[NCS-학습모듈의 위치]

대분류	정보통신
중분류	정보기술
소분류	정보기술개발

세분류	능력단위	학습모듈명
SW아키텍처	요구사항 확인	요구사항 확인
응용SW엔지니어링	어플리케이션 설계	어플리케이션 설계
시스템엔지니어링	어플리케이션 구현	자바기반 어플리케이션 구현
DB엔지니어링	화면 구현	화면 구현
NW엔지니어링	데이터 입출력 구현	데이터 입출력 구현
보안엔지니어링	통합 구현	통합 구현
UI/UX엔지니어링	개발자 테스트	개발자 테스트
	정보시스템 이행	정보시스템 이행
	제품소프트웨어 패키징	제품소프트웨어 패키징
	소프트웨어공학 활용	소프트웨어공학 활용
	데이터베이스 요구사항 분석	데이터베이스 요구사항 분석
	데이터베이스 구현	데이터베이스 구현
	SQL 활용	SQL 활용

차 례

어플리케이션 구현 학습모듈의 개요

학습모듈의 목표

개발에 필요한 환경을 구축하고, 어플리케이션설계를 바탕으로 공통모듈,단위테스트,어플리케이션 성능개선을 수행할 수 있다.

선수 학습

이해(JAVA)

교육훈련 대상 및 이수시간(예시)

	학습내용	교육훈련 대상 및 이수시간(hour)			
		고등학교	전문대학	대학교	대학원
1. 구축하기	1-1. 개발필요 사항 검토,준비				40
	1-2. 개발필요 H/W,S/W 구축				
	1-3. 형상관리 환경 구축				
2. 공통 모듈 구현하기	2-1. 디자인 패턴				48
	2-2. 공통모듈 작성				
	2-3. 결합도 줄이고 응집도 높인 공통모듈 구현				
	2-4. 모듈의 기능 및 인터페이스 테스트				
3. 단위 테스트하기	3-1. 단위테스트 표준,절차,기법 정의				14
	3-2. 단위테스트 계획수립과 수행				
	3-3. 단위 모듈/컴포넌트 테스트케이스 검증				
	3-4. 테스트결과 결함발견 개선				
4. 어플리케이션 성능 개선하기	4-1. 어플리케이션 성능확인 및 목표성능 충족				10
	4-2. 프로그램표준 및 가이드라인에 따른 코드 품질 메트릭스				
	4-3. 소스코드 내재 품질 수준 분석				

※ 교육훈련 대상 및 이수시간은 NCS 능력단위 요소별 수준에 근거하며, 교육훈련 및 산업체 현장 전문가의 의견을 수렴하여 참고로 제시한다.

핵심 용어

개발방법론, UML, CBD, 객체지향개발방법, 디자인패턴, 구조적개발방법, 네트워크, 백신, 침입탐지, 방화벽, DB, SQL, API, EAI, WebService, 모듈, EDA, MDA, Middleware, WAS, TP Monitor, SOA, Agile기법, ISTQB Syllabus, CSTE CBOK, 테스트레벨, 형상관리, 단위테스트, 인스펙션, 플랫폼, 알고리즘, 자료구조

학습 1	개발환경 구축
학습 2	JCF(Java Collection Framework)
학습 3	제너릭스와 어노테이션
학습 4	스레드
학습 5	입/출력
학습 6	네트워킹
학습 7	DB 연동 프로그래밍
학습 8	Servlet

1-1. iBatis(MyBatis) Plug-in 설치 및 환경설정

학습목표	• iBatis(MyBatis) Plug-in 설치 후, 환경설정을 통하여 개발환경을 구축할 수 있다.

필요 지식 /

1 iBatis 및 환경설정 하기

1. 해당 라이브러리를 추가한다.

(1) ibatis-sqlmap-2.3.4.726.jar파일을 프로젝트 폴더에 복사하여 넣어준다.

※ iBatis사용시, 오라클 드라이버 파일(ojdb6.jar)도 필요하기 때문에 같이 복사해 주어야 함.

2. 복사한 라이브러리 파일을 해당 프로젝트의 build path 경로에 추가해 준다.

(1) 프로젝트폴더 우클릭 > properies > Java Build Path 의 Libraries 탭을 선택한다.

(2) Add JARs 버튼을 클릭하여 해당 프로젝트 내의 jar 파일들을 선택해 준다.

(3) 정상적으로 처리되었다면 그림과 같이 라이브러리 목록에서 해당 jar파일을 확인 할 수 있다.

3. iBatis 환경설정을 위한 설정파일을 클래스 패스에 위치시킨다.

4. iBatis 환경설정을 위한 세부사항을 설정한다.

(1) SqlMapConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
    <!--
    DB와 연결하는 정보를 properties파일에 설정해 놓고 이 properties파일의
    내용을 읽어와 설정한다.
     -->
    <properties resource="SqlMapConfig.properties" />

    <!-- ibatis를 처리하기 위한 환경설정 부분 -->
    <settings cacheModelsEnabled="true" enhancementEnabled="true"
        lazyLoadingEnabled="true" maxRequests="32" maxSessions="10"
        maxTransactions="5" useStatementNamespaces="true" />

    <!--
    VO클래스등의 이름이 패키지명까지 기술하면 길어지는데 이것을 간단히
    표시하기 위해서 alias를 지정할 수 있다.
    형식) <typeAlias alias="alias명" type="클래스의 풀네임"/>
     -->
    <typeAlias alias="memVO" type="ibatis.member.vo.MemberVO"/>

    <!--
    DB와의 연결을 처리하는 부분
    SqlMapConfig.properties에 설정해 놓은 정보를 이용하여 구성한다.
     -->
    <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
        <!--
            각종 설정을 직접 기술해서 처리할 수도 있다.
            <property name="JDBC.Driver" value="oracle.jdbc.driver.OracleDriv
er" />
         -->
            <property name="JDBC.Driver" value="${driver}" />
            <property name="JDBC.ConnectionURL" value="${url}" />
            <property name="JDBC.Username" value="${username}" />
            <property name="JDBC.Password" value="${password}" />
        </dataSource>
    </transactionManager>

    <!--
    실제 처리할 SQL문은 xml문서로 따로 만든 후
    그 xml문서와 아래와 같이 연결하여 사용한다.
    형식) <sqlMap resource="경로명/파일명.xml"/>
    -->
    <sqlMap resource="basic/member.xml"/>

</sqlMapConfig>
```

(2) member.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
            PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
            "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<!-- namespace속성 : 아래에서 작성한 SQL을 호출할 때 같이 사용된다. -->
<sqlMap namespace="memberTest">
    <!--
        이 영역에 sql문에 맞는 태그를 사용하여 SQL문을 기술한다.

        사용할 수 있는 기본적인 태그들
        <select> ~~~ </select>
        <insert> ~~~ </insert>
        <update> ~~~ </update>
        <delete> ~~~ </delete>
     -->

     <!-- insert 연습 -->
     <insert id="insertMember" parameterClass="basic.MemberVO">
            insert into mymember (mem_id, mem_name, mem_tel, mem_addr)
            values ( #mem_id#, #mem_name#, #mem_tel#, #mem_addr# )
     </insert>

     <!-- update 연습 (parameterClass에 alias를 지정할 수 있다.) -->
     <update id="updateMember" parameterClass="memVO">
            update mymember set mem_name=#mem_name#,
                mem_tel=#mem_tel#, mem_addr=#mem_addr#
            where mem_id=#mem_id#
     </update>

     <!-- delete 연습 -->
     <delete id="deleteMember" parameterClass="String">
            delete from mymember where mem_id=#sss#
     </delete>

     <!-- select 연습 -->
     <select id="getMemberAll" resultClass="memVO">
            select * from mymember
     </select>
</sqlMap>
```

5. 간단한 예제를 통해 제대로 설정되었는지 확인해 본다.

```
/* iBatis의 환경설정파일을 읽어와 SqlMapClient객체 생성하기 */
    public static void main(String[] args) {
        try {
            // 1. Reader객체 이용하여 xml문서 읽어오기
            Charset charset = Charset.forName("UTF-8");//설정파일의 인코딩정보
            Resources.setCharset(charset);
            Reader rd = Resources.getResourceAsReader("sqlMapConfig.xml");

            // 2. 위에서 읽어온 Reader객체를 이용하여
            // 실제 작업을 진행할 객체 생성
            SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
            // 3. 사용한 Reader객체 닫기
            rd.close();
        } catch (IOException e) {
            System.out.println("설정파일 로딩 실패!!!");
        }
    }
```

1-2. SVN 설치 및 환경설정

학습목표	• SVN 설치하고 필요한 환경설정을 할 수 있다.

필요 지식 /

1 SVN 및 환경설정

1. 형상관리 도구의 종류

1. Git	2. SVN
2005	2000년
(distributed revision control) 리누스 토발즈가 리눅스 커널 개발에 이용하려고 개발	CVS를 대체하기 위해 개발, 현재는 아파치재단에서 개발
로컬저장(분산저장)으 로 Offline 개발 가능, 속도빠름	CVS와 비교해서 장점 - 폴더 이동이 자유롭다. - Commit단위로 revision 관리 - Atomic Commit: 행동단위로 복구가능
OpenCV MPC-HC	BitCoin

3. < 2-1> 형상관리 도구 종류

2. 형상관리 도구의 선정

(1) 개발 시스템 환경등을 고려하여 선정한다. (본 예제에서는 SNV을 사용함.)

(2) 클라이언트 프로그램 설치(SVN 서버는 설치되었거나 오픈 서버 사용을 가정함)

(가) 이클립스 마켓에서 SVN 플러그인 검색

Help 메뉴 -> Eclipse Market place

그림 2-19 이클립스 마켓에서 플러그인 검색

( ) 플러그인 설치

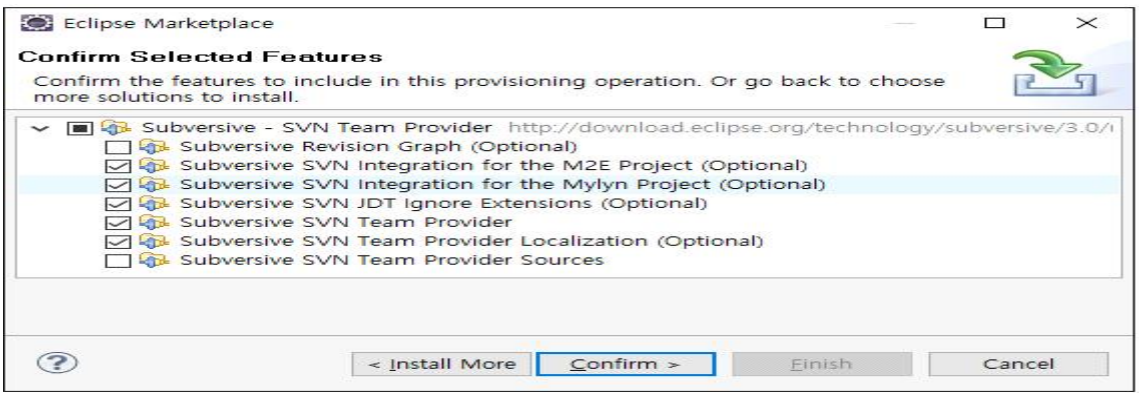


그림 2-20 이클립스 SVN 클라이언트 플러그인 설치

(다) 플러그인 설치 후 이클립스 재시작

(라) 퍼스펙티브를 SVN Repository Exploring 으로 변경 후 SVN 커넥터 설치

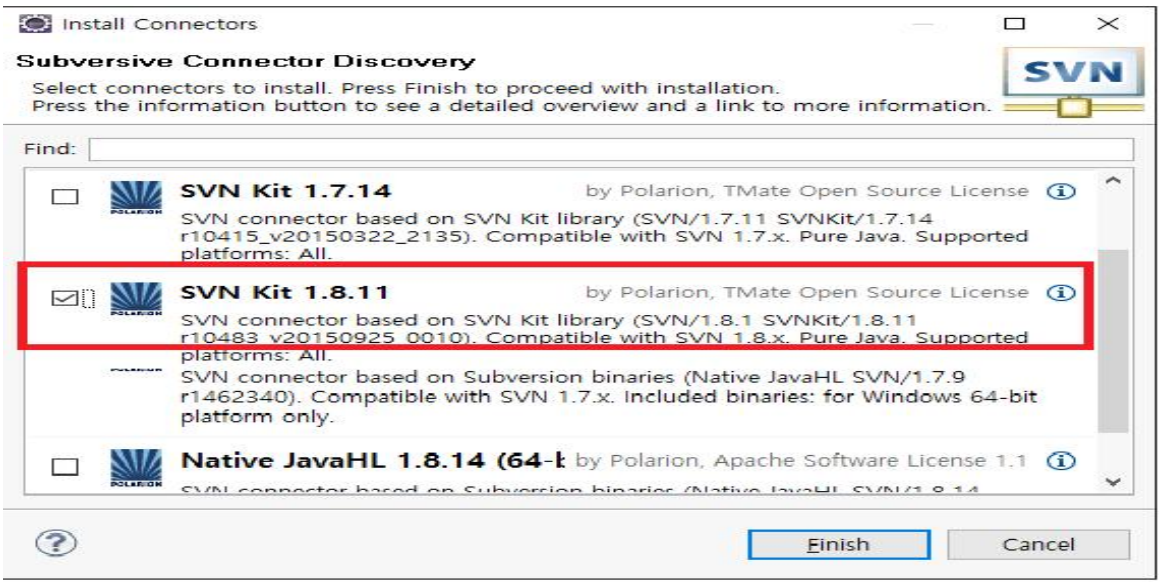


그림 2-21 SVN 커넥터 설치(1)

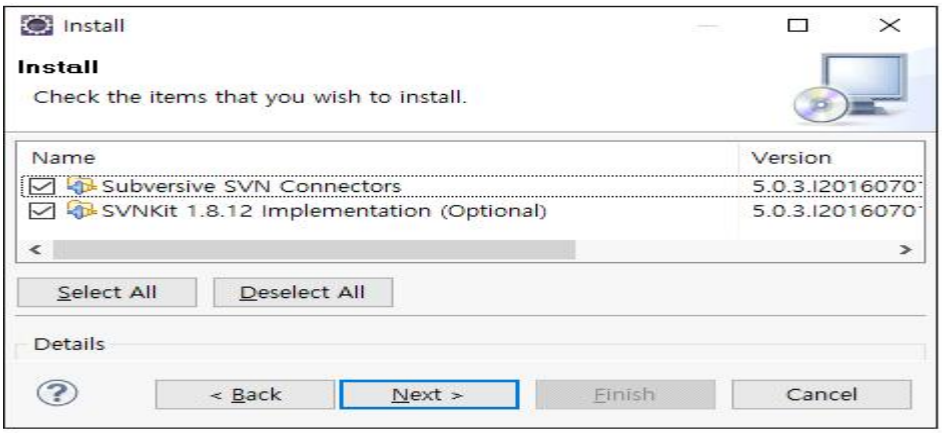


그림 2-22 SVN 커넥터 설치(2)

( ) 이클립스 재시작

(바) SVN 리포지터리 탐색 뷰에 SVN 서버의 리포지터리 추가

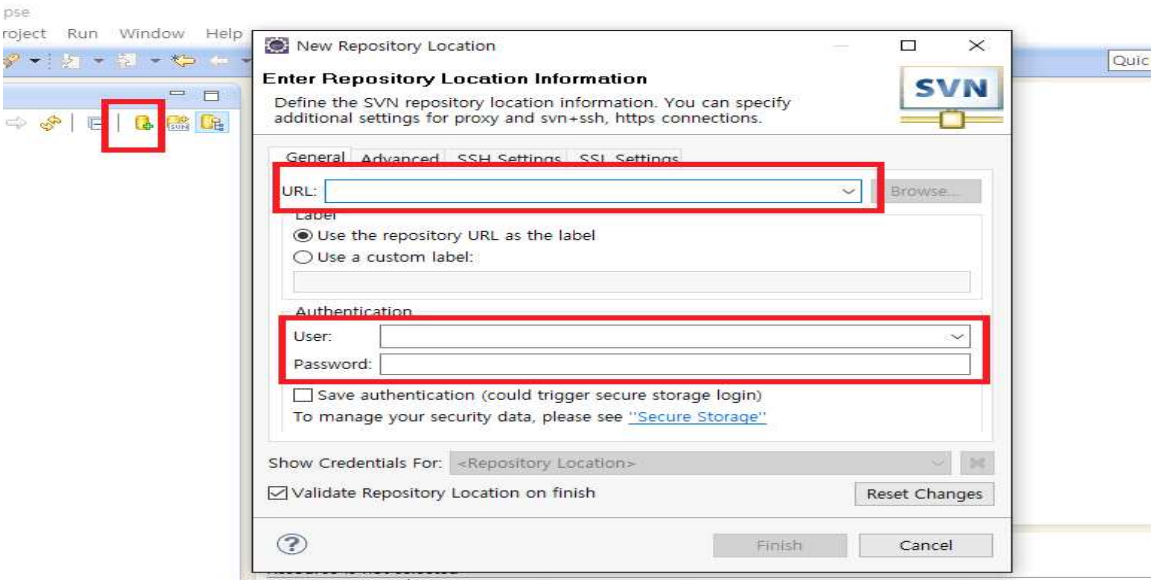


그림 2-23 SVN 리포지터리 추가

(사) 프로젝트 동기화 테스트

tip

○ 개발에 필요한 하드웨어와 소프트웨어를 파악하여 해당 명세대로 개발환경을 구축하고, 형상관리 지침에 따라 형상관리 환경을 구축할 수 있도록 한다.

1-3. Redmine 설치 및 환경설정

학습목표	• Redmine 설치 후, 환경설정을 통하여 개발환경을 구축할 수 있다.

필요 지식 /

1 Redmine 및 환경설정 하기

1. 기존에 이미 운영서버에 설치된 프로그램을 사용할 예정임으로, 프로그램 설치관련 내용은 구글 검색을 통해 진행한다. ([레드마인 설치] Windows 7/10에 Redmine 3.4.5 설치하기**)**

2. 프로젝트 진행을 위한 환경 설정하기

(1) 회원 등록

(가) 프로젝트 관리용 redmine에 접속한다.

http://112.220.114.130:83/redmine

(나) 오른쪽 상단의 등록버튼을 클릭한다.

(다) 해당 내용을 작성한다.

- 로그인용 아이디를 입력한다.

  학생 : 201912_HKD의 형태로 지정한다.

- 이름의 경우 “길동“같이 이름만 입력한다.
- 성의 경우 “홍“ 자신의 성만 입력한다.
- email의 경우 해당 알림을 받을 수 있도록 정상적으로 등록한다.

(라) 확인버튼을 클릭시 등록 대기상태가 된다.

(마) 등록확인

- 처음 등록한 교수자의 경우 권한이 없기 때문에 admin계정을 통해 권한을 부여한다.
- 왼쪽 상단의 “관리“를 클릭한다.
- 왼쪽의 “사용자“를 클릭한다.
- 검색조건의 상태를 “등록대기“를 선택후 “적용“버튼을 클릭한다.
- “활성화“를 통해 해당 계정을 등록 하여준다.
- 관리자로 변경은 활성화후 상세 정보에 왼쪽 정보 탭에 “관리자“체크박스를 이용한다.

