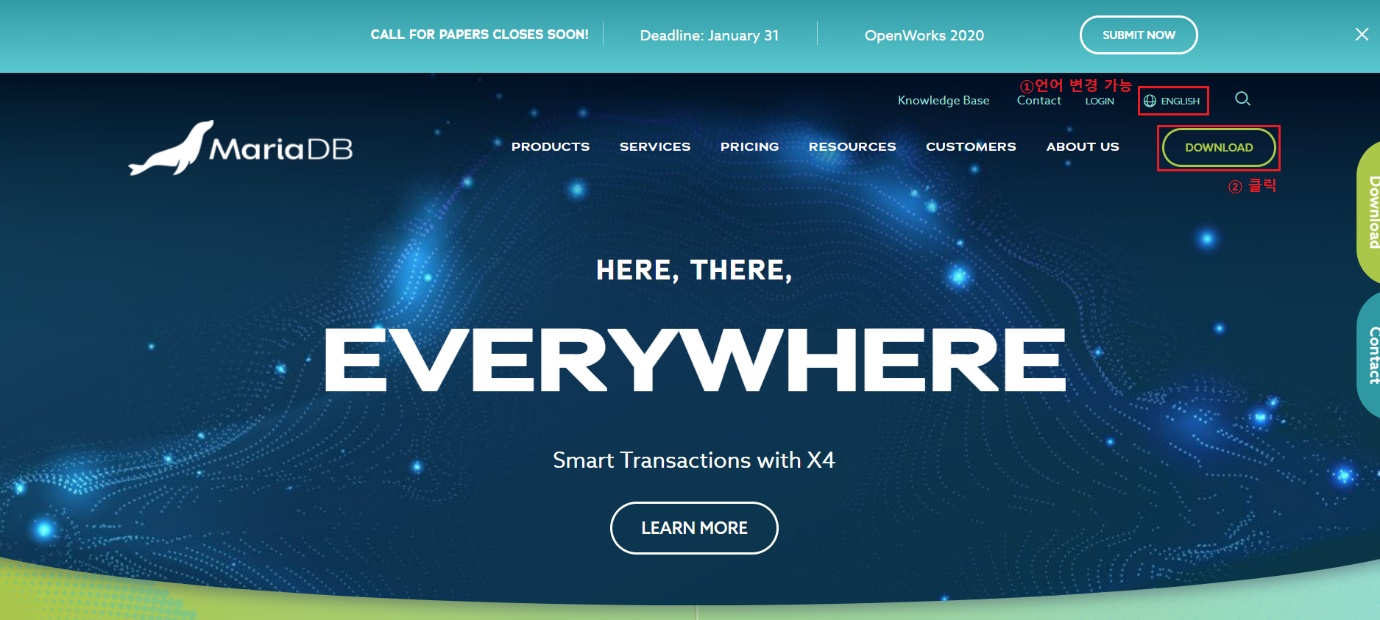


java socekt과 BSD socket의 구조상의 차이는 별로 없어 보인다. TCP의 경우 java외 bsd모두 TCP connection establishment, data request, data reply의 과정을 따르며 UDP의 경우 connectionless하여 서버에서 클라이언트가 보낸 데이터를 수신하기 전까지는 block한다는 점도 같다. 하지만 자바는 객체지향언어이고 C는 절차지향언어이므로 구현상의 차이가 존재할 것으로 보인다.