(2) 프로젝트 등록

(가) 로그인 후 좌측상단에 ‘관리’ 탭을 선택한다.

( ) 좌측의 '프로젝트' 탭을 선택한다.

(다) 우측상단의 '새 프로젝트' 를 선택한다.

(라) 아래와 같이 작성한다.

1) 중분류 프로젝트 등록

- 프로젝트 명은 "201912_중간프로젝트"의 형태로 작성한다.
- 설명은 적지 않는다.
- 공개 여부는 클릭하지 않는다.
  (해당 프로젝트가 다른반이나 다른팀에 보여지게 된다.)
- 상위프로젝트
  : 최상위 프로젝트인 "프로젝트 관리"를 선택한다.
- 상위프로젝트의 구성원 상속 체크를 통해 상위 멤버를 상속할 수 있다.
- 하단의 "일감 유형"의 체크박스를 모두 체크한다.

2) 팀별 프로젝트 등록

- 프로젝트 명은 "201912_중간프로젝트_1팀"의 형태로 작성한다.
- 설명은 적지 않는다.
- 공개 여부는 클릭하지 않는다.
  (해당 프로젝트가 다른반이나 다른팀에 보여지게 된다.)
- 상위프로젝트
  : 팀별 프로젝트가 속애있는 상위 프로젝트를 선택한다.
  : "201912_중간프로젝트"의 형태를 선택하면 된다.
- 상위프로젝트의 구성원 상속 체크를 통해 상위 멤버를 상속할 수 있다.
- 하단의 "일감 유형"의 체크박스를 모두 체크한다.

(3) 일감 등록

(가) 자신이 속한 프로젝트를 선택한다.

(나) 상단탭에서 '일감' 탭을 선택한다.

(다) 우측 상탄의 '새 일감만들기' 버튼을 클릭한다.

(라) 아래와 같이 진행한다.

- 유형을 선택한다. (학습생들의 경우 새기능을 가장 많이 쓴다.)
- 제목을 입력한다.
- 상위 일감을 통해서 연계 관리도 가능하다.
- 시작시간과 완료기한을 정해야만 간트차트에서 확인이 가능하다.
- 해당 일감의 진척도를 통해 진척도 현황파악에 활용 할 수 있다.

학습 1	개발환경 구축
학습 2	**JCF(Java Collection Framework)**
학습 3	제너릭스와 어노테이션
학습 4	스레드
학습 5	입/출력
학습 6	네트워킹
학습 7	DB 연동 프로그래밍
학습 8	Servlet

2-1. Java Collection Framework

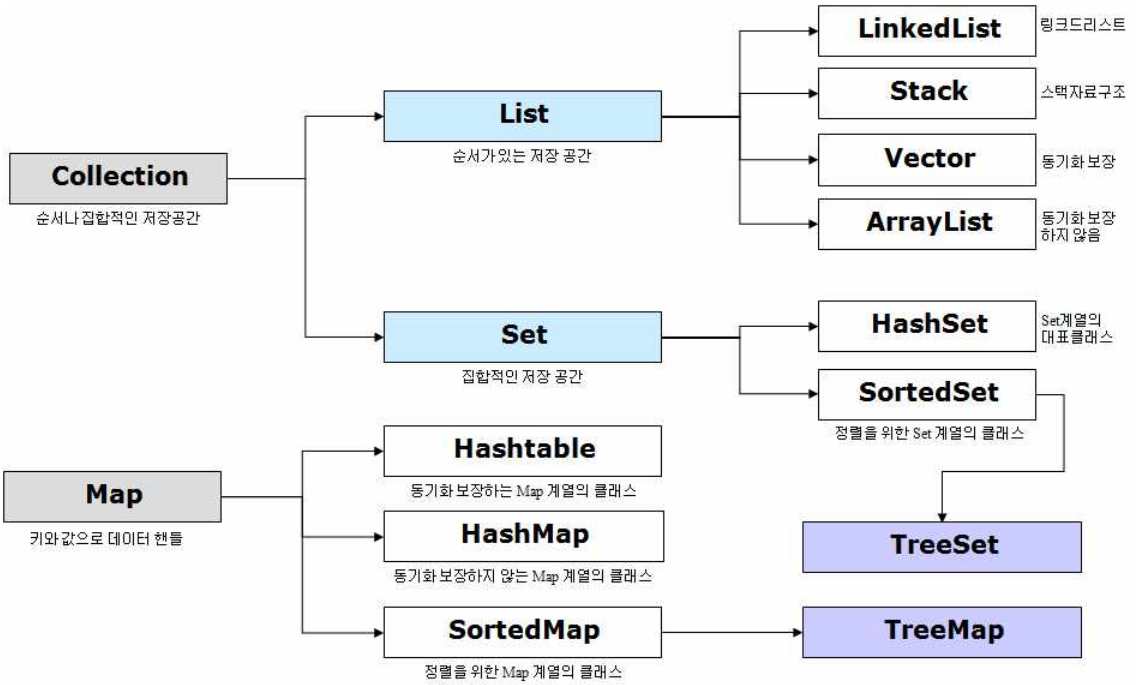
학습목표	• Collection Framework에 대해 설명하고 필요한 곳에 적절한 컬렉션 클래스를 사용할 수 있다.

필요 지식 / 프로그램(Java)언어 기본문법에 대한 이해

1 Java Collection Framework

1. 정의

Java 컬렉션 프레임워크는 자바의 컬렉션(모음)객체들을 다루기 위해 제공되는 재사용 가능한 클래스 또는 인터페이스를 통칭하는 표현임. 프레임워크(Framework)라 말하지만, 사실 라이브러리(Library)처럼 사용된다.



2 List

리턴타입	메서드	
boolean	add(E e)	마지막에 해당 항목을 추가한다.
void	add(int index, E element)	리스트의 해당 인덱스의 위치에 항목을 삽입한다.
boolean	addAll(collection<? extends E> c)	컬렉션 항목을 리스트에 추가한다.
boolean	addAll(int index, Collection<? extends E> c)	컬렉션 항목을 해당 인덱스의 위치에 삽입한다.
void	clear()	리스트의 모든 항목을 제거한다.
boolean	contains(Object o)	특정 항목을 포함하고 있는지 확인한다.
boolean	containsAll(Collection<?> c)	컬렉션 항목을 포함하고 있는지 확인한다.
boolean	equals(Object o)	리스트와 특정 객체가 동등한지 확인한다.
E	get(int index)	특정 인덱스의 항목을 가져온다.
int	hashCode()	리스트의 해쉬코드값을 가져온다.
int	indexOf(Object o)	리스트 내에서 해당 객체의 인덱스 값을 가져온다.
boolean	isEmpty()	리스트에 항목이 비어있는지 확인한다.
Iterator<E>	iterator()	리스트 항목들을 조회하기 위한 반복자를 가져온다.
int	lastIndexOf(Object o)	항목이 조회되는 마지막 인덱스 값을 가져온다.
ListIterator<E>	listIterator()	리스트의 항목에 대한 ListIterator객체를 가져온다.
ListIterator<E>	listIterator(int index)	해당 인덱스부터 시작하는 리스트항목에 대한 ListIterator 객체를 가져온다.
E	remove(int index)	해당 인덱스의 항목을 리스트에서 삭제한다.
boolean	remove(Object o)	해당 객체를 리스트에서 삭제한다.
boolean	removeAll(Collection<?> c)	컬렉션 객체를 리스트에서 모두 삭제한다.
default void	replaceAll(UnaryOperator<E> operator)	리스트의 모든 항목을 operator를 통해 바꾼다.
boolean	retainAll(Collection<?> c)	컬렉션 항목이 현재 리스트에 존재하는지 확인한다.
E	set(int index, E element)	해당 인덱스에 해당 항목으로 리스트를 수정한다.
int	size()	리스트의 사이즈를 리턴한다.
default void	sort(Comparator<? super E> c)	외부 정렬자를 이용하여 리스트를 정렬한다.
default Spliterator<E>	spliterator()	현재 리스트에 대해서 Spliterator객체를 리턴한다.
List<E>	subList(int fromIndex, int toIndex)	시작인덱스 및 끝인덱스를 이용하여 부분리스트를 조회함.
Object[]	toArray()	리스트를 배열로 변환한다.
<T> T[]	toArray(T[] a)	리스트를 T 타입의 배열로 변환한다.

< 1-1> List 인터페이스의 메서드 종류

1. List 인터페이스 특징

(1) (인덱스)가 존재하는 데이터의 집합이다.

(2) 데이터가 중복되어도 저장이 가능하다.(순서만 다르면 중복저장 가능함.)

2. List 인터페이스를 구현하고 있는 클래스

Stack, Vector, LinkedList, ArrayList 등

3. 실습 예제

```
public static void main(String[] args) {
    // ArrayList는 기본적인 사용법이 Vector와 같다.
    // DEFAULT_CAPACITY = 10;
    List list1 = new ArrayList();

    // add()메서드를 사용해서 데이터를 추가한다.
    list1.add("aaa");
    list1.add("bbb");
    list1.add(111);
    list1.add('k');
    list1.add(true);
    list1.add(12.34);

    // size() ==> 데이터 개수
    System.out.println("size => " + list1.size());
    System.out.println("list1 => " + list1);

    // get으로 데이터 꺼내오기
    System.out.println("1번째 자료 : " + list1.get(1));

    // 데이터 끼워 넣기도 같다.
    list1.add(0, "zzz");
    System.out.println("list1 => " + list1);

    // 데이터 변경하기 (set메서드)
    String temp = (String) list1.set(0, "YYY");
    System.out.println("temp => " + temp);
    System.out.println("list1 => " + list1);

    // 삭제하기도 같다.
    list1.remove(0);
    System.out.println("삭제후 : " + list1);

}
```

3 List

1. List 정렬에 대하여

관련된 interface는 Comparable과 Comparator 이렇게 두가지가 있다.

- 보통 객체 자체에 정렬 기능을 넣기 위해서는 Comparable을 구현하고
  위의 예제처럼 정렬 기준을 별도로 구현하고 싶을 때는 Comparator를
  구현하여 사용하면 된다.

- Comparable에서는 compareTo()메서드를 구현해야 하고,
  Comparator에서는 compare()메서드를 구현해야 한다.

2. Comparable 인터페이스

int compareTo(T o) : 현재객체(this)와 대상객체(o)의 순서를 비교한다.
ex) a.compareTo(b);
* 결과값 : 양수(현재객체가 대상객체보다 순서 값이 작은 경우)
0 (현재객체가 대상객체와 순서 값이 동일한 경우)
음수(현재객체가 대상객체보다 순서 값이 큰 경우)

3. Comparable 구현 예제

```
// 회원이름을 기준으로 오름차순 정렬이 될 수 있는 클래스 만들기
class Member implements Comparable<Member>{

    private int num;     // 번호
    private String name; // 이름
    private String tel;  // 전화번호

    public Member(int num, String name, String tel) {
        super();
        this.num = num;
        this.name = name;
        this.tel = tel;
    }

    // 이름을 기준으로 오름차순 정렬이 되도록 설정한다.
    @Override
    public int compareTo(Member mem) {
        return getName().compareTo(mem.getName());
    }
```

4. Comparator 인터페이스

int compare(T o1, T o2) : 첫 번째 객체(o1)와 두 번째 객체(o2)의 순서를 비교한다.
ex) compare(a, b);
* 결과값 : 양수(첫 번째 객체가 두 번째 객체보다 순서 값이 작은 경우)
0 (첫 번째 객체가 두 번째 객체와 순서 값이 동일한 경우)
음수(첫 번째 객체가 첫 번째 객체보다 순서 값이 큰 경우)

5. Comparator 구현 예제

```
// Member의 번호(num)의 내림차순으로 정렬하기
class SortNumDesc implements Comparator<Member>{
    @Override
    public int compare(Member mem1, Member mem2) {
        /*
        if(mem1.getNum() > mem2.getNum()){
            return -1;
        }else if(mem1.getNum() == mem2.getNum()){
            return 0;
        }else{
            return 1;
        }
        */

        // Wrapper클래스에서 제공하는 메서드를 이용하는 방법1
        // 내림차순일 경우에는 -1을 곱해준다.
        //return Integer.compare(mem1.getNum(), mem2.getNum()) * -1 ;

        // Wrapper클래스에서 제공하는 메서드를 이용하는 방법2
        return new Integer(mem1.getNum()).compareTo(mem2.getNum()) * -1;
    }
}
```

4 Set

리턴타입	메서드	
boolean	add(E e)	Set 마지막에 해당 항목을 추가한다.
boolean	addAll(collection<? extends E> c)	컬렉션 항목을 해당 인덱스의 위치에 삽입한다.
void	clear()	Set의 모든 항목을 제거한다.
boolean	contains(Object o)	특정 항목을 포함하고 있는지 확인한다.
boolean	containsAll(Collection<?> c)	컬렉션 항목을 포함하고 있는지 확인한다.
boolean	equals(Object o)	Set의 특정 객체가 동등한지 확인한다.
int	hashCode()	Set의 해쉬 코드 값을 가져온다.
boolean	isEmpty()	Set에 항목이 비어있는지 확인한다.
Iterator<E>	iterator()	Set의 항목에 대한 ListIterator객체를 가져온다.
boolean	remove(Object o)	해당 객체를 Set에서 삭제한다.
boolean	removeAll(Collection<?> c)	컬렉션 객체를 Set에서 모두 삭제한다.
boolean	retainAll(Collection<?> c)	컬렉션 항목이 현재 Set에 존재하는지 확인한다.
int	size()	Set의 사이즈를 리턴한다.
Spliterator<E>	spliterator()	현재 Set에 대해서 Spliterator객체를 리턴한다.
Object[]	toArray()	Set을 배열로 변환한다.
<T> T[]	toArray(T[] a)	Set을 T 타입의 배열로 변환한다.

< 1-1> Set 인터페이스의 메서드 종류

1. Set 인터페이스 특징

(1) 순서(인덱스)가 유지되지 않는 데이터의 집합이다.

(2) 데이터 중복을 허용하지 않는다. (데이터를 등록할 때 중복된 데이터가 있는지 확인 후 없으면 등록한다.)

2. Set 인터페이스를 구현하고 있는 클래스

SortedSet, TreeSet, HashSet

3. 실습 예제

```
public static void main(String[] args) {

    Set hs1 = new HashSet();

    // Set에 데이터를 추가할 때도 add()메서드를 사용한다.
    hs1.add("DD");
    hs1.add("AA");
    hs1.add(2);
    hs1.add("CC");
    hs1.add("BB");
    hs1.add(1);
    hs1.add(3);

    System.out.println("Set 데이터 : " + hs1);
    System.out.println();

    boolean isAdd = hs1.add("FF");
    System.out.println("중복되지 않을 때 : " + isAdd);
    System.out.println("Set 데이터 : " + hs1);
    System.out.println();

    isAdd = hs1.add("CC");
    System.out.println("중복될 때 : " + isAdd);
    System.out.println("Set 데이터 : " + hs1);
    System.out.println();

    // 예) 'FF'를 'EE'로 수정하기
    hs1.remove("FF");   // FF자료 삭제
    System.out.println("삭제 후 Set 데이터 : " + hs1);
    System.out.println();

    hs1.add("EE");  // EE자료 추가
    System.out.println("Set 데이터 : " + hs1);
    System.out.println();

    //hs1.clear();  // 전체 자료 삭제
    //System.out.println("clear 후 Set 데이터 : " + hs1);
    System.out.println("Set의 자료 개수 : " + hs1.size());
    System.out.println();

    // Set의 데이터를 Iterator로 변환하기 ==> Set의 iterator()메서드를 호출하면 된다.
    Iterator it = hs1.iterator();

    while(it.hasNext()){   // 다음 자료가 있는지 검사
     // next()메서드 ==> 포인터를 다음 자료 위치로 이동하고, 이동한 위치의 자료를 반환한다.
        System.out.println(it.next());
    }
}
```

수 hashCode(), equals()

1. 객체의 동등 비교 방법에 대하여

HashSet, HashMap, Hashtable 같은 객체들을 사용할 경우
객체가 서로 같은지를 비교하기 위해 equals()메서드와 hashCode()메서드를 호출한다.
그래서 객체가 서로 같은지 여부를 결정하려면 두 메서드를 재정의 해야 한다.
HashSet, HashMap, Hashtable에서는 객체가 같은지 여부는 데이터를 추가할 때 검사한다.

- equals()메서드는 두 객체의 내용(값)이 같은지 비교하는 메서드 이고
- hashCode()메서드는 두 객체가 같은 객체인지를 비교하는 메서드 이다.

2. equals()메서드와 hashCode()메서드 구현에 대하여

(1) 두 객체가 같으면 반드시 같은 hashcode를 가져야 한다.

(2) 두 객체가 같으면 equals()메서드를 호출했을 때 true를 반환해야 한다.

즉, 객체 a, b가 같다면 a.equals(b)와 b.equals(a) 둘 다 true이여야 한다.

(3) 두 객체의 hashcode가 같다고 해서 두 객체가 반드시 같은 객체는 아니다.

하지만, 두 객체가 같으면 반드시 hashcode 값이 같아야 한다.

(4) equals()메서드를 override하면 반드시 hashCode() 메서드도 override해야 한다.

(5) hashCode()는 기본적으로 Heap에 있는 각 객체에 대한 메모리 주소 매핑 정보를

기반으로 한 정수값을 반환한다. 그러므로, 클래스에서 hashCode()메서드를 override하지 않으면 절대로 두 객체가 같은 것으로 간주될 수 없다.

3. hashCode() 의 구현예제

```
    @Override
    public int hashCode() {
    // id와 name 멤버변수를 이용한 hashCode()메서드 구현하기
        final int prime = 31;
        int result = 1;

        result = prime * result + (name==null ? 0 : name.hashCode());
        result = prime * result + id;

        return result;
    }
}
```

4. equals()의 구현예제

```
    @Override
    public boolean equals(Object obj) {

        if(obj==null){
            return false;
        }
        if(this.getClass() != obj.getClass()){
            return false;
        }
        if(this==obj){
            return true;
        }
        Person test = (Person) obj;
        if(this.name==null && test.name != null){
            return false;
        }
        if(this.id==test.id && this.name.equals(test.name)){
            return true;
        }

        return false;
    }
```

6 Map

리턴타입	메서드	
void	clear()	모든 항목을 삭제한다.
boolean	containsKey(Object key)	키에 매핑된 항목이 존재하는지 확인한다.
boolean	containsValue(Object value)	특정 값에 매핑되어 있는 항목이 존재하는지 확인한다.
set<Map.Entry<K,V>>	entrySet()	엔트리타입의 Set객체를 가져온다.
boolean	equals(Object o)	맵 객체와 동등한 객체인지 비교한다.
V	get(Object key)	키에 매핑되어 있는 값을 가져온다.
default V	getOrDefault(Object key, V defaultValue)	키에 매핑되어 있는 값을 가져오고, 없으면 기본값을 가져온다.
int	hashCode()	현재 맵의 해시 코드값을 가져온다.
boolean	isEmpty()	현재 맵의 비어 있는지 확인한다.
Set<K>	keySet()	현재 맵의 key값으로 이루어진 Set객체를 가져온다.
V	put(K key, V value)	맵에 key와 value 값을 등록한다.
void	putAll(Map<? extends K, ? extends V> m	특정 맵의 데이터를 현재 맵으로 모두 복사한다.
default V	putIfAbsent(K key, V value)	key에 해당하는 데이터가 없으면 맵에 등록한다.
V	remove(Object key)	key에 해당하는 데이터를 맵에서 지운다.
default boolean	remove(Object key, Object value)	key와 value값에 일치하는 데이터를 제거한다.
default V	replace(K key, V value)	key에 매핑하는 값을 새로운 값으로 바꾼다.
default boolean	replace(K key, V oldValue, V newValue)	현재 key에 매핑하는 값이 old값이면 new값으로 바꾼다.
int	size()	맵의 데이터 사이즈를 리턴한다.
Collection<V>	values()	맵의 값으로 이루어진 컬렉션 뷰를 리턴한다.