<Maria DB 설치기록>

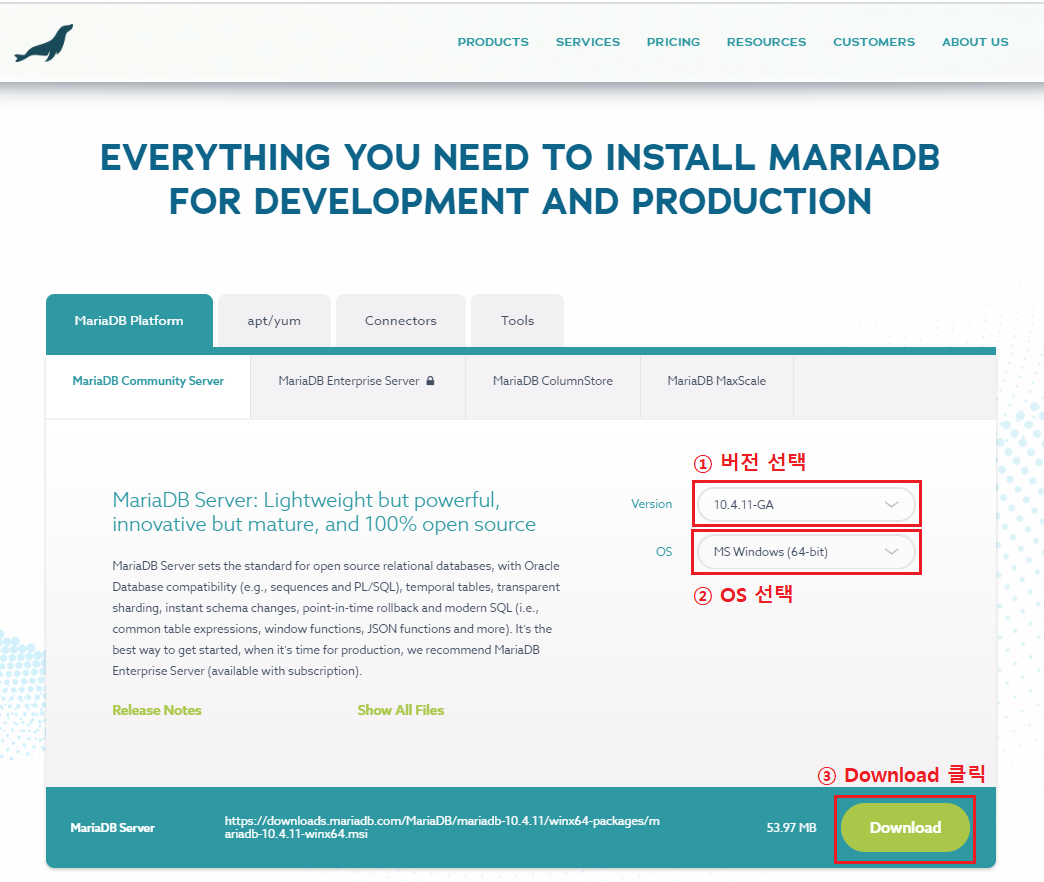
1. 마리아 DB 사이트 접속

<http://mariadb.org/>

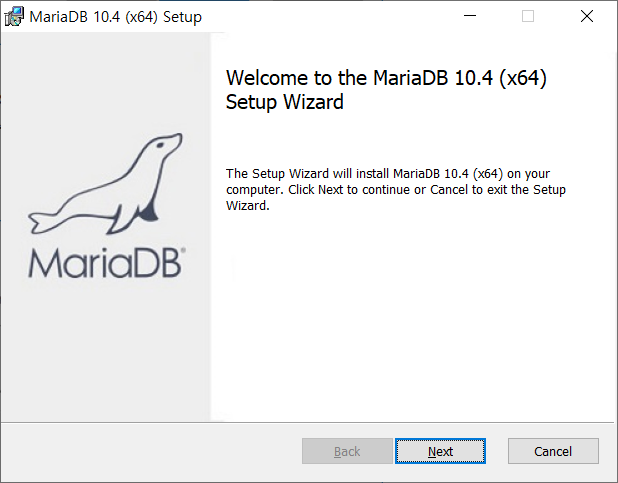


2. 버전 및 OS 선택

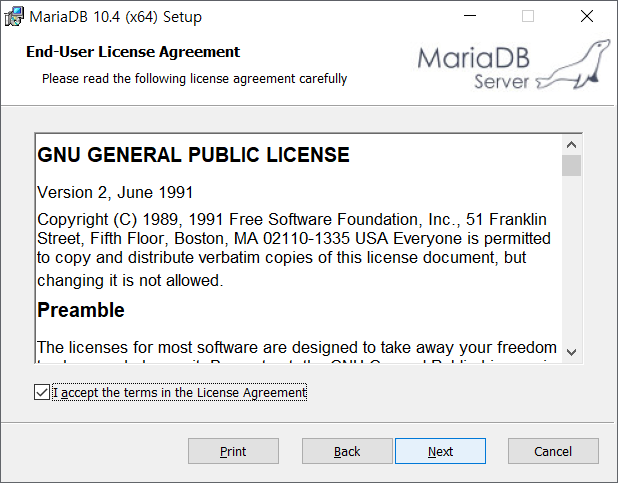
<https://downloads.mariadb.org/>



3. 다운로드 받은 설치 파일을 더블클릭하여 실행한다

- “Next” 버튼을 눌러 진행한다

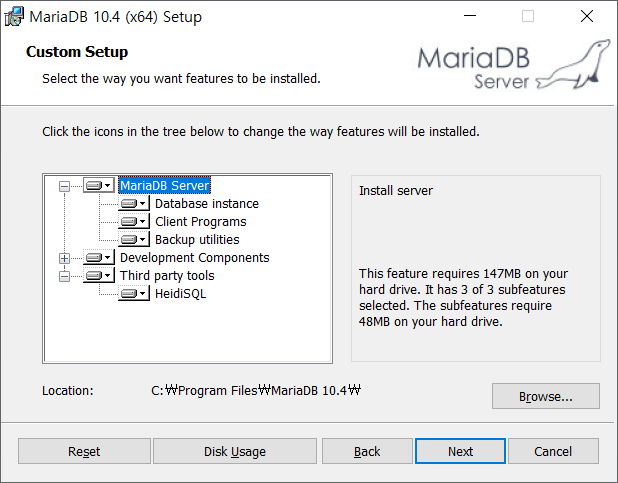
- 라이선스 동의에 체크하고, “Next” 버튼을 눌러 진행한다



- 설치 경로를 설정한다

기본 경로:C:\Program Files\MariaDB 10.4\

기본경로로 설치할 예정이므로 “Next”를 눌러 진행한다



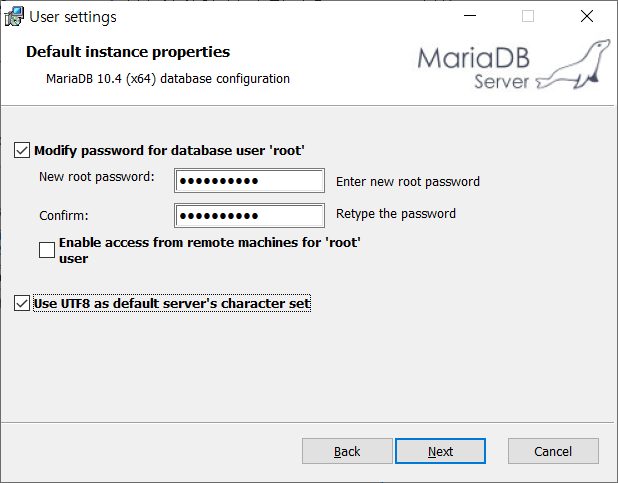