< 1-1> Map 인터페이스의 메서드 종류

1. Map 인터페이스 특징

(1) key값과 value값을 한 쌍(Entry)의 데이터로 관리한다.

(2) key값은 중복을 허용하지 않고, 순서를 유지하지 않는다.(Set의 특징)

(3) value값은 중복을 허용한다.(List의 특징)

2. Map 인터페이스를 구현하고 있는 클래스

Hashtable, HashMap, SortedMap, TreeMap

3. 실습 예제

```
public static void main(String[] args) {

    Map<String, String> map = new HashMap<String, String>();

    // 자료 추가 ==> put(key값, value값);
    map.put("name", "홍길동");
    map.put("addr", "대전");
    map.put("tel", "010-1234-5678");

    System.out.println("map => " + map);

    //  자료 수정 ==> 데이터를 저장할 때 key값이 같으면 나중에 입력한 값이 저장된다.
    //           ==> put(수정할key값, 새로운value값)
    map.put("addr", "서울");
    System.out.println("map => " + map);

    // 자료 삭제 ==> remove(삭제할key값);
    map.remove("name");
    System.out.println("map => " + map);

    // 자료 읽기 ==> get(key값);
    System.out.println("addr = " + map.get("addr"));
    System.out.println("=============================================");

    // key값들을 읽어와 자료를 출력하는 방법

    // 방법1 ==> keySet()메서드 이용하기
    //    keySet()메서드 ==> Map의 key값들만 읽어와 Set형으로 반환한다.
    Set<String> keySet = map.keySet();

    System.out.println("Iterator를 이용한 방법");

    Iterator<String> it = keySet.iterator();

    while(it.hasNext()){
        String key = it.next();
        System.out.println(key + " : " + map.get(key));
    }

    System.out.println("---------------------------------------------");

    ...
}
```

학습 1	개발환경 구축
학습 2	JCF(Java Collection Framework)
학습 3	**제너릭스와 어노테이션**
학습 4	스레드
학습 5	입/출력
학습 6	네트워킹
학습 7	DB 연동 프로그래밍
학습 8	Servlet

3-1. Generics

학습목표	• Generics 대해서 설명할 수 있다.

필요 지식 /

1 Generics

1. Generic이란 ?

클래스에 사용할 타입을 디자인(설계)시에 지정하는 것이 아니라 클래스를 사용할 때 지정한 후 사용하는 기술을 말한다.

2. Generic 사용시 장점

(1) 컴파일시 잘못된 타입 사용을 체크하기 때문에 타입 안전한(Type Safty) 코딩을 할 수 있다.

(2) 불필요한 타입변환(casting)을 하지 않아도 된다. (프로그램 성능 향상됨.)

3. Generic 클래스 선언 방법

```
class 클래스명<제네릭타입글자>{
    제네릭타입글자 변수명;   // 변수 선언에 제네릭을 사용할 경우
    ...

    제네릭타입글자 메서드명(){  // 반환값이 있는 메서드에서 사용
        ...
        return 값;
    }
    ...
}
```

```
-- 제네릭타입글자 --
T  ==> Type
K  ==> Key
V  ==> Value
E  ==> Elment(자료구조에 들어가는 것들을 나타낼 때 사용)
```

4. Generic 메서드 선언 방법

```
제너릭 메서드<T, R> R method(T t)
파라미터 타입과 리턴타입으로 타입파라미터를 가지는 메서드
선언방법: 리턴타입 앞에<> 기호를추가하고 타입파라미터를 기술 후 사용함.
```

5. 제한된 타입 파라미터(Bounded Type Parameter)

Generic 이용한 타입(문자)에 대해 구체적으로 타입을 제한할 경우에 사용함.

```
<T extends 제한타입> => 제너릭타입의 상한 제한. 제한타입과 그 자손(타입)들만 가능.
```

6. 와일드카드(Wildcard)에 대하여

제너릭이 사용된 객체를 참조할 때 참조할 객체의 타입을 제한하기 위해 사용한다.

```
<? extends T> => 와일드 카드의 상한 제한. T와 그 자손들만 가능
<? super T>   => 와일드 카드의 하한 제한. T와 그 조상들만 가능(Object 제외.)
<?>           => 모든타입이 가능 <? extends Object>와 동일
```

7. 실습 예제

```
  public class N05WildCardTest {
    public static void main(String[] args) {

        FruitBox<Fruit> fruitBox = new FruitBox<>(); // 과일상자
        FruitBox<Apple> appleBox = new FruitBox<>(); // 사과상자

        fruitBox.add(new Apple());
        fruitBox.add(new Grape());

        appleBox.add(new Apple());
        appleBox.add(new Apple());

        Juicer.makeJuice(fruitBox); // 메서드 호출
    }
}

/*
 * 쥬서
 */
class Juicer{
    static void makeJuice(FruitBox<Fruit> box) {
     // 제너릭 타입 객체를 파라미터에 사용시 문제점 발생함.
     // 지네릭 타입이 다른 것만으로는 오버로딩이 성립하지 않음
     //    => 컴파일 후 제거됨.=> 메서드 중복정의
    //static void makeJuice(FruitBox<? extends Fruit> box) {
     // 방법1) 와일드카드를 이용하여 제너릭 타입의 객체 참조
    //static <T extends Fruit> void makeJuice(FruitBox<T> box) {
   // 방법2) 지네릭 메서드(제한된 타입 파라미터)로 선언함.

        String fruitListStr = ""; // 과일목록

        int cnt = 0;
        for(Fruit f : box.getFruitList()){
            if(cnt == 0) {
                fruitListStr += f;
            }else {
                fruitListStr += "," + f;
            }
            cnt++;
        }

        System.out.println(fruitListStr + "=> 쥬스완성!");
    }
}
```

3-2. enums

학습목표	• enum( )에 대해서 설명할 수 있다.

필요 지식 /

1 enum ( )

1. enum 이란?

enum type 은 상수(Constant)로 사용 할 값들을 미리 선언하여 사용할 때 사용하는 특별한 데이터 타입이다. (JDK1.5부터 지원됨.)

2. enum 사용시 장점

(1) 기존 static final 키워드를 이용한 상수선언 방식에서는 해당 상수의 값만을 비교하게 되는반면, enum을 이용하여 상수를 정의하게 되면 값 뿐만 아니라 타입까지도 체크하기 때문에 타입 안전(Type Safe)한 코드를 작성할 수 있다.

(2) 기존 상수 선언방식을 사용하는 코드에서는 상수값이 변경되는 경우, 해당 상수를 참조하는 소스코드도 모두 재컴파일을 해줘야 하는데 enum 상수를 정의하여 사용하는 경우에는 그럴 필요가 없다.

3. enum 상수 선언하는 방법

```
enum 열거형이름 { 상수값1, 상수값2, ...., 상수값n };
```

4. 실습 예제

```
// City 열거형 객체 선언 (기본값을 이용하는 열거형)
    public enum City { 서울, 부산, 대구, 광주, 대전};

   // 데이터값을 임의로 지정한 열거형 객체 선언
   // 데이터값을 정해줄 경우에는 생성자를 만들어서 괄호속의 값이 변수에 저장되도록 해야한다.
    public enum Season{
        봄("3월부터 5월까지"), 여름("6월부터 8월까지"), 가을("9월부터 11월까지"),
       겨울("12월부터 2월까지");

        // 괄호속의 값이 저장될 변수 선언
        private String str;

        // 생성자 만들기(열거형의 생성자는 제어자가 묵시적으로 'private' 이다.)
        Season(String data){  // ==> private Season(String data){ 와 같다.
            str = data;
        }

        // 값을 반환하는 메서드 작성
        public String getStr(){
            return str;
        }

    }
```

3-3. Annotation

학습목표	• Annotation 대해서 설명할 수 있다.

필요 지식 /

1 Annotation

1. Annotation 이란?

자바 소스코드 안에서 컴파일러나 또는 다른 프로그램이 사용할 수 있도록 유용한 정보를 정해진 표기법에 따라 표기하기 위한 일종의 메타데이터 라고 할 수 있다.
또한, 주석처럼 프로그램에 영향을 미치지 않으면서도 다른 프로그램에게 유용한 정보를 제공한다.

2. Annotation 종류

	표준(내장) 어노테이션	메타 어노테이션
정의	메타 어노테이션을 제외한 일반적인 어노테이션을 말함	어노테이션을 정의할 때 사용되는 어노테이션.
종류	@Override, @SuppressWarnings @Deprecated, @Repeatable(1.7) @FunctionalInterface(1.7) 등	@Documented, @Target, @Retention, @Inherited 등

3. Annotation 타입 정의하기

```
@interface 애너테이션이름{
    반환타입 타입요소이름();  // 반환값이 있고 매개변수는 없는 추상메서드의 형태
            ...
}
```

4. Annotation 타입 요소 작성시 유의 사항

(1) 요소의 타입은 Primative(기본형), String, enum, annotation, class 만 허용함.

(2) 괄호안에 매개변수를 선언할 수 없다. (메서드가 아님에 주의)

(3) 선언할 수 없다.(메서드가 아님)

(4) 요소의 타입을 표현시, 제너릭 타입문자를 사용 할 수 없다.

5. Annotation 예제

(1) 표준(내장) 어노테이션

```
public class Animal {
    public void speak() {
    }

    public String getType() {
        return "Generic animal";
    }
}

public class Cat extends Animal {
    @Override
    public void speak() { // 오버라이드 메서드
        System.out.println("Meow.");
    }

    @Override
    public String gettype() { // 컴파일시 에러 발생(오버라이드 아님)
        return "Cat";
    }
}
```

(2) 메타 어노테이션

```
@Target(ElementType.METHOD)          // annotation이 적용가능한 대상을 지정함.
@Retention(RetentionPolicy.RUNTIME)       // annotation이 유지되는 기간을 지정
                                          // (SOURCE, CLASS:기본값, RUNTIME)
public @interface CustomAnnotation {
    int id = 100; // 상수선언 가능. static final int id = 100;
    String value() default "-"; // 기본값을 '-'로 지정
    int count() default 20;  //  기본값을 20으로 지정
}
```

4-1. Process와 Thread

학습목표
- Process Thread 에 대해서 설명할 수 있다.

필요 지식 /

1 Process

1. Process 란?

프로세스란 운영체제에서 실행 중인 하나의 프로그램을 의미한다.

2. Multi Process 란?

멀티 프로세스(Multi Process)란 두 개 이상의 프로세스가 실행되는 것을 의미함.

3. Multi Tasking 란?

멀티 태스킹(Multi Tasking)이란 두 개 이상의 프로세스를 실행하여 일을 처리하는 것을 의미함.

2 Thread

1. Thread에 대하여

- 스레드(Thread)란 프로그램의 실행 흐름을 의미함.
- 프로세스 내에서 실행되는 세부 작업 단위이다.
- 하나의 프로세스 내에는 여러 개의 쓰레드가 존재할 수 있다.
- 경량화(lightweight) 프로세스라고도 부른다.
- 두 개 이상의 스레드를 멀티스레드라고 부른다.

2. Thread의 특징 (VS Process)

(1) 프로세스에 비해 문맥교환(Context Switching) 시간이 적게 걸린다.

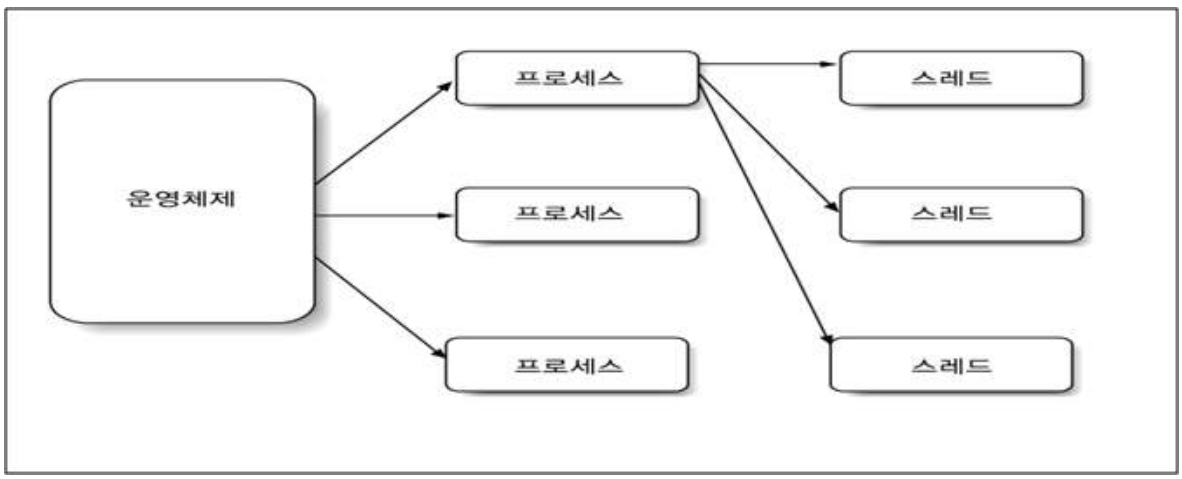
(2) 스레드는 동료 쓰레드와 사용메모리를 공유할 수 있다.

(3) 스레드간의 통신은 프로세스간의 통신에 비해 시간이 적게 걸린다(빠르다).

(4) 스레드는 프로세스에 비해 생성 및 종료 시간이 적게 걸린다(빠르다).

(5) 스레드는 동료 쓰레드와 사용메모리를 공유할 수 있다.

3. Process와 Thread의 관계



4-2. Single and Multi Thread

학습목표	• Single Thread Multi Thread에 대해서 구분하여 설명할 수 있다.

필요 지식 /

1 Single Thread and Multi Thread

1. Single Thread Program

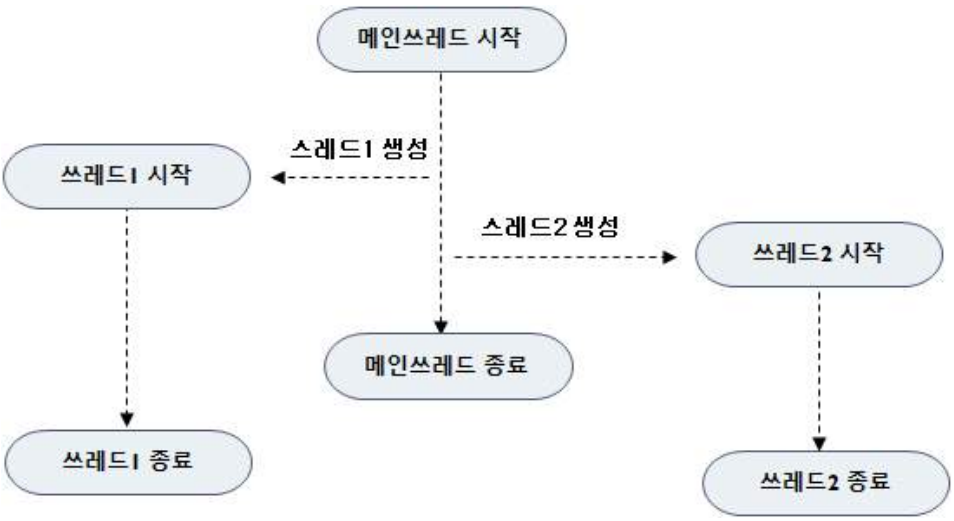
Thread가 하나뿐인 프로그램을 의미한다.

2. Multi Threaded Program

Thread가 2개 이상인 프로그램을 의미한다.

2 Multi Thread Programming

1. Multi Thread의 실행 흐름



2. Multi Thread 프로그램 작성방법

(1) Thread 상속하여 만든 클래스를 이용하여 Thread를 생성하는 방법

```
class MyThread1 extends Thread{
    @Override
    public void run() {
        for(int i=1; i<=200; i++){
            System.out.println("*");
            try {
                // Thread.sleep(시간) ==> 주어진 시간동안 작업을 잠시 멈춘다.
                // 시간은 밀리세컨드 단위를 사용한다.
                // 즉, 1000은 1초를 의미한다.
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

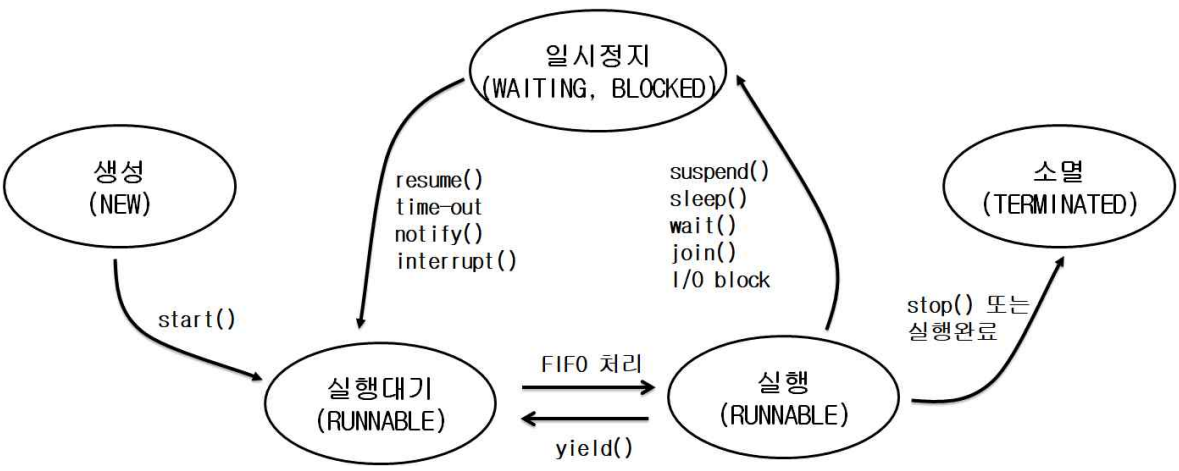
(2) Runnable 인터페이스를 구현한 클래스의 인스턴스를 생성하여 Thread를 생성하는 방법

```
class MyThread2 implements Runnable{
    @Override
    public void run() {
        for(int i=1; i<=200; i++){
            System.out.println("$");
            try {
                // Thread.sleep(시간) ==> 주어진 시간동안 작업을 잠시 멈춘다.
                // 시간은 밀리세컨드 단위를 사용한다.
                // 즉, 1000은 1초를 의미한다.
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

(3) Runnable 구현한 익명클래스를 이용하여 Thread를 생성하는 방법

```
Thread th3 = new Thread(
                new Runnable(){
                    @Override
                    public void run() {
                        for(int i=1; i<=200; i++){
                            System.out.println("@");
                            try {
                                Thread.sleep(100);
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
            );
```

3. Thread의 생명 주기



	설명
NEW	생성되고 아직 start()가 호출되지 않은 상태
RUNNABLE	실행 중 또는 실행 가능한 상태
BLOCKED	동기화 블록에 의해서 일시 정지된 상태 (Lock이 풀릴때까지 기다리는 상태)
WAITING, TIMED_WAITING	스레드의 작업이 종료되지는 않았지만 실행가능하지 않은(UNRUNNABLE) 일시정지 상태. TIMED_WAITING은 일시정지 시간이 지정된 경우임.
TERMINATED	스레드의 작업이 종료된 상태

4. 실습 예제

```
class StatePrintThread extends Thread{
    private Thread targetThread;  // 상태를 출력할 쓰레드가 저장될 변수

    public StatePrintThread(Thread targetThread) {
        this.targetThread = targetThread;
    }
    @Override
    public void run() {
        while(true){
            // Thread의 상태 구하기 (getState()메서드 이용)
            Thread.State state = targetThread.getState();
            System.out.println("타겟 쓰레드의 상태값 : " + state);

            // NEW상태인지 검사
            if(state == Thread.State.NEW){
                targetThread.start();
            }

            // 타겟 쓰레드가 종료 상태인지 검사
            if(state == Thread.State.TERMINATED){
                break;
            }

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4-3. Daemon Thread

학습목표	• Daemon Thread 대해서 설명할 수 있다.

필요 지식 / Thread 대한 이해

1 Daemon Thread ?

1. Daemon Thread

데몬 스레드는 다른 일반 스레드(데몬 스레드가 아닌 스레드)의 작업을 돕는 보조적인 역할을 수행하는 스레드 이다. 그러므로, 일반 스레드가 모두 종료되면 데몬 스레드는 자동으로 종료된다.

2. Daemon Thread 작성방법

데몬 스레드로 설정하기 위해서는 반드시 실행 전(start메서드 호출 전)에 설정해야 한다.

```
// 데몬 스레드로 설정하기 (start()메서드 호출하기 전에 설정한다.)
스레드객체.setDaemon(true);
스레드객체.start();
```

3. Daemon Thread 적용 예제

```
public class ThreadDeamonTest {
    public static void main(String[] args) {
        AutoSaveThread autoSave = new AutoSaveThread();

        // 데몬 쓰레드로 설정하기 (start()메서드 호출하기 전에 설정한다.)
        autoSave.setDaemon(true);
        autoSave.start();

        try {
            for(int i=1; i<=20; i++){
                System.out.println("작업 " + i);
                    Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("메인 쓰레드 종료...");
    }
}

/**
 * 자동 저장하는 쓰레드(3초에 한번씩 저장하기)
 */
class AutoSaveThread extends Thread{
    public void save(){
        System.out.println("작업 내용을 저장합니다...");
    }

    @Override
    public void run() {
        while(true){
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            save(); // 저장기능 호출
        }
    }
}
```

4-4. 스레드 동기화

학습목표	• 동기화에 대해서 설명할 수 있다.

필요 지식 /

1 (Synchronization)

1. 동기화란?

공유자원(객체)을 상대로 순서대로 작업이 이루어지도록 처리하는 방법을 의미한다.

2. 동기화가 필요한 이유

프로세스 내의 자원을 여러 개의 스레드가 공유하여 작업을 진행하다 보면 예기치 못한 결과를 초래할 수 있는데, 이런 경우에 문제가 발생할 여지가 있을 법한 영역인 임계영역(Critical Section)에 동기화 처리를 해 줌으로써 문제발생을 방지할 수 있다. 하지만 과도한 동기화는 전체적으로 프로그램의 성능을 저하시키기 때문에 필요한 영역만 선별적으로 사용하여야 한다.

3. 동기화 처리 방법

(1) Synchroized 를 이용한 동기화

(가) 메서드 자체에 동기화 설정하기

```
class ShareObject{
    private int sum = 0;

    // 동기화 하는 방법 1 : 메서드 자체에 동기화 설정하기
    public synchronized void add(){
        for(int i=0; i<1000000000; i++) {} // 동기화처리 전까지의 시간벌기용

            int n = sum;
            n += 10; // 10 증가
            sum = n;
            System.out.println(Thread.currentThread().getName() + "합계: " + sum);
    }

    public int getSum(){
        return sum;
    }
}
```

( ) 메서드 내 동기화 블록 설정하기

```
class ShareObject{
    private int sum = 0;

    public void add(){

        for(int i=0; i<1000000000; i++) {} // 동기화처리 전까지의 시간벌기용

        // 동기화 하는 방법2 : 동기화 블럭으로 설정하기
        synchronized (this) {
            int n = sum;
            n += 10; // 10 증가
            sum = n;
            System.out.println(Thread.currentThread().getName()
                                + "합계: " + sum);
        }
    }

    public int getSum(){
        return sum;
    }
}
```

(2) Lock 객체를 이용한 동기화

(가) java.util.concurrent.locks 패키지가 제공하는 Lock클래스를 이용하는 방법

```
class ShareObject{
    // lock객체 생성 ==> 되도록이면 private final로 만든다.
    private final ReentrantLock lock = new ReentrantLock();

    private int sum = 0;

    public void add(){

        lock.lock(); // 락 설정

        for(int i=0; i<1000000000; i++) {} // 동기화처리 전까지의 시간벌기용

        int n = sum;
        n += 10; // 10 증가
        sum = n;
        System.out.println(Thread.currentThread().getName() + "합계: " + sum);

        lock.unlock(); // 락 해제
    }
}
```

학습 1	개발환경 구축
학습 2	JCF(Java Collection Framework)
학습 3	제너릭스와 어노테이션
학습 4	스레드
학습 5	**입/출력**
학습 6	네트워킹
학습 7	DB 연동 프로그래밍
학습 8	Servlet

5-1. 바이트기반 스트림

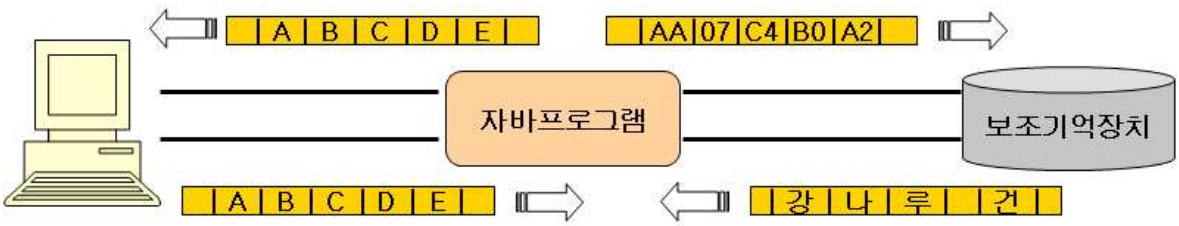
학습목표	• 스트림에 대해서 설명할 수 있다.

필요 지식 / 프로그램(Java)언어 기본문법에 대한 이해

1 대하여

1. 스트림(Stream)

- 일차원적인 데이터의 흐름을 의미.
- 데이터를 목적지로 입출력 하기 위한 방법임.
- 자바의 스트림객체를 이용하여 스트림 형식으로 데이터를 읽고 쓸 수 있다.
- 스트림 형식으로 데이터를 읽기 위해서 입력스트림(InputStream)을 사용한다.
- 스트림 형식으로 데이터를 쓰기 위해서 출력스트림(OutputStream)을 사용한다.



스트림 형태로 이루어지는 입력과 출력

2 기반 스트림

1. 바이트기반 스트림

- 스트림은 1 byte를 입출력 할 수 있는 스트림이다.
- 일반적으로 바이트로 구성된 파일, 즉 동영상 파일, 이미지 파일, 음악 파일을 처리하기에 적합한 스트림이다.
- InputStream는 입력용, OutputStream는 출력용 바이트 스트림이다.
- 'InputStream'나 'OutputStream'이라는 단어가 붙어 있다면 바이트 스트림이다.

2. 바이트기반 스트림 종류

구분	스트림명	설명
InputStream	BufferedInputStream	Reader 버퍼기능을 제공하는 보조 스트림, 라인단위 읽기가능
	LineNumberInputStream	Reader스트림에 버퍼기능을 제공하는 보조 스트림, 라인 번호를 유지
	ByteArrayInputStream	문자배열로부터 문자를 읽기 위한 스트림
	FileInputStream	파일로부터 바이트를 읽을 때 문자 단위 스트림으로 처리해 주는 스트림
	FilterInputStream	필터(기능) 적용을 위한 추상클래스
	PushbackInputStream	읽어들인 문자를 되돌리는(pushback) 기능을 제공하는 스트림
	PipedInputStream	파이프(Pipe) 기능을 이용한 스트림 처리를 위한 기능을 제공하는 스트림
	StringBufferInputStream	문자열로부터 문자를 읽기위한 스트림
OutputStream	BufferedOutputStream	Writer스트림에 버퍼기능을 제공하는 보조 스트림
	ByteArrayOutputStream	문자배열에 문자를 쓰기 위한 스트림
	FilterOutputStream	파일에 문자 단위 스트림으로 쓰는 기능을 제공해 주는 스트림
	FileOutputStream	파일에 데이터를 쓸 때 문자 단위 스트림으로 처리해 주는 스트림
	PrintStream	Writer스트림에 다양한 타입의 데이터 출력 기능 제공하는 보조스트림
	PipedOutputStream	PipedReader에 출력기능 제공하는 스트림

< 1-1> 바이트기반 스트림 종류

3 바이트기반 스트림 예제

(1) ByteArrayInputStream ByteArrayOutputStream 예제

```
public static void main(String[] args) {
    byte[] inSrc = {0,1,2,3,4,5,6,7,8,9};
    byte[] outSrc = null;

    // 스트림 선언 및 객체 생성
    ByteArrayInputStream input = null;  // 스트림 선언
    input = new ByteArrayInputStream(inSrc);  // 객체 생성

    ByteArrayOutputStream output = new ByteArrayOutputStream();

    int data;   // 읽어온 자료를 저장할 변수

    // read()메서드 ==> byte단위로 자료를 읽어와 int형으로 반환한다.
    //              ==> 더 이상 읽어올 자료가 없으면 -1을 반환한다.
    while((data=input.read())!=-1){
        output.write(data);  // 출력하기
    }

    // 출력된 스트림 값들을 배열로 변환해서 반환하는 메서드
    outSrc = output.toByteArray();

    System.out.println("inSrc  ==> " + Arrays.toString(inSrc));
    System.out.println("outSrc ==> " + Arrays.toString(outSrc));

    try {
        input.close();
        output.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

(2) 사용하는 ByteArrayInputStream 및 ByteArrayOutputStream 예제

```
public static void main(String[] args) {
    byte[] inSrc = {0,1,2,3,4,5,6,7,8,9};
    byte[] outSrc = null;

    byte[] temp = new byte[4];  // 자료를 읽을 때 사용할 배열

    ByteArrayInputStream input = new ByteArrayInputStream(inSrc);
    ByteArrayOutputStream output = new ByteArrayOutputStream();

    try {

        // available() ==> 읽어 올 수 있는 byte수를 반환
        while(input.available()>0){
            /*
            input.read(temp);// temp배열 크기만큼 자료를 읽어와 temp배열에 저장한다.
            output.write(temp);  // temp배열의 내용을 출력한다.
            */

            // 실제 읽어온 byte수를 반환한다.
            int len = input.read(temp);

            // temp배열의 내용 중에서 0번째 부터 len개수만큼 출력한다.
            output.write(temp, 0, len);

            System.out.println("temp : " + Arrays.toString(temp));
        } // while문

        System.out.println();
        outSrc = output.toByteArray();
        System.out.println("inSrc  ==> " + Arrays.toString(inSrc));
        System.out.println("outSrc ==> " + Arrays.toString(outSrc));
        input.close();
        output.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

(3) FileInputStream

```
public static void main(String[] args) {
    // FileInputStream객체를 이용한 파일 내용 읽기
    FileInputStream fin = null;  // 선언

    try {
        // 방법1 (파일정보를 문자열로 지정하기)
        fin = new FileInputStream("d:/D_Other/test2.txt");  // 생성

        // 방법2 (파일정보를 File객체를 이용하여 지정하기)
        //File file = new File("d:/D_Other/test.txt");
        //fin = new FileInputStream(file);  // 생성

        int c; // 읽어온 데이터를 저장할 변수

        // 읽어온 값이 -1이면 파일의 끝까지 읽었다는 의미이다.
        while( (c=fin.read())!=-1){
            // 읽어온 자료 출력하기
            System.out.print((char)c);
        }

        fin.close(); // 작업 완료 후 스트림 닫기

    } catch (FileNotFoundException e) {
        System.out.println("지정한 파일이 없습니다.");
    } catch (IOException e) {
        System.out.println("알 수 없는 입출력 오류입니다.");
    }
}
```

(4) FileOutputStream

```
public static void main(String[] args) {
    // 파일에 출력하기
    FileOutputStream fout = null;

    try {
        // 출력용 OutputStream객체 생성
        fout = new FileOutputStream("d:/D_Other/out.txt");
        for(char ch='a'; ch<='z'; ch++){

            fout.write(ch);
        }
        System.out.println("파일에 쓰기 작업 완료...");
        // 쓰기작업 완료 후 스트림 닫기
        fout.close();

        // ======================================

        // 저장된 파일의 내용을 읽어와 화면에 출력하기
        FileInputStream fin = new FileInputStream("d:/D_Other/out.txt");
        int c;
        while( (c=fin.read())!=-1){
            System.out.print((char)c);
        }
        System.out.println();
        System.out.println("출력 끝...");
        fin.close();

    } catch (FileNotFoundException e) {
        System.out.println("지정한 파일이 없습니다.");
    } catch (IOException e) {
        System.out.println("알 수 없는 입출력 오류입니다.");
    }
}
```

5-2. 문자기반 스트림

학습목표	• 스트림에 대해서 설명할 수 있다.

필요 지식 /

1 스트림

1. 문자기반 스트림

- 문자(Char)단위로 스트림을 처리하기 위한 스트림을 말한다.
- Reader는 입력용 문자기반 스트림이다.
- Writer는 출력용 문자 스트림이다.
- Reader나 Writer가 스트림 이름에 붙어 있다면, 문자기반 스트림이다.
- 문자기반 스트림은 문서(Text)파일을 입출력 하기에 적합니다.

2 문자기반 스트림 종류

구분	스트림명	설명
Reader	BufferedReader	Reader 버퍼기능을 제공하는 보조 스트림, 라인단위 읽기가능
	LineNumberReader	Reader스트림에 버퍼기능을 제공하는 보조 스트림, 라인 번호를 유지
	CharArrayReader	문자배열로부터 문자를 읽기 위한 스트림
	InputStreamReader	바이트 기반 스트림을 문자기반 스트림인 Reader로 변환해 주는 보조 스트림
	FileReader	파일로부터 바이트를 읽을 때 문자 단위 스트림으로 처리해 주는 스트림
	FilterReader	필터(기능) 적용을 위한 추상클래스
	PushbackReader	읽어들인 문자를 되돌리는(pushback) 기능을 제공하는 스트림
	PipedReader	파이프(Pipe) 기능을 이용한 스트림 처리를 위한 기능을 제공하는 스트림
	StringReader	문자열로부터 문자를 읽기위한 스트림
Writer	BufferedWriter	Writer스트림에 버퍼기능을 제공하는 보조 스트림
	CharArrayWriter	문자배열에 문자를 쓰기 위한 스트림
	FilterWriter	파일에 문자 단위 스트림으로 쓰는 기능을 제공해 주는 스트림
	OutputStreamWriter	바이트 기반 스트림을 문자기반 스트림인 Writer로 변환해 주는 보조 스트림
	FileWriter	파일에 데이터를 쓸 때 문자 단위 스트림으로 처리해 주는 스트림
	PrintWriter	Writer스트림에 다양한 타입의 데이터 출력 기능 제공하는 보조스트림
	PipedWriter	PipedReader에 출력기능 제공하는 스트림
	StringWriter	문자열 출력을 위한 스트림

< 1-1> 문자기반 스트림 객체 종류

3 문자기반 스트림 예제

(1) FileReader

```
public static void main(String[] args) throws IOException {
    // 문자 기반의 스트림을 이용한 파일 내용 읽기
    FileReader fr = null;

    // 문자 단위의 입력을 담당하는 Reader형 객체 생성
    fr = new FileReader("d:/D_Other/testChar.txt");

    int c;

    while((c=fr.read())!=-1){
        System.out.print((char)c);
    }
    fr.close();
}
```

(2) FileWriter 예제

```
public static void main(String[] args) {
    // 사용자가 입력한 내용을 그대로 파일로 저장하기

    // 콘솔(표준 입출력장치)과 연결된 입력용 문자 스트림 생성
    // InputStreamReader스트림 ==> 바이트 기반 스트림을 문자기반
    //              스트림으로 변환해 주는 보조 스트림이다.
    InputStreamReader isr = new InputStreamReader(System.in);

    FileWriter fw = null;  // 파일 출력용 문자 기반 스트림

    try {
        // 파일 출력용 문자 스트림 객체 생성
        fw = new FileWriter("d:/D_Other/testChar.txt");
        int c;

        System.out.println("아무거나 입력하세요.");

        // 콘솔에서 입력할 때 입력의 끝표시는 Ctrl + Z키를 누르면 된다.
        while( (c=isr.read())!= -1){
            fw.write(c); // 콘솔에서 입력받은 값을 파일에 출력하기
        }

        isr.close();
        fw.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

(3) InputStreamReader

```
public static void main(String[] args) {
    // 파일 인코딩을 이용하여 읽어오기
    // InputStreamReader스트림 ==> 바이트 기반 스트림을 문자기반
    //              스트림으로 변환해 주는 보조 스트림이다.
    FileInputStream fin = null;
    InputStreamReader isr = null;
    try {
        /*
        FileInputStream객체를 생성한 후 이 객체를 매개변수로 받는
        InputStreamReader객체를 생성한다.
        (바이트 입력 스트림에 연결되어 문자 입력 스트림인 Reader로
         변환시키는 보조스트림)
        */
        //fin = new FileInputStream("d:/D_Other/test_utf8.txt");
        fin = new FileInputStream("d:/D_Other/test_ansi.txt");
        //isr = new InputStreamReader(fin);
        //isr = new InputStreamReader(fin, "euc-kr");
        isr = new InputStreamReader(fin, "cp949");
        int c;
        while((c=isr.read())!=-1){
            System.out.print((char)c);
        }
        System.out.println();
        System.out.println("출력 끝...");
        isr.close();  // 보조스트림만 닫아도 된다.
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

(4) OutputStreamWriter

```
public class F10_FileEncodingTest {
    /*
    OutputStreamWriter객체 ==> OutputStream(바이트기반의 출력용 객체)를
                    Writer(문자기반의 출력용 객체)로 변환하는 객체
            ==> 이 객체도 출력할 때 '인코딩 방식'을 지정해서 출력할 수 있다.
*/
    public static void main(String[] args) throws Exception {
        // 키보드로 입력한 내용을 파일로 저장하는데
        // out_utf8.txt파일은 'utf-8'인코딩 방식으로
        // out_ansi.txt파일은 'ms949'인코딩 방식으로 저장한다.