- ‘root' 계정의 비밀번호를 입력한다.

1) "Enable access from remote machines for 'root' user" 체크 해제

- 굳이 내 컴퓨터를 중앙 Database로 설정하지 않을 예정이기 때문에 다른 컴퓨터(remote machines)에서 root 계정으로의 접속을 허용하지 않았다.

2) "Use UTF8 as default server's character set" 체크

- 데이터베이스의 기본 캐릭터셋을 UTF로 지정한다.



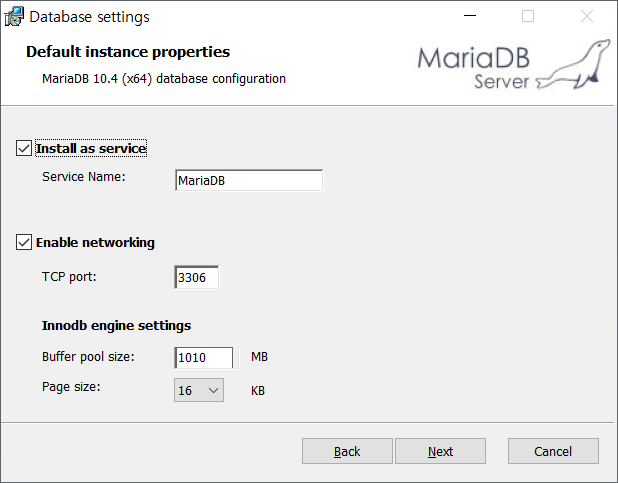
- 기본 설정 그대로 "Next"클릭하여 진행 한다.

1) "Install as service"

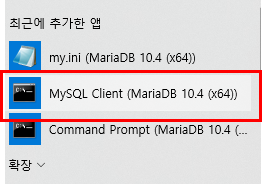
- 서비스명은 설정 하는 부분이며, "MariaDB"를 기본으로 사용하자.

2) "Enable networking"