        // InputStreamReader 바이트 입력 스트림에 연결되어
        // 문자 입력 스트림인 Reader로 변환시키는 보조스트림
        InputStreamReader isr = new InputStreamReader(System.in);

        // 파일 출력용
        FileOutputStream fos1 =
                  new FileOutputStream("d:/D_Other/out_utf8.txt");
        FileOutputStream fos2 =
                  new FileOutputStream("d:/D_Other/out_ansi.txt");

        // OutputStreamWriter 은 바이트 출력 스트림에 연결되어
        //문자 출력 스트림인 Writer로 변환시키는 보조 스트림.
        OutputStreamWriter osw1 = new OutputStreamWriter(fos1, "utf-8");
        OutputStreamWriter osw2 = new OutputStreamWriter(fos2, "euc-kr");

        int c;

        System.out.println("아무거나 입력하세요");

        while((c=isr.read())!=-1){
            osw1.write(c);
            osw2.write(c);
        }

        System.out.println("작업 완료...");
        isr.close();
        osw1.close();
        osw2.close();
    }
}
```

5-3. 표준 입출력과 File

학습목표	• 입출력 및 파일에 대해서 설명 할 수 있다.

필요 지식 /

1 입출력

JAVA에서는 java.lang.System 클래스를 통해 표준 입출력 및 에러를 제공한다.

1. 표준 입력(System.in)

표준 입력은 프로그램으로 들어가는 데이터(보통은 문자열) 스트림이다. 프로그램은 read 명령을 이용하여 데이터 전송을 요청한다. 모든 프로그램이 입력을 요구하는 것은 아니다. 이를테면 dir이나 ls 프로그램(디렉터리에 들어 있는 파일 이름을 보여 주는 명령)은 실행시 옵션과 같은 명령줄 매개변수를 받을 수는 있으나, 동작 중에 데이터 스트림의 입력 없이 명령을 수행한다.
별도의 리다이렉션 없이 프로그램을 시작한 경우, 표준 입력 스트림은 키보드에서 받아온다.
Java에서는 System.in 객체를 통해서 키보드로부터 입력을 받아 온다.
참고로, 키보드로 입력 받을 때 Ctrl-Z를(EOF을 의미함) 눌러서 콘솔 입력을 종료시킬 수 있다.

2. 표준 출력(System.out)

표준 출력은 프로그램이 출력데이터를 기록하는 스트림이다. 이 스트림을 통해서 출력을 하면 보통은 모니터를 통해 내용을 확인할 수 있다.
자바에서는 PrintStream 클래스 타입인 System.out 객체를 통해 표준 출력을 처리한다.
자바에서 우리가 자주 사용하는 System.out.println() 메서드가 사용하는 객체가 바로 이 표준 출력용 객체다.

3. 표준 오류(System.err)

표준 오류는 프로그램이 오류 메시지나 진단을 출력하기 위해서 사용되는 또 다른 출력 스트림 이다.
표준 출력과는 독립적인 스트림이며 별도의 리다이렉트 될 수 있다.
자바에서는 PrintStream 클래스 타입인 System.err 객체를 통해 에러를 출력한다.
표준출력과 마찬가지로 모니터를 통해서 에러내용을 확인해 볼 수 있다.

2 File

1. File 클래스에 대하여

- 시스템에 있는 파일이나 디렉토리를 추상화한 클래스이다.

- 파일의 생성, 삭제, 크기, 읽기 또는 쓰기 모드 등과 같은 파일 자체를 관리하기 위한 클래스이다.

2. File 클래스 주요 메서드

반환형	메서드명	
boolean	canRead()	읽을 수 있으면 true, 그렇지 않으면 false를 반환한다.
	canWrite()	파일을 쓸 수 있으면 true, 그렇지 않으면 false를 반환한다.
	createNewFile()	파일을 새로 생성하면 true, 그렇지 않으면 false를 반환한다.
	delete()	파일을 지우고 성공하면 true, 그렇지 않으면 false를 반환한다.
	exists()	파일이나 디렉토리가 존재하면 true, 그렇지 않으면 false를 반환한다.
String	getAbsolutePath()	파일의 절대경로를 반환한다.
	getCanonicalPath()	파일의 정규경로를 반환한다.
	getName()	파일명을 반환한다.
boolean	isDirectory()	디렉토리이면 true, 그렇지 않으면 false를 반환한다.
	isFile()	파일이면 true, 그렇지 않으면 false를 반환한다.
long	lastModified()	파일을 지우고 성공하면 true, 그렇지 않으면 false를 반환한다.
	length()	파일이나 디렉토리가 존재하면 true, 그렇지 않으면 false를 반환한다.
String[]	list()	파일을 지우고 성공하면 true, 그렇지 않으면 false를 반환한다.
boolean	mkdir()	파일이나 디렉토리가 존재하면 true, 그렇지 않으면 false를 반환한다.
	renameTo(File dest)	dest 파일 객체로 이름을 바꾸면 true, 그렇지 않으면 false를 반환한다.

< 1-1> File 클래스의 주요 메서드

3. File 클래스 관련 예제

(1) file 생성 예제

```
// File객체 만들기 연습

    // 1. new File(String 파일또는경로명)
    //    ==> 디렉토리와 디렉토리 사이 또는 디렉토리와 파일명 사이의
    //        구분 문자는 '\'를 사용하거나 '/'를 사용할 수 있다.

    //File file = new File("d:/D_Other/test.txt");
    File file = new File("d:\\D_Other\\test.txt");
    System.out.println("파일명 : " + file.getName());
    System.out.println("파일 여부 : " + file.isFile());
    System.out.println("디렉토리(폴더) 여부 : " + file.isDirectory());
    System.out.println("----------------------------");

    File file2 = new File("d:\\D_Other");
    //File file2 = new File("d:/D_Other/test.txt");
    System.out.print(file2.getName() + "은 ");
    if(file2.isFile()){
        System.out.println("파일");
    }else if(file2.isDirectory()){
        System.out.println("디렉토리(폴더)");
    }
    System.out.println("----------------------------");

    // 2. new File(File parent, String child)
    //  ==> 'parent'디렉토리 안에 있는 'child'파일 또는 디렉토리를 갖는다.
    File file3 = new File(file2, "test.txt");
    System.out.println(file3.getName() + "의 용량 크기 : " +
                               file3.length() + "bytes");

    // 3. new File(String parent, String child)
    //File file4 = new File("d:/D_Other", "test.txt");
    File file4 = new File("\\D_Other\\test\\..", "test.txt");
    System.out.println("절대 경로 : " + file4.getAbsolutePath()); // 절대경로
    System.out.println("경로 : " + file4.getPath()); // 생성자에 설정해준 경로
    System.out.println("표준 경로 : " +  file4.getCanonicalPath());
    System.out.println("--------------------------------");
```

(2) 생성 예제

```
/*
        디렉토리(폴더) 만들기
    1. mkdir()  ==> File객체의 경로 중 마지막 위치의 디렉토리를 만든다.
                ==> 중간의 경로가 모두 미리 만들어져 있어야 한다.

    2. mkdirs() ==> 중간의 경로가 없으면 중간의 경로도 새롭게 만든 후
                    마지막 위치의 디렉토리를 만들어 준다.
    ==> 위 두 메서드 모두 만들기를 성공하면 true, 실패하면 false 반환
    */
    File file5 = new File("d:/D_Other/연습용");
    if(file5.mkdir()){
        System.out.println(file5.getName() + " 만들기 성공!");
    }else{
        System.out.println(file5.getName() + " 만들기 실패!!!");
    }
    System.out.println();
    File file6 = new File("d:/D_Other/test/java/src");
    //if(file6.mkdir()){
    if(file6.mkdirs()){
        System.out.println(file6.getName() + " 만들기 성공!");
    }else{
        System.out.println(file6.getName() + " 만들기 실패!!!");
    }
    System.out.println();
```

3 (Encoding)

1. 한글 인코딩 에 대하여

- 한글 인코딩 방식은 크게 UTF-8 과 EUC-KR 두 가지 방식으로 나뉜다.

- 한글윈도우는 기본적으로 CP949(Code Page 949) 방식을 사용하는데, 윈도우를 개발한 마이크로소프트에서 EUC-KR 방식에서 확장하였기 때문에 MS949 라고도 부른다.

- 한글윈도우의 메모장에서 이야기하는 ANSI 방식의 인코딩이란, CP949(MS949) 를 말한다.

- CP949는 EUC-KR의 확장이며, 하위 호환성이 있다.

2. 한글 인코딩 처리 관련 예제

```
public static void main(String[] args) throws IOException {
    String propery = System.getProperty("file.encoding");
    System.out.println("기본 인코딩 방식: " + propery);
    System.out.println();

    String msg = "홍길동"; // 인코딩 대상 문자열

    byte[] msgByte1 = msg.getBytes();
    byte[] msgByte2 = msg.getBytes("MS949");
    byte[] msgByte3 = msg.getBytes("EUC-KR");
    byte[] msgByte4 = msg.getBytes("UTF-8");

    System.out.println("기본인코딩:\t" + Arrays.toString(msgByte1));
    System.out.println("MS949:\t" + Arrays.toString(msgByte2));
    System.out.println("EUC-KR:\t" + Arrays.toString(msgByte3));
    System.out.println("UTF-8:\t" + Arrays.toString(msgByte4));

    System.out.println(msg + " => 문자열 길이: " + msg.length());
    System.out.println(msg + " => 문자열 길이(기본인코딩): " + msgByte1.length);
    System.out.println(msg + " => 문자열 길이(MS949): " + msgByte2.length);
    System.out.println(msg + " => 문자열 길이(EUC-KR): " + msgByte3.length);
    System.out.println(msg + " => 문자열 길이(UTF-8): " + msgByte4.length);

    System.out.println("기본인코딩:\t"  + new String(msgByte1));
    System.out.println("MS949:\t "  + new String(msgByte2, "MS949"));
    System.out.println("EUC-KR:\t "  + new String(msgByte3, "EUC-KR"));
    System.out.println("UTF-8:\t "  + new String(msgByte4, "UTF-8"));
}
```

5-4. 직렬화

학습목표	•	대해서 설명 할 수 있다.

필요 지식 /

1 (Serialization)

1. 객체의 직렬화

- 객체의 직렬화란 객체를 스트림 형식으로 파일에 저장하는 방법이다.
- 객체를 직렬화 하기 위해서는 해당 객체에 Serializable 인터페이스를 구현해 주어야 한다.

2. Serializable 인터페이스

- Serializable 인터페이스를 구현한 클래스를 작성하면 해당 클래스의 모든 멤버변수가 직렬화 대상이 된다.
- 객체를 스트림을 통해 입출력 하기 위해서는 직렬화 과정이 필요한다. 직렬화 대상은 해당 클래스의 모든 멤버변수가 된다.
- 객체의 멤버변수 중에서 직렬화 대상에서 제외하고 싶다면 transient 키워드를 사용하여 제외시킨다.
- 객체를 직렬화하여 출력가능하도록 만들어주는 스트림 객체가 ObjectOutputStream이다
- 역직렬화를 통해서 객체를 만들어주는 스트림 객체가 ObjectInputStream이다

3. 직렬화를 이용한 객체 입출력 처리 예제

(1) 직렬화 처리 대상 객체

```java
class Member implements Serializable {
    // 자바는 Serializable 인터페이스를 구현한 클래스만 직렬화 할수 있도록 제한하고 있음
    // 구현안하면 직렬화작업시 java.io.NotSerializableException 예외 발생!

    private static final long serialVersionUID = 20231413671217877591;
    // transient ==> 직렬화가 되지 않을 멤버변수에 지정한다.
    //               (* static 필드도 직렬화가 되지 않는다.)
    //          직렬화가 되지 않는 멤버변수는 기본값으로 저장된다.
    //          (참조형변수 : null, 숫자형변수 : 0)
    private transient String name;
    private transient int age;
    private String addr;

    public Member(String name, int age, String addr) {
        super();
        this.name = name;
        this.age = age;
        this.addr = addr;
    }
```

(2) 통한 객체 저장(출력)

```java
public static void main(String[] args) {
        // Member 인스턴스 생성
        Member mem1 = new Member("홍길동", 20, "대전");
        Member mem2 = new Member("일지매", 30, "경기");
        Member mem3 = new Member("이몽룡", 40, "강원");
        Member mem4 = new Member("성춘향", 20, "광주");

        try {
            // 객체를 파일에 저장하기

            // 출력용 스트림 객체 생성
            ObjectOutputStream oos = new ObjectOutputStream(
                        new BufferedOutputStream(
                            new FileOutputStream("d:/D_Other/memObj.bin")
                        )
                    );

            // 쓰기 작업
            oos.writeObject(mem1); // 직렬화
            oos.writeObject(mem2);
            oos.writeObject(mem3);
            oos.writeObject(mem4);

            System.out.println("쓰기 작업 완료");
            oos.close();
```

(3) 통한 객체 생성

```
        // 저장한 객체를 읽어와 출력하기

        // 입력용 스트림 객체 생성
        ObjectInputStream ois = new ObjectInputStream(
                new BufferedInputStream(
                    new FileInputStream("d:/D_Other/memObj.bin")
                )
            );

        Object obj = null;

        try {
            while((obj=ois.readObject())!=null ){
                // 읽어온 데이터를 원래의 객체형으로 변환 후 사용한다.
                Member mem = (Member)obj;
                System.out.println("이름 : " + mem.getName());
                System.out.println("나이 : " + mem.getAge());
                System.out.println("주소 : " + mem.getAddr());
                System.out.println("-------------------------");
            }

            ois.close();

        } catch (ClassNotFoundException e) {

        }

    } catch (IOException e) {
         // 더이상 읽어올 객체가 없으면 예외발생함.
        System.out.println("출력 작업 끝...");
    }
}
```

6-1. 네트워킹(TCP, UDP, RMI)

학습목표	• 대한 기본적 이해를 바탕으로 TCP, UDP, RMI 프로토콜의 특징 및 장단점을 설명할 수 있다.

필요 지식 / 프로그램(Java)언어 기본문법에 대한 이해

1 기초

1. 네트워크

교환을 목적으로 로컬 컴퓨터와 원격 컴퓨터 간에 데이터의 흐름을 나타내는 구조

2. IP(Internet Protocol)

주소라 불리는 유일한 32비트 숫자로 구성된 주소체계.(221.214.3.102)

▸ 포트(PORT)

프로그램에서 사용되는 논리적인 접속 장소이다. 80(HTTP), 21(FTP), 22(SSH), 23(TELNET)등이 있다.

포트번호는 0~65535까지 이며, 0~1023까지는 시스템에 의해 예약된 포트번호이기 때문에 될 수 있는 한 사용하지 않는 것이 바람직하다.

▸ 프로토콜(PROTOCOL)

클라이언트와 서버간의 통신 규약 이다. 상호간의 접속이나 절단방식, 통신방식, 데이터의 형식, 전송속도 등에 대하여 정하는 것.

▸ TCP/IP(Transmission Control Protocol/Internet Protocol)

인터넷상에서 호스트들을 서로 연결시키는데 사용하는 통신 프로토콜로 기종이 서로 다른 컴퓨터 시스템을 서로 연결해 데이터를 전송하기 위한 통신 프로토콜 이다.

▸ UDP(User Datagram Protocol)
IP를 사용하는 네트워크 내에서 컴퓨터들 간에 메시지들을 교환할 때 제한된 서비스만을 제공하는 통신 프로토콜.
▸ URL(Uniform Resource Locator)
인터넷상에 있는 각종 정보들의 위치를 나타내는 표준이다.
▸ URI(Uniform Resource Identifier)
특정 자원에 접근하기 위한 형식이나 고유한 이름으로 URL보다 넓은 의미의 개념이다.
▸ Broadcast
여러 방향으로 동시에 전송하여 동일 IP그룹에 있는 컴퓨터라면 데이터를 수신할 수 있는 방식이다.

▸ Unicast
특정한 대상 수신자에게만 보내는 방식이다.
▸ Multicast
다중의 수신 대상자들에게 보내는 방식이다.
▸ RMI(Remote Method Invocation)
자바 프로그래밍 언어와 개발 환경을 사용하여 서로 다른 컴퓨터 상에 있는 객체들이 분산 네트워크 내에서 상호 작용하는 객체지향형 프로그램을 작성할 수 있도록 해주는 방식이다. RMI는 일반적으로 RPC라고 알려진 것의 자바 버전이다.

[TCP 구현을 위한 클래스들]

▸ InetAddress 클래스
▸ URL, URLConnection 클래스
▸ Socket 클래스
▸ ServerSocket 클래스

[InetAddress 클래스]

- 정의: IP주소를 표현한 클래스

반환형	메서드	설명
InetAddress[]	getAllByName(String host)	매개변수 host에 대응되는 InetAddress 배열을 반환한다.
InetAddress	getByAddress(byte[] addr)	매개변수 addr에 대응되는 InetAddress 객체를 반환한다.
	getByAddress(String host, byte[] addr)	매개변수 host와 addr로 Inetaddress객체를 생성한다.
	getByName(String host)	매개변수 host에 대응되는 InetAddress 객체를 반환한다.
	getLocalHost()	로컬호스트의 InetAddress 객체를 반환한다.

반환형	메서드	설명
byte[]	getAddress()	InetAddress 객체의 실제 IP 주소를 바이트 배열로 리턴한다.
String	getHostAddress()	IP 주소를 문자열로 반환한다.
	getHostName()	호스트 이름을 문자열로 반환한다.
	toString()	IP 주소를 스트링 문자열로 오버라이딩 한 메소드

[URL 통신]

- URL 클래스
  - URL : 인터넷에서 접근 가능한 자원(Resource)의 주소를 표현할 수 있는 형식.
  - URL을 추상화 하여 만든 클래스

<protocol>://<host>:<port>/<path>?<query>#<reference>
http://www.daum.net:80/member/mem.jsp?name=sung#content

- URLConnection 클래스
  - 원격자원에 접근하는데 필요한 정보를 갖고 있다.
  - 원격서버의 헤더 정보, 해당 자원의 길이와 타입정보, 언어 등을 얻어 올 수 있다.
  - 추상화 클래스 이므로 URL 클래스의 openConnection() 메소드를 이용 객체 생성.
  - URLConnection 클래스의 connect()메소드를 호출해야 객체가 완성됨.

```
URL url = new URL("http://java.sun.com");
URLConnection urlCon = url.openConnection();
urlCon.connect();
```

[Socket/ServerSocket클래스]

- Socket 클래스
  - 서버 프로그램으로 연결 요청을 한다.
  - 데이터 전송을 담당한다.

Socket 클래스의 주요 생성자

생성자	설명
Socket(InetAddress address, int port)	InetAddress 객체와 port를 이용하여 Socket 객체를 생성한다.
Socket(String host , int port)	host와 port를 이용하여 Socket 객체를 생성한다.

- Socket 클래스

Socket 클래스의 주요 메서드

반환형	메서드	설명
void	close( )	소켓 객체를 닫는다.
InetAddress	getInetAddress( )	소켓 객체를 InetAddress 객체로 반환한다.
InputStream	getInputStream( )	소켓 객체로부터 입력할 수 있는 InputStream 객체를 반환한다.
InetAddress	getLocalAddress( )	소켓 객체의 로컬 주소를 반환한다.
int	getPort( )	소켓 객체의 포트를 반환한다.
boolean	isClosed( )	소켓 객체가 닫혀있으면 true를, 열려있으면 false를 반환한다.
	isConnected( )	소켓 객체가 연결되어 있으면 true, 연결되어 있지 않으면 false를 반환한다.

- ServerSocket 클래스
  - 서버 프로그램에서 사용하는 소켓
  - 포트를 통해 연결 요청이 오기를 대기
  - 요청이 오면 클라이언트와 연결을 맺고 또 다른 소켓을 만드는 일을 한다.
  - 새로 만들어진 소켓은 클라이언트 소켓과 데이터를 주고 받는다.

ServerSocket 클래스의 주요 생성자

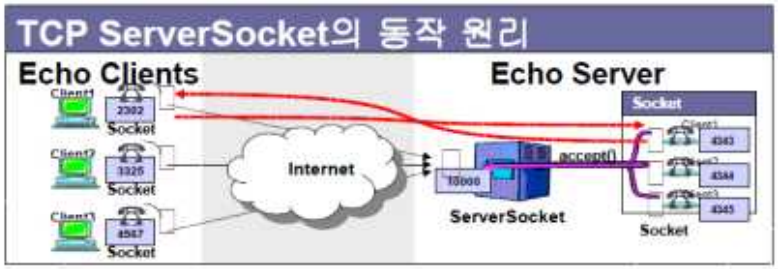
생성자	설명
ServerSocket(int port)	port를 이용하여 ServerSocket 객체를 생성한다.

ServerSocket 클래스

ServerSocket 클래스의 주요 메서드

반환형	메서드	설명
Socket	accept( )	클라이언트의 Socket 객체가 생성될 때까지 블로킹되는 메서드다. 클라이언트의 Socket 객체가 생성되면 서버에서 클라이언트와 통신할 수 있는 Socket 객체를 반환하게 된다.
void	close( )	ServerSocket 객체를 닫는다.
int	getLocalPort( )	ServerSocket 객체가 청취하고 있는 포트번호를 반환한다.
	getSoTimeout( )	ServerSocket 클래스의 accept( ) 메서드가 유효할 수 있는 시간을 밀리 세컨드로 반환한다. 만약, 0이면 무한대를 의미한다.
boolean	isClosed( )	ServerSocket 객체의 닫힌 상태를 반환한다.
void	setSoTimeout(int timeout)	ServerSocket 클래스의 accept( ) 메서드가 유효할 수 있는 시간을 밀리 세컨드로 설정해야 한다. 만약, 시간이 지나면 java.net.SocketTimeoutException 예외가 발생하는데, 이 예외가 발생하더라도 ServerSocket 객체는 계속 유효하다.

[TCP 통신 동작 원리]



- ServerSocket
  - ServerSocket은 클라이언트와 통신할 수 있는 서버용 Socket을 만들어 준다.
- ServerSocket의 생성과 접속 대기
  - ServerSocket ss = new ServerSocket(10000);
  - Socket socket = ss.accept();
- ServerSocket의 accept()에서 리턴된 서버용 Socket
  - 클라이언트 Socket으로 ServerSocket에 연결요청할 때 accept()가 반응한다.
  - ServerSocket이 accept()할 때 리턴된 서버용 Socket은 해당 클라이언트와 통신할 수 있는 유일한 수단이다.
  - accept()할 때 리턴된 서버용 Socket은 자동으로 포트(Port)를 할당받는다.
- 서버용 Socket의 생성 및 스트림 개설
  - ServerSocket ss = new ServerSocket(10000);
  - Socket socket = ss.accept();
  - InputStream is = socket.getInputStream();
  - OutputStream os = socket.getOutputStream();

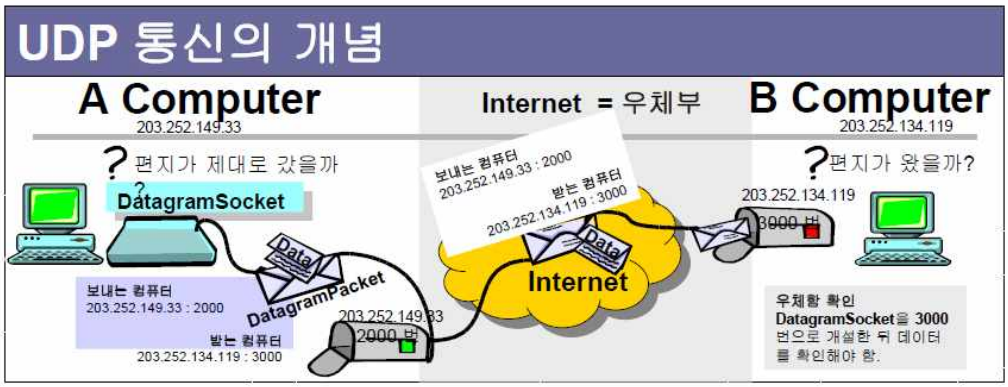
[UDP 통신]

▸UDP(User Datagram Protocol)

▸ .

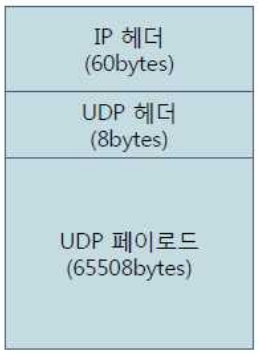
▸데이터의 신뢰성을 보장 하지 않는다.

▸TCP에 비해 전송 속도가 빠르다.



▸UDP Datagram

IP 헤더는 패킷의 발신지와 목적지 주소, 길이, 체크섬, TTL 그리고 다른 IP 옵션들이 포함된다. UDP 헤더는 발신지 포트와 목적지 포트, 헤더를 포함하는 길이, 체크섬으로 구성된다. UDP Payload에는 실제 데이터가 전송되는 공간으로 실제 데이터가 들어가 있다. 그러나 일반적으로 512바이트로 제한하는 경우가 많다.



[UDP 통신 관련 클래스]

▸ DatagramPacket 클래스
▸ UDP 데이터그램을 추상화 한 클래스.
▸ 애플리케이션에서 주고받을 데이터와 관련된 클래스.
▸ 데이터를 송신기능과 수신 기능으로 크게 분리된다.
▸ 출발지 주소와 목적지 주소를 설정하거나 주소를 얻어오는 메소드 제공
▸ 출발지 포트와 목적지 포트를 설정하거나 포트를 얻어오는 메소드 제공

DatagramPacket 클래스의 주요 생성자

생성자	설명
DatagramPacket(byte[ ] buf, int length)	데이터를 수신하기 위한 생성자로 바이트 배열 buf의 length 만큼 저장한다.
DatagramPacket(byte[ ] buf, int length, InetAddress address, int port)	데이터를 송신하기 위한 생성자로 address와 port로 바이트 배열 buf의 length만큼 저장한다.
DatagramPacket(byte[ ] buf, int offset, int length)	데이터를 수신하기 위한 생성자로 바이트 배열 buf의 offset 위치에서 length만큼 저장한다.
DatagramPacket(byte[ ] buf, int offset, intlength, InetAddress address, int port)	데이터를 송신하기 위한 생성자로 address와 port로 바이트 배열 buf의 offset 위치에서 length만큼 저장한다.

DatagramPacket 클래스의 주요 메서드

반환형	메서드	설명
InetAddress	getAddress( )	데이터그램에 대한 목적지 또는 출발지 주소를 반환한다.
byte[ ]	getData( )	버퍼에 들어있는 실제 데이터를 바이트 배열로 반환한다.
int	getLength( )	버퍼에 들어있는 실제 데이터의 길이를 반환한다.
	getOffset( )	버퍼에 들어있는 실제 데이터의 시작 위치를 반환한다
	getPort( )	데이터그램에 대한 목적지 또는 출발지 포트를 반환한다.
void	setAddress (InetAddress iaddr)	데이터그램을 보낸 호스트 주소를 설정한다.
	setData(byte[ ] buf)	버퍼에 들어있는 실제 데이터를 바이트 배열 buffer로 설정한다.
void	setData(byte[ ] buf, int offset, int length)	버퍼에 들어있는 실제 데이터를 바이트 배열 buffer의 offset 위치에서 length만큼 설정한다.
	setLength(int length)	버퍼에 들어있는 실제 데이터의 길이를 설정한다.
	setPort(int port)	데이터그램에 대한 목적지 또는 출발지 포트를 설정한다.

▸ DatagramSocket 클래스

▸TCP 스트림 소켓과 달리 서버와 클라이언트 데이터 그램 소켓 사이에는 차이가 없으며 모든 데이터그램 소켓은 데이터그램을 전송할 뿐만 아니라 수신에서 사용할 수 있다.

▸모든 DatagramSocket 객체는 데이터그램을 수신하기 위해서 사용될 수 있기 때문에 로컬 호스트 내의 유일한 UDP 포트와 연관되어 있다.

DatagramSocket 클래스의 주요 생성자

생성자	설명
DatagramSocket( )	할당된 특정한 포트번호가 중요하지 않다면 사용 가능한 임시 UDP 포트로 소켓을 생성하여 DatagramSocket 객체를 생성한다.
DatagramSocket(int port)	매개변수 포트로 소켓을 생성하여 DatagramSocket 객체를 생성한다.
DatagramSocket(int port, InetAddress iaddr)	매개변수 port와 iaddr로 소켓을 생성하여 DatagramSocket 객체를 생성한다.

▸DatagramSocket 클래스의 주요 메서드 기능은 DatagramPacket을 보내거나 받을수 있는 메서드를 제공하는 것이다.

DatagramSocket 클래스의 주요 메서드

반환형	메서드	설명
void	send(DatagramPacket dp)	UDP 데이터그램(dp)을 전송하는 메서드다.
	receive(DatagramPacket dp)	UDP 데이터그램을 받아서 이미 존재하는 DatagramPacket 객체인 dp에 저장한다.
	close( )	데이터그램 소켓이 점유하고 있는 포트를 자유롭게 놓아준다.
int	getLocalPort( )	현재 소켓이 데이터그램을 기다리고 있는 로컬 포트가 몇 번인지를 리턴한다.
void	connect(InetAddress address, int port)	DatagramSocket이 지정된 호스트의 지정된 포트하고만 패킷을 주고받을 것이라고 정한다.
	disconnect( )	현재 연결된 DatagramSocket의 연결을 끊는다. 연결이 끊기면 아무것도 하지 못하게 된다.
int	getPort( )	DatagramSocket이 연결되어 있다면 소켓이 연결되어 있는 원격지 포트번호를 반환한다.
InetAddress	getInetAddress( )	DatagramSocket이 연결되어 있다면 소켓이 연결되어 있는 원격지 주소를 반환한다.

[Multicast 통신]

▸UDP 통신은Unicast, Broadcast, Multicast의 구분

▸Multcast를 위한 자바 클래스들

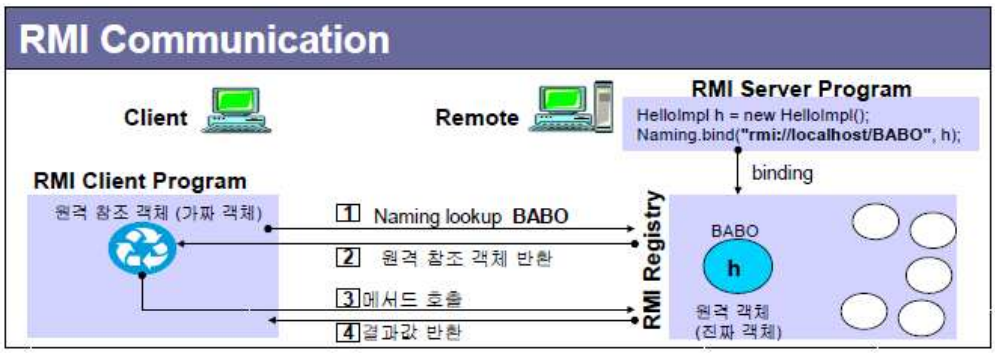
▸DatagramPacket 클래스

▸MulticastSocket 클래스

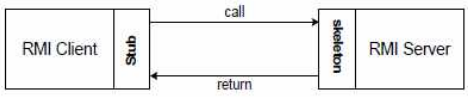
[RMI 통신]

▸ RMI 정의(Remote Method Invocation)

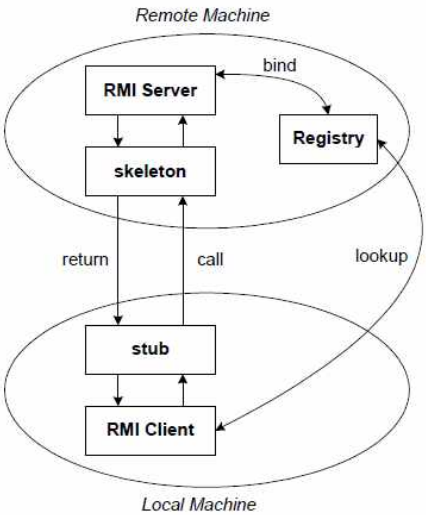
▸ 로컬 컴퓨터에서 원격 컴퓨터의 메서드를 호출하는 기술이다.



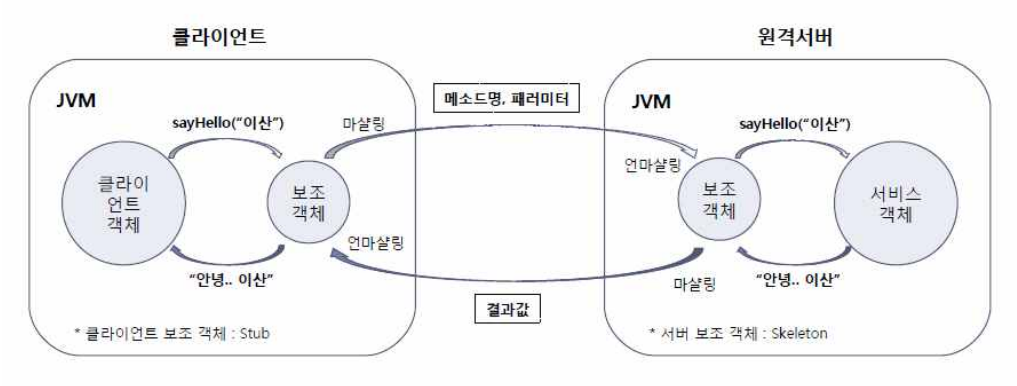
▸ The General RMI Architecture



- The Stub and Skeleton
- Stub
- 실제 통신작업을 처리하는 객체.
- 소켓 및 스트림과 관련된 작업을 처리.
- 클라이언트에서 호출하려고 하는 메소들를 가지고 있는 객체인 척하는 객체.
- Skeleton
- 소켓 연결을 통해 클라이언트 보조 객체(Stub)에서 보낸 요청을 받아서 실제 서비스 객체에 있는 메소드를 호출한다.
- 서비스 객체로 부터 리턴값을 받아서 포장해서 다시 클라이언트 보조 객체(Stub)로 보낸다.



[RMI 작성 방법 및 실행]



▸1 . 원격 인터페이스 작성

서버와 클라이언트가 서로 가지고 있는 메서드를 공유할 목적으로 원격에서 호출 할수 있도록 만든 인터페이스.

인터페이스 작성 규칙

java.rmi.Remote 인터페이스를 상속받아야 한다. 이 인터페이스에 있는 메소드는 로컬이 아닌 원격에서 사용 할 수 있어야 한다는 것을 명시하기 위해. 무조건 public이어야 한다. 원격 접속이 가능하여야 하기 때문. 메소드는 무조건 RemoteException 처리를 해야 한다. 원격처리를 하다가 인터넷상에서 예외가 발생 할 수 있기 때문.

▸2단계 : 서버측 구현 클래스 작성

인터페이스의 구현 클래스 이며 원격으로 객체를 전송하기 위해 UnicastRemoteObject와 같은 직렬화 가능 클래스를 상속받아야 한다.

구현 클래스 작성 규칙

무조건 public이어야 한다. 인터페이스를 구현해야 한다. UnicastRemoteObject 계열의 클래스를 상속받아야 한다.

□ 해당 통신의 하부 구조를 일부 구현해 주고 이 클래스가 객체의 직렬화를 만들어 주기 때문.

▸RemoteException 처리를 위해 디폴트 생성자를 정의 해야 한다.

□ 상속받은 UnicastRemoteObject 생성자에서 해당 예외가 발생시키기 때문.

▸3단계 : Stub 클래스 생성

구현 클래스를 rmi 컴파일(rmic)을 통해 스텁클래스를 생성. 스텁 클래스 는 내부 통신의 통로로 사용되는 클래스이다.

▸4단계 : 서버측 바인딩 클래스 작성

실행을 위한 서버측 실행 클래스 작성

▸5단계 : 클라이언트 lookup 클래스 작성

원격 객체의 메소드를 사용할 클라이언트 클래스 작성

▸6단계 : 실행하기

1. rmiregistry.exe(자바설치폴더bin) 파일실행을 통해 RMI 서버 가동.

2. 서버측 바인딩 클래스 실행(RMI서버에 객체 바인딩)

3. lookup 클래스 실행(Client 실행)

[RMI]

▸RMI를 이용한 채팅

▸원격 메서드를 이용한 채팅 구현

▸폴링(Polling) 기법

▸클라이언트에서 서버의 메서드를 호출하는 방식

▸지금까지 테스트한 HelloRMI와 DayTimeRMI에서 사용한 기법을 말한다.

▸콜백(Callback) 기법

▸폴링과 반대되는 개념으로 클라이언트의 원격 메서드를 서버에서 호출하는 방식

[RMI 프로그램-채팅]

```
import java.rmi.*;

public interface IChatClient extends Remote{
    // 메시지 출력 메소드
    public void printMessage(String msg) throws RemoteException;
}
```

```
import java.rmi.*;

public interface IChatServer extends Remote{

    // 클라이언트 등록 하는 메소드
    public void addClient(IChatClient client, String name) throws RemoteException;

    // 메시지 전송 메소드
    public void sendMessage(String msg) throws RemoteException;

    // 클라이언트 접속 종료 메소드
    public void disconnect(IChatClient client, String name) throws RemoteException;

}
```

6-2. 소켓 프로그래밍

학습목표	• 이용하여 네트워크 프로그램을 만들 수 있다.

필요 지식 / 프로그램(Java)언어 기본문법에 대한 이해

1 (Socket) 통신 기초

1. 소켓(Socket)이란?

네트워크로 연결된 두 대의 호스트간 통신을 위한 양쪽 끝을 의미한다.

2. 소켓(Socket)의 역할

커넥션(Connection)을 개설하기 위한 도구

ex) , 무전기

3. 대표적인 잘 알려진 포트들(Well Known Port)

(1) 7 : Echo

(2) 13 : DayTime

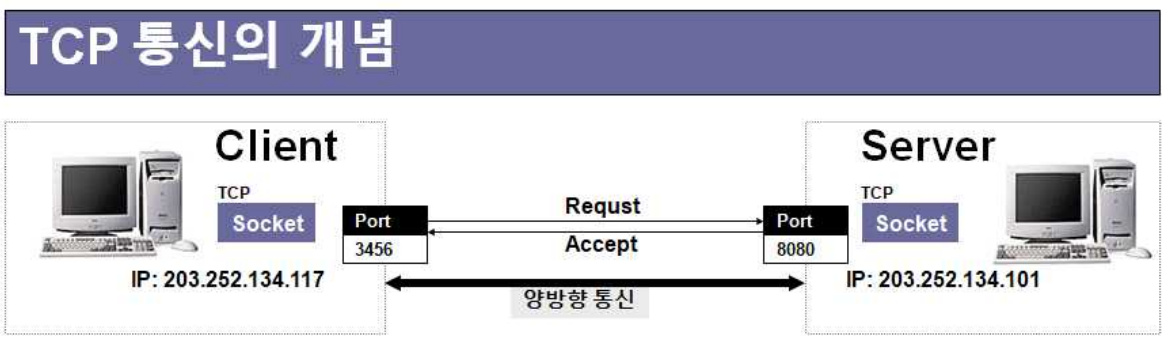
(3) 21 : Ftp

(4) 23 : Telnet

(5) 25 : SMTP

(6) 80 : HTTP

2 TCP 소켓(Socket) 통신



1. TCP 소켓 서버 프로그래밍

(1) 과정

(가) TCP 소켓통신을 하기 위해 ServerSocket 객체를 생성한다.

(나) ServerSocket객체의 accept()메서드를 호출하여 Client로부터 연결요청이 올 때까지 계속 기다린다.

(다) 연결 요청이 들어오면 새로운 Socket객체를 생성하여 Client의 Socket과 연결한다.

(라) Socket객체의 Stream객체(InputStream, OutputStream)를 이용하여 메시지를 주고 받는다.

(마) 사용이 완료된 소켓은 close()메서드를 이용하여 종료 처리한다.

(2) 예제소스

```
public class TcpSocketServer {

    public static void main(String[] args) throws IOException {
        // TCP 소켓 통신을 하기 위해 ServerSocket객체 생성
        ServerSocket server = new ServerSocket(7777);
        System.out.println("서버가 접속을 기다립니다...");

        // accept()메서드는 Client에서 연결 요청이 올 때까지 계속 기다린다.
        // 연결 요청이 오면 Socket객체를 생성해서 Client의 Socket과 연결한다.
        Socket socket = server.accept();
        //---------------------------------
        // 이 이후는 클라이언트와 연결된 후의 작업을 진행하면 된다.