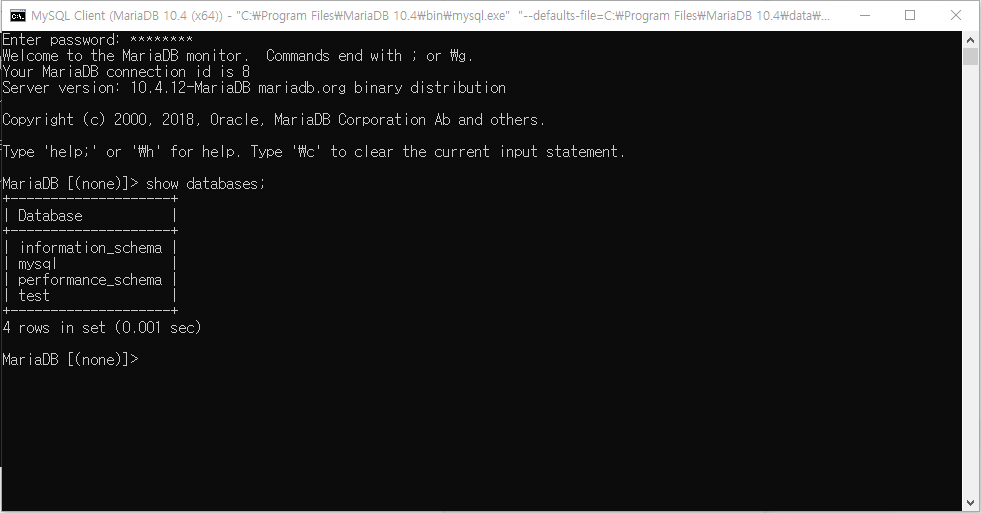
- 포트는 3306을 기본값으로 사용한다. (오라클은 1521을 기본으로 사용하니 겹치진 않는다.)

- Install이 나올때까지 “Next”를 눌러서 install이 나오면 install해준다. 잠시 기다리면 설치가 완료된다.

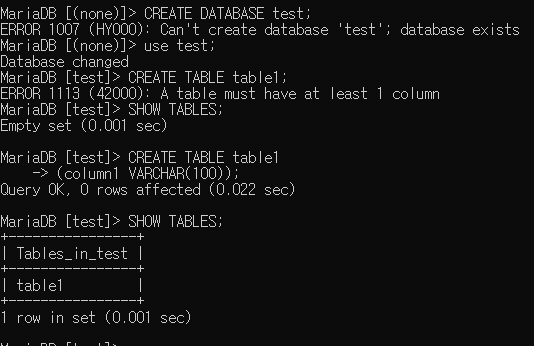
- 시작에서 MySQL Client(MariaDB 10.4)를 찾아 실행한다



- 실행후 아까 설정했던 비밀번호를 입력해주면 올바르게 접속하는 것을 볼 수 있다.

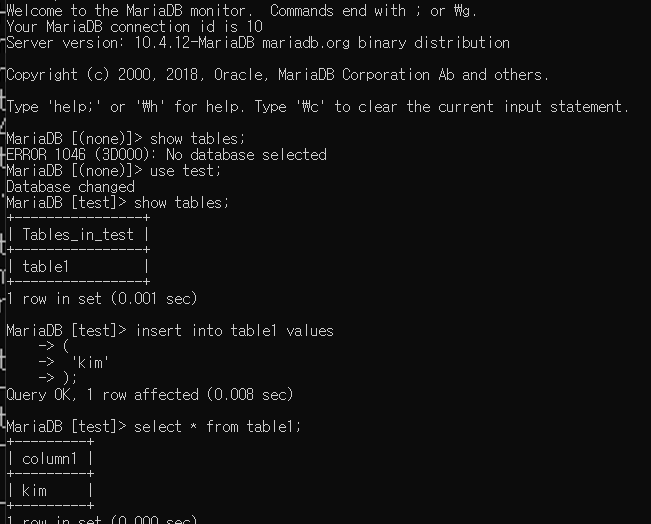


<table생성>



CREATE DATABASE test;는 test라는 이름을 가진 데이터베이스를 생성한다. use test는 작업환경을 test데이터베이스로 옮긴다. CREATE TABLE table1->(colum1 VARCHARR(100));은 table1이라는 테이블을 생성하고 데이터타입이 VARCHAR(100)이고 column1이라는 이름을 가진 attribute를 생성한다. SHOW TABLES를 수행하면

<tuple 집어넣기>



insert into table1 values->(‘kim’);을 수행하면 column1이 ‘kim’인 tuple하나가 생성되는 것을 볼 수 있다.

참고자료)

JDBC란 <https://o-jing.tistory.com/28>

System Architecture, JDBC Driver 유형, JDBC API <https://racoonlotty.tistory.com/41>

Socket이란…<https://cbts.tistory.com/84>

TCP/UDP

<http://blog.naver.com/PostView.nhn?blogId=cyberhole&logNo=110133796699&categoryNo=55&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=1&from=postView&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=1>

소켓의 통신방식 <https://mainpower4309.tistory.com/25>

자바 소켓 통신 예제 : <https://www.javatpoint.com/socket-programming>

BSD Socket <https://www.slideshare.net/zhesto/zeromq>

MariaDB 설치 <https://goddaehee.tistory.com/201>