        System.out.println("접속한 클라이언트 정보");
        System.out.println("주소 : " + socket.getInetAddress());

        // Client에 메시지 보내기
        // OutputStream객체를 구성하여 전송한다.
        // 접속한 Socket의 getOutputStrea()메서드를 이용하여 구한다.
        OutputStream out = socket.getOutputStream();
        DataOutputStream dos = new DataOutputStream(out);
        dos.writeUTF("어서오세요...");  // 메시지 보내기
        System.out.println("메시지를 보냈습니다.");

        dos.close();

        server.close();
    }
}
```

2. TCP 소켓 클라이언트 프로그래밍

(1) 과정

(가) 소켓을 생성하여 서버에 연결 요청을 한다.

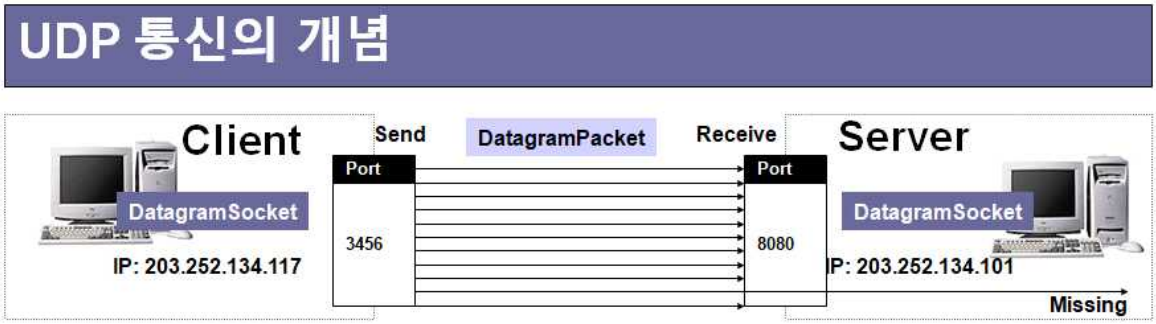
(나) 연결이 정상적으로 이루어 지면 생성된 소겟객체를 이용하여 서버와 메시지를 주고 받는다.
(생성된 소켓으로부터 스트림(InputStream, OutputStream)객체를 가져와 이용함.)

(다) 사용이 완료된 소켓은 close()메서드를 이용하여 종료 처리한다.

(2) 예제소스

```
public class TcpSocketClient {

    public static void main(String[] args) throws UnknownHostException,
                                                   IOException {
        String serverIp = "127.0.0.1";
        // 자기 자신 컴퓨터를 나타내는 방법
        // IP : 127.0.0.1
        // 컴이름 : localhost

        System.out.println(serverIp + " 서버에 접속 중입니다.");

        // 소켓을 생성해서 서버에 연결을 요청한다.
        Socket socket = new Socket(serverIp, 7777);

        // 연결이 되면 이후의 명령이 실행된다.
        System.out.println("연결되었습니다..");

        // 서버에서 보내온 메시지 받기
        // 메시지를 받기 위해 InputStream객체를 생성한다.
        // Socket의 getInputStream()메서드 이용
        InputStream is = socket.getInputStream();
        DataInputStream dis = new DataInputStream(is);

        // 서버로부터 받은 메시지 출력하기
        System.out.println("받은 메시지 : " + dis.readUTF());

        System.out.println("연결 종료.");

        dis.close();

        socket.close();
    }
}
```

3 UDP (Socket) 통신



1. UDP 소켓 서버 프로그래밍

(1) 과정

(가) TCP 소켓통신을 하기 위해 ServerSocket 객체를 생성한다.

(나) ServerSocket객체의 accept()메서드를 호출하여 Client로부터 연결요청이 올 때까지 계속 기

(2) UDP 소켓 서버 역할 프로그래밍 예제

```java
public class UdpServer {
    private DatagramSocket socket;

    public void start() throws IOException {

        socket = new DatagramSocket(8888);  // 포트 8888번을 사용하는 소켓을 생성한다.
        DatagramPacket inPacket, outPacket; // 패킷 송수신을 위한 객체변수 선언

        byte[] inMsg = new byte[10];// 패킷수신을 위한 바이트배열 선언
        byte[] outMsg;              // 패킷송신을 위한 바이트배열 선언

        while(true) {
            // 데이터를 수신하기 위한 패킷을 생성한다.
            inPacket = new DatagramPacket(inMsg, inMsg.length);

            // 패킷을 통해 데이터를 수신(receive)한다.
            socket.receive(inPacket);

            // 수신한 패킷으로 부터 client의 IP주소와 Port를 얻는다.
            InetAddress address = inPacket.getAddress();
            int port = inPacket.getPort();

            // 서버의 현재 시간을 시분초 형태([hh:mm:ss])로 반환한다.
            SimpleDateFormat sdf = new SimpleDateFormat("[hh:mm:ss]");
            String time = sdf.format(new Date());
            outMsg = time.getBytes(); // 시간문자열을 byte배열로 변환한다.

            // 패킷을 생성해서 client에게 전송(send)한다.
            outPacket = new DatagramPacket(outMsg, outMsg.length, address, port);
            socket.send(outPacket); // 전송시작
        }
    } // start()

    public static void main(String args[]) {
        try {
            new UdpServer().start(); // UDP서버를 실행시킨다.
        } catch (IOException e) {
            e.printStackTrace();
        }
    } // main
}
```

2. UDP 소켓 클라이언트 프로그래밍

(1) 과정

(가) 소켓을 생성하여 서버에 연결 요청을 한다.

( ) 연결이 정상적으로 이루어 지면 생성된 소겟객체를 이용하여 서버와 메시지를 주고 받는다.
(생성된 소켓으로부터 스트림(InputStream, OutputStream)객체를 가져와 이용함.)

(다) 사용이 완료된 소켓은 close()메서드를 이용하여 종료 처리한다.

(2) 예제소스

```
public class UdpClient {
    public void start() throws IOException, UnknownHostException {
        DatagramSocket datagramSocket = new DatagramSocket(); // 소캣객체 생성
        InetAddress    serverAddress  = InetAddress.getByName("127.0.0.1");

        // 데이터가 저장될 공간으로 byte배열을 생성한다.(패킷 수신용)
        byte[] msg = new byte[100];

        DatagramPacket outPacket = new DatagramPacket(msg, 1, serverAddress, 8888);
        DatagramPacket inPacket  = new DatagramPacket(msg, msg.length);

        datagramSocket.send(outPacket);   // DatagramPacket을 전송한다.
        datagramSocket.receive(inPacket); // DatagramPacket을 수신한다.

        System.out.println("현재 서버 시간 => " + new String(inPacket.getData()));

        datagramSocket.close(); // 소켓 종료
    } // start()

    public static void main(String args[]) {
        try {
            new UdpClient().start();
        } catch(Exception e) {
            e.printStackTrace();
        }
    } // main
}
```

4 RMI

1. RMI 통신을 위한 서버 프로그래밍

(1) 과정

(가) RMI용 인터페이스를 정의한다.(Remote Interface을 상속 받아야 한다.)

(나) RMI용 인터페이스를 구현한 객체 생성

(다) 구현한 객체를 클라이언트에서 찾을 수 있도록 Registry객체를 생성해서 등록한다.

(라) Registry서버에 서비스를 제공할 원격객체 등록하여 클라이언트에서 가져가 사용할 수 있도록 한다.

(2) 예제 소스

```
// RMI용 인터페이스는 Remote를 상속해야 한다.
public interface RemoteInterface extends Remote {
    // 이 인터페이스에서 선언된 모든 메서드는 RemoteException을 throws해야 한다.

    // 이 곳에서 선언된 메서드의 파라미터 변수는 클라이언트에서
    // 보내오는 데이터가 저장되고, 반환값은 서버에서 클라이언트 쪽으로
    // 보내는 데이터가 된다.

    public int doRemotePrint(String str) throws RemoteException;

    public void doPrintList(ArrayList<String> list) throws RemoteException;

    public void doPrintVo(TestVO vo) throws RemoteException;

    public void setFiles(FileInfoVO[] fInfo) throws RemoteException;

}
```

```java
public class RemoteServer extends UnicastRemoteObject implements RemoteInterface {

    // RMI용 인터페이스를 구현한 객체의 생성자도 RemoteException을
    // throws하도록 작성한다.
    protected RemoteServer() throws RemoteException {
        super();
    }
    public static void main(String[] args) {
        try {
            // 구현한 RMI용 객체를 클라이언트에서 사용할 수 있도록
            // RMI서버에 등록한다.

            // 1. RMI용 인터페이스를 구현한 객체 생성
            RemoteInterface inf = new RemoteServer();

            // 2. 구현한 객체를 클라이언트에서 찾을 수 있도록
            //    Registry객체를 생성해서 등록한다.

            // 포트번호를 지정하여 Registry객체 생성(기본포트값 : 1099)
            Registry reg = LocateRegistry.createRegistry(8888);

            // Registry서버에 제공하는 객체 등록
            // 형식) Registry객체변수.rebind("객체의Alias", 등록할객체변수);
            reg.rebind("server", inf);

            System.out.println("서버가 준비되었습니다.");

        }catch(RemoteException e) {
            e.printStackTrace();
        }
    }

    // RMI용 인터페이스에서 선언한 메서드들을 작성한다.

         .... 이후 생략.....
}
```

2. RMI 통신을 위한 클라이언트 프로그래밍

(1) 과정

(가) RMI용 인터페이스를 클라이언트 쪽에서도 서버와 같은 패키지 구조로 생성해야 한다.

(나) Registry서버에 등록된 원격객체를 가져오기 위해 Registry 객체를 생성한다.

( ) lookup()메서드를 호출하여 등록된 원격객체를 불러와 사용한다.

(2) 예제 소스

```
/*
    클라이언트쪽의 프로젝트에도 서버의 패키지와 같은 구조로
    Interface와 VO파일이 있어야 한다.
*/
public class RemoteClient {

    public static void main(String[] args) {
        // Registry서버에 등록된 객체를 구한다.
        try {
            // 1. 등록된 서버를 찾기 위해 Registry객체를 생성한 후
            //    사용할 객체를 불러온다.
            Registry reg = LocateRegistry.getRegistry("localhost", 8888);
            RemoteInterface clientInf = (RemoteInterface) reg.lookup("server");

            // 이제부터는 불러온 객체의 메서드를 호출해서 사용할 수 있다.
            int a = clientInf.doRemotePrint("안녕하세요");
            System.out.println("반환값 => " + a);
            System.out.println("-------------------------");

             ... 이후 소스는 생략 ...
```

7-1. SQL Mapper / Data Mapper 개요

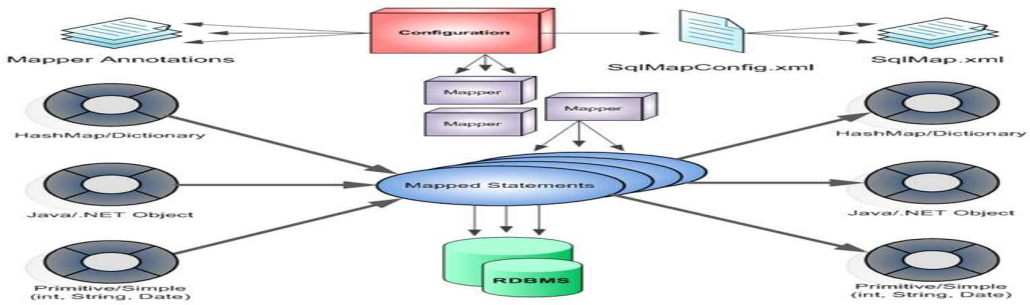
학습목표	• SQL Mapper / Data Mapper 대해서 설명할 수 있다.

필요 지식 /

1 SQL Mapper / Data Mapper 대하여

1. MyBatis란?

일반적으로 어플리케이션들은 다양한 형태의 명령을 처리하기 위한 기반 데이터를 데이터베이스와 같은 영속영역에 관리하기 때문에 이러한 영역과의 통신을 통한 데이터 확보는 필수적인 작업이라 할 수 있는데, 불특정 다수로부터 요청이 발생하는 웹 어플리케이션에 있어서는 그 중요성이 훨씬 더하다 할 것이다. 데이터베이스로부터 데이터를 확보하기 위해 Java 진영에서는 자바 어플리케이션과 데이터베이스시스템간의 연동을 위한 표준 스펙을 제공하고, 각 벤더들이 해당 스펙에 따른 JDBC 드라이버를 제공함으로써 다양한 어플리케이션과 다양한 데이터베이스 간의 공통적인 통신 코드를 산출할 수 있는 구조가 만들어져있다. 그런데 이러한 드라이버를 사용하는 공통 코드들이 대부분 동일한 인터페이스에 따라 같은 형태의 코드가 반복되는 구조이다 보니, 단순한 CURD를 위해서도 비슷한 형태의 코드가 여러차례 반복되는 코드 중복 현상이 발생하게 된다. 이러한 코드의 중복을 해결하기 위해 반복적인 형태를 띄는 기반 설정 작업들이 미리 코드화되어있는 프레임워크의 사용이 증가했는데, Mybatis도 그러한 데이터 매퍼 프레임워크의 한 종류라 할 수 있다.



<그림 1-1> MyBatis Architecture

	설명
(SqlMapConfig.xml)	DataSource, Data Mapper, Transaction Manager 등 데이터베이스 사용을 위한 기본 설정들을 가진 설정 파일이다.
매퍼 인터페이스	SQL을 정의한 XML이나 어노테이션으로 정의된 SQL을 가진 인터페이스로 이를 통해 매퍼 객체가 생성된다.
결과 매핑과 매핑 구문	질의 실행 결과 집합을 자바 객체로 매핑하기 위한 규칙을 정의한 XML이나 어노테이션 설정을 의미한다.
매핑된 구문맵	XML이나 어노테이션으로 등록된 질의문이 관리되는 맵
파라미터	질의 실행을 위해 필요한 파라미터 데이터로 일반적인 자바객체 형태로 표현되면 myBatis에 설정된 타입핸들러에 의해 데이터베이스 데이터타입의 파라미터로 전환된다.
결과 객체	Cursor 형태의 결과집합으로 조회된 결과가 myBatis의 타입핸들러에 의해 자바 객체 형태로 전환된다.

<표 1-1> MyBatis 구성 요소

그림과 같은 구조를 지닌 Mybatis의 역할은 다음과 같이 정리해 볼 수 있다.

- 쿼리를 실행할 수 있도록 연결 설정을 위한 기반 코드의 제공.
- 등록된 쿼리를 유지하기 위한 쿼리 맵의 관리
- 쿼리를 실행하기 위해 어플리케이션에서 넘기는 파라미터의 바인딩
- 쿼리의 실행
- 쿼리를 실행한 결과 객체를 어플리케이션으로 넘기기 위한 결과 객체 매핑

정리하자면, 어플리케이션과 관계형 데이터베이스간에 데이터를 송수신할 수 있는 연결 객체 생성 및 관리와 양쪽에서 서로 다른 형태로 표현되는 데이터를 변환하기 위한 절차가 생략될 수 있도록 고도의 추상화가 제공되는 프레임워크라 할 수 있겠다.

접근방법	JDBC 프로그래밍에 필수적인 자원의 연결/해제 및 공통 에러처리 등을 통합 지원함으로써 쿼리의 실행에만 집중할 수 있는 추상화 제공
코드와 SQL의 분리	소스와 쿼리를 분리 유지함으로서 유지보수 및 튜닝의 용이성을 보장
쿼리 실행 객체 매핑	쿼리의 입력 파라미터에 대한 바인딩과 실행 결과집합의 매핑을 위한 객체 수준의 자동화를 지원
동적 SQL지원	코드작성, API의 직접 사용없이 입력 조건에 따른 동적 쿼리 변경을 지원
다양한 DB 처리 지원	기본 쿼리 외에 배치, 페이징, Callable 함수, binary 데이터 처리등을 지원

<표 1-2> MyBatis 의 제공 기능

Mybatis는 원래 아파치 재단에서 iBatis 라는 이름의 프로젝트로 운영되면서 다른 ORM 프레임워크들에 비해 상대적으로 가볍고 쉬운 장점으로 널리 쓰이다가 2010년 Google Code로 프로젝트가 이전 후 3.0 beta 버전으로 업그레이드되면서 MyBatis로 명칭이 변경되었다. 명칭의 변경뿐 아니라, 데이터 매핑을 위한 기본 설정부터 연결객체 생성 및 자원 관리를 위한 설정 방식 그리고 매퍼 객체의 사용방식에 있어서 상당한 차이를 보이는데, 이 단원을 통해 천천히 학습하도록 하겠다.

iBatis	MyBatis
쿼리 등록 및 관리를 위한 설정에 XML 우선	매퍼 설정에 XML 과 어노테이션 모두 지원
namespace의 의미가 상대적으로 단순	namespace를 통해 매퍼객체가 생성되므로 namespace 선언이 필수사항이며 반드시 매퍼 인터페이스의 qualified name사용
SqlMapClient의 질의 실행 API에 의존	매퍼 인터페이스의 메소드를 직접 사용하므로 type safety 보장
thread safety 한 SqlMapClient 사용	메소드 스코프의 SqlSession 사용
inline parameter : #param#, replaceText : $text$	inline parameter : #{param}, replaceText : ${text}
dynamic 엘리먼트를 이용한 동적 쿼리	if, where 등의 직관적 엘리먼트 사용

<표 1-3> IBatis와 MyBatis의 주요 차이

7-2. iBatis(Mybatis) 의존성 추가

학습목표	• iBatis(Mybatis) 의존성 추가에 필요한 작업을 설명 할 수 있다.

필요 지식 / 프로그램(Java)언어 기본문법에 대한 이해

1 Mybatis 추가

1. MyBatis 시작하기

(1) Mybatis 플러그인 설치 : 이클립스 마켓에서 Mybatis 설치.

(2) Mybatis 라이브러리 의존성 추가 혹은 http://blog.mybatis.org/p/products.html 에서 다운로드

```
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-ehcache</artifactId>
        <version>1.0.0</version>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.8</version>
</dependency>
```

<예제 2-1> ${basedir}/pom.xml

7-3. config file 작성

학습목표	• iBatis(Mybatis) 의존성 추가에 필요한 작업을 설명 할 수 있다.

필요 지식 /

1 MyBatis 설정

1. MyBatis 매퍼 설정(SqlMapConfig.xml)

	의미
properties	설정을 로딩하거나 전역으로 접근할 프로퍼티를 설정하기 위한 요소
settings	런타임에 Mybatis의 행위를 조정하기 위한 설정값들로 SqlSession 관련 설정 최적화를 위한 요소들로 구성되며, 모든 요소는 optional 설정
typeAliases	파라미터나 결과객체의 class qualified name 대신 사용할 수 있는 별칭 집합
typeHandlers	파라미터 바인딩이나 결과 객체 매핑시 타입 변환에 사용될 타입 핸들러 집합, 기본으로 제공되는 타입핸들러 이외의 커스텀 핸들러 등록에 사용
objectFactory	구문 실행 결과를 매핑할 결과객체를 생성하기 위한 factory 클래스 지정 결과 객체를 매핑할 타입에 파라미터가 있는 생성자를 사용하는 경우 설정
plugins	매핑 구문 실행의 특정 시점에 호출을 가로채기 위한 플러그인으로 Mybatis 기반 클래스나 메소드를 수정할 때 사용
environments	transactionManager 나 dataSource 등에 대한 설정을 가진 environment 들의 집합
databaseIdprovider	DatabaseMetaData#getDatabaseProductName() 이 돌려주는 db 서버 별칭 등록시 사용
mappers	구문이 작성된 매퍼xml 이나 어노테이션으로 구문이 작성된 매퍼 인터페이스의 등록에 사용

<표 2-4> 매퍼 설정 프로퍼티 종류

상위 표에 정리된 여러 가지 프로퍼티들을 통해 Mybatis 매퍼 환경에 다양한 설정들을 제어할 수 있는데, settings 는 런타임에 Mybatis 의 행위 및 수행 성능에 영향을 미치는 중요한 값들을 설정할 수 있다. settings 주요 설정 몇가지와 그 의미를 정리하면 다음과 같다.

	의미
cacheEnabled	true, 캐시 사용 여부, false : 각 매퍼별로 캐시 설정
lazyLoadingEnabled	기본 false, 성능 개선을 위한 Lazy Loading 사용 여부. 조회시 데이터를 한꺼번에 가져오지 않고, 필요한 시점에 가져오도록 동작하므로 시스템의 부하를 분산시켜 성능의 향상을 꾀할 수 있다. 특히 기본키를 기준으로 데이터를 가져오는 경우 lazy Loading 이 아닌 것처럼 동작해 빠른 처리가 가능하다.
defaultStatementTimeout	데이터베이스 요청을 처리하는 도중 처리 지연이 발생시 자동으로 중지하도록 하기 위한 지연시간 설정. 기본값은 JDBC 드라이버에 설정된 값
mapUnderscoreToCamelCase	기본값 false, 데이터베이스 테이블 컬럼명에 사용되는 언더바(_)를 자바의 카멜표기법으로 고쳐서 표현할지 여부 ex) co_author => coAuthor
localCacheScope	기본 SESSION, 캐시데이터 저장 범위 설정 SESSION : SqlSession을 기준으로 캐시 저장 STATEMENT : 구문별로 캐시 저장,
useGeneratedKeys	기본값 false, 생성키 사용여부, MySQL의 auto_increment, Oracle의 Sequence, SQL Server의 identify등이 생성키로 제공된다.
defaultExecutorType	기본값 SIMPLE, REUSE : 구문객체 재사용, BATCH : REUSE + 수정작업 배치 처리

<표 2-5> settings 설정 종류

이외에도 다양한 설정들로 Mybatis 의 런타임 수행 성능을 제어할 수 있는데, 자세한 사항은 Mybatis Settings 문서를 참고하도록 한다.

매퍼 설정 파일에서 가장 기본적으로 사용되는 environment 요소는 데이터베이스 접속 환경과 트랜잭션 관리 설정에 대한 정보를 가진 설정 요소로 다음과 같은 특징들을 갖는다.

```
<environments default="development">
  <environment id="development">
    <transactionManager type="JDBC">
      <property name="..." value="..."/>
    </transactionManager>
    <dataSource type="POOLED">
      <property name="driver" value="${driver}"/>
      <property name="url" value="${url}"/>
      <property name="username" value="${username}"/>
      <property name="password" value="${password}"/>
    </dataSource>
  </environment>
</environments>
```

(1) environments default 속성

SqlSessionFactory 객체를 생성하는 과정에서 등록된 여러개의 environment 들 중 특정 하나

가 지정되지 않았다면 기본적으로 사용될 environment의 ID.

```
sessionFactory = new SqlSessionFactoryBuilder().build(reader,
"development");
```

(2) environment 하위요소 transactionManager의 타입

(가) JDBC : JDBC 커밋과 롤백을 직접 처리하기 위해 사용되며, 트랜잭션의 스코프를 관리하기 위해 dataSource로부터 커넥션을 가져온다.

(나) MANAGED : 커밋이나 롤백을 직접 처리하지 않고, 컨테이너가 트랜잭션의 모든 생명주기를 관리하는 경우 사용된다. 참고로 스프링의 경우는 어떠한 트랜잭션관리자를 설정하더라도 스프링 모듈 자체를 구성하기 때문에 설정이 무시된다.

(3) environment의 하위요소 dataSource의 타입

(가) UNPOOLED : 매번 요청이 발생하는 순간 커넥션을 열고 닫는 Simple DataSource로 ibatis 의 "SIMPLE" 과 유사한 타입이며, 다음과 같은 프로퍼티를 갖는다.

	의미
driver	JDBC 패키지 경로를 포함한 자바 클래스명
url	데이터베이스 인스턴스에 대한 JDBC URL
username/password	데이터베이스 접속 계정
defaultTransactionIsolationLevel	커넥션에 대한 기본 트랜잭션 격리 레벨
"driver." prefix	선택적인 프로퍼티 설정. ex) driver.encoding=UTF8

<표 2-6> UNPOOLED 타입 프로퍼티 종류

(나) POOLED : DataSource에 풀링이 적용된 JDBC 커넥션을 위한 구현체(PooledDataSource)로, 새로운 Connection 인스턴스를 생성하기 위해 매번 초기화하는 것을 피할 수 있기 때문에 빠른 응답속도를 요하는 웹 어플리케이션에서 가장 흔히 사용된다. POOLED 타입은 UNPOOLED 에 비해 몇가지 프로퍼티들을 더 설정할 수 있다.

프로퍼티명	의미
poolMaximumActiveConnections	동시에 풀링되어 사용(활성화)될수 있는 최대 커넥션 수
poolMaximumIdleConnections	최대 유휴 커넥션 수
poolTimeToWait	풀이 로그를 출력하고 비정상적으로 긴 경우 커넥션을 다시 얻기 위해 시도하기까지의 대기 시간, 기본 20000ms
poolPingQuery	현재 커넥션의 상태 점검을 위한 테스트 쿼리(핑쿼리), 기본 무설정
poolPingEnabled	핑쿼리 사용 여부 설정, true 일 때 poolPingQuery 설정에 쿼리 등록
poolPingConnectionsNotUsedFor	핑쿼리 실행 간격 설정, 지나치게 자주 핑하는 경우를 피하기 위해 타임아웃 값과 같게 설정하는게 일반적이나, 기본값은 0으로 모든 커넥션에 대해 핑쿼리를 테스트함.

<표 2-7> POOLED 타입 프로퍼티 종류

( ) JNDI : 컨테이너가 JNDI 컨텍스트에 등록한 데이터베이스 풀링 객체를 lookup을 통해 사용하기 위한 타입으로 풀링 데이터소스 객체의 JNDI 등록에 관한 자세한 설명은 톰캣7 JNDI 문서를 참고하기 바란다.

	의미
initial_context	InitialContext에서 컨텍스트를 찾기위해 사용 ex) initialContext.lookup(property_value);
data_source	DataSource 인스턴스의 참조를 찾을 수 있는 컨텍스트 경로

<표 2-8> JNDI 타입 프로퍼티 종류

environment 가 데이터베이스 연결과 트랜잭션 관리를 위한 로우할 레벨의 설정이라면, 이러한 환경 객체를 통해 확보한 커넥션을 통해 실행될 구문에 관한 설정이 필요한데, 이때 사용되는 것이 mappers 요소다. mappers 는 구문에 관한 정보를 가진 xml 이나 자바소스를 등록할 수 있는 mapper 요소의 집합으로, 다음과 같은 형식으로 등록할 수 있다.

```
<!-- Using classpath relative resources -->
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

mapper를 루트 엘리먼트로 하는 매퍼 XML은 insert, select, update, delete 등의 태그를 통해 CRUD를 위한 기본 구문을 등록할 수 있고, 이 경우 매퍼를 등록할 때 상위 그림과 같은 방식을 사용한다.

```
<!-- Using mapper interface classes -->
<mappers>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <mapper class="org.mybatis.builder.BlogMapper"/>
  <mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```

매퍼 인터페이스를 작성하고 자바 코드로 구문을 등록하는 경우는 @Insert, @Select, @Update, @Delete 등의 어노테이션을 통해 구문을 작성한 다음, 상위 그림과 같이 매퍼를 등록한다.

```
<typeAliases>
  <package name="domain.blog"/>
</typeAliases>
```

```
@Alias("author")
public class Author {
    ...
}
```

```
<!-- Register all interfaces in a package as mappers -->
<mappers>
  <package name="org.mybatis.builder"/>
</mappers>
```

또한, CoC 에 충실한 프레임워크들의 경향에 따라 Mybatis도 다양한 방식으로 CoC를 지원하는데, 이를 통해 설정을 간소화할 수 도 있다. 예를 들어, 특정 패키지의 빈들에 대해 축약형 별칭을 자동등록하고 싶거나 특정 패키지의 인터페이스를 매퍼로 자동등록할 때는 상위 그림과 같은 형태의 설정을 통해 가능하다. 타입별칭이 등록될 때는 CoC에 따라 타입명의 첫문자가 소문자로 바뀌어 등록되는데, 이를 명시적으로 바꾸고 싶다면, @Alias 어노테이션을 사용하면 가능하다. Mybatis는 이외에도 다양한 프로퍼티들을 사용하여 설정 외부화나 타입별칭 지정 혹은 타입변환이나 런타임시 수행성능 향상을 위한 캐시 설정 등 다양한 환경들을 제어할 수 있으니 자세한 사항은 Mybatis configuration 문서를 통해 확인하기 바란다.

7-4. 어노테이션 기반, XML 기반 mapper 정의

학습목표	•	및 xml기반 mapper정의 방법을 설명 할 수 있다.

필요 지식 /

1 기반, XML 기반 Mapper 정의

1. 어노테이션 기반, XML 기반 Mapper 작성하기

(1) 결과 매핑 객체(DataBasePropertiesVO) 작성

```
@Alias("dataBasePropertiesVO")
public class DataBasePropertiesVO {
        private String property_name;
        private String property_value;
        private String description;
        public String getProperty_name() {
                return property_name;
        }
        // getter, setter..
}
```

<예제 2-2> kr.or.ddit.chap02.DataBasePropertiesVO

(2) 매퍼 인터페이스 작성 및 매핑 구문 작성

(가) XML Config

```
public interface ICommonDAO {
        public List<DataBasePropertiesVO> retrieveProps();
}
```

<예제 2-3-1> kr.or.ddit.chap02.ICommonDAO

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
                        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="kr.or.ddit.chap02.ICommonDAO">
        <select id="retrieveProps" resultType="dbPropertiesVO">
                SELECT PROPERTY_NAME, PROPERTY_VALUE,
DESCRIPTION
                FROM DATABASE_PROPERTIES
        </select>
</mapper>
```

<예제 2-3-2> src/main/resources/kr/or/ddit/mybatis/mappers/commons.xml

(나) Java Config

```
public interface ICommonDAO {
        @Select({
                "SELECT PROPERTY_NAME, PROPERTY_VALUE,
DESCRIPTION",
                "FROM DATABASE_PROPERTIES" })
        public List<DataBasePropertiesVO> retrieveProps(String textParam);
}
```

<예제 2-3-3> kr.or.ddit.chap02.ICommonDAO

Namespace 는 iBatis 까지는 선택사항이었지만, Mybatis 에서는 각 구문 그룹을 구분할 수 있는 식별자로서 namespace가 필수 설정사항이다. 그런데 단순히 그룹 식별자 역할보다 더 중요한 기능은, Mybatis가 매퍼 인터페이스를 바인딩하고, 매퍼 프록시 객체를 생성하기 위한 조건으로 바로 이 namespace가 사용된다는 것이다. 때문에 단지 관행적으로 뿐만아니라 분석의 용이성을 위해 XMLConfig 나 Java Config에서 모두 패키지 경로를 포함한 매퍼 인터페이스의 qualified name을 namespace로 사용하는 것이 일반적이다.

(3) 설정 및 SqlSessionFactory 빌드하기

(가) XML Config : 데이터베이스 접속 정보는 외부 설정화(kr/or/ddit/database_info.properties)

```
        <?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd" >
        <configuration>
                <properties resource="kr/or/ddit/database_info.properties" />
                <typeAliases>
                        <typeAlias
type="kr.or.ddit.chap02.DataBasePropertiesVO"
        alias="dbPropertiesVO"/>
```

```
        </typeAliases>
        <environments default="development">
                <environment id="development">
                        <transactionManager type="JDBC" />
                        <dataSource type="POOLED">
                                <property name="driver"
value="${maindb.driverClassName}"/>
                                <property name="url" value="${maindb.url}"/>
                                <property name="username"
value="${maindb.username}"/>
                                <property name="password"
value="${maindb.password}"/>
                                <property
name="poolMaximumActiveConnections"
                                                value="${maindb.maxActive}"/>
                                <property name="poolTimeToWait"
value="${maindb.maxWait}"/>
                        </dataSource>
                </environment>
        </environments>
        <mappers>
                <mapper resource="kr/or/ddit/mybatis/mappers/commons.xml"/>
        </mappers>
</configuration>
```

<예제 2-4-1> src/main/resources/kr/or/ddit/mybatis/SqlMapConfig.xml

```
public class CustomSqlSessionFactoryBuilderXMLConfig {
        private static SqlSessionFactory sessionFactory;
        static{
                Reader reader;
                try {
                        reader =
Resources.getResourceAsReader("kr/or/ddit/mybatis/SqlMapConfig.xml");
                        sessionFactory = new
SqlSessionFactoryBuilder().build(reader, "development");
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        public static SqlSessionFactory getSessionFactory() {
                return sessionFactory;
        }
}
```

<예제 2-4-2> kr.or.ddit.mybatis.CustomSqlSessionFactoryBuilderXMLConfig

( ) Java 코드로 설정하는 경우

```
public class CustomSqlSessionFactoryBuilderJavaConfig {
        private static SqlSessionFactory sessionFactory;
        static{
                        ResourceBundle databaseInfo =

ResourceBundle.getBundle("kr.or.ddit.database_info");
                        String driver =
databaseInfo.getString("maindb.driverClassName");
                        String url = databaseInfo.getString("maindb.url");
                        String username =
databaseInfo.getString("maindb.username");
                        String password =
databaseInfo.getString("maindb.password");
                        int poolMaximumActiveConnections
                                =
Integer.parseInt(databaseInfo.getString("maindb.maxActive"));
                        int poolTimeToWait
                                =
Integer.parseInt(databaseInfo.getString("maindb.maxWait"));
                        TransactionFactory transactionFactory = new
JdbcTransactionFactory();
                        PooledDataSource dataSource =
                                        new PooledDataSource(driver, url,
username, password);

dataSource.setPoolMaximumActiveConnections(poolMaximumActiveConnections);
                        dataSource.setPoolTimeToWait(poolTimeToWait);
                        Environment environment =
                                        new Environment("development",
transactionFactory, dataSource);
                        Configuration configuration = new
Configuration(environment);
                        configuration.addMapper(ICommonDAO.class);
                        sessionFactory = new
SqlSessionFactoryBuilder().build(configuration);
        }
        public static SqlSessionFactory getSessionFactory() {
                return sessionFactory;
        }
}
```

<예제 2-4-3> kr.or.ddit.mybatis.CustomSqlSessionFactoryBuilderJavaConfig

```
<tbody>
	<%
		SqlSessionFactory factory
			=
CustomSqlSessionFactoryBuilderXMLConfig.getSessionFactory();
		SqlSession sqlSession = factory.openSession();
		try{
			// iBatis 에서 주로 사용하던 방식으로 type 안정성이
보장되지 않고, 오타에 취약함.
			// List<DataBasePropertiesVO> propsList
			//	=
sqlSession.selectList("kr.or.ddit.chap02.ICommonDAO.retrieveProps");
			// Mybatis 에서 주로 사용하는 방식으로, 매퍼 인터페이스를
사용하는 이유.
			ICommonDAO commonDAO =
sqlSession.getMapper(ICommonDAO.class);
			List<DataBasePropertiesVO> propsList =
commonDAO.retrieveProps("ss");
			for(DataBasePropertiesVO prop : propsList){
				out.println("<tr>");

out.println("<td>"+ prop.getProperty_value()+ "</td>");

out.println("<td>"+ prop.getProperty_name()+ "</td>");

out.println("<td>"+ prop.getDescription()+ "</td>");
				out.println("</tr>");
			}
		}finally {
			sqlSession.close();
		}
	%>
</tbody>
```

<예제 2-5> /chap02/mybatisSample.jsp

(4) SqlSession 통한 데이터 조회

Mybatis 는 매핑 구문을 등록하거나 매퍼 설정을 할 때 xml config 와 java config를 모두 지원하기 때문에 상당히 유연하다. 어느쪽이 더 나은 방식이라는 정답은 없으니 프로젝트 가이드라인에 따라 적절한 선택을 하되 매핑 구문은 일관된 방식으로 정의하는 것이 좋다. Mybatis에 구문을 등록하고, Factory를 통해 SqlSession을 사용하는 과정에서 가장 어려운 부분이 Scope와 LifeCycle 부분이다.

- SqlSessionFactoryBuiilder : SqlSessionFactory 객체 생성 이후 유지할 필요가 없기 때문에 메소드 스코프로 사용하고 유지하지 않는다.
- SqlSessionFacotry : 어플리케이션 스코프로 유지하고, 삭제나 재생성이 필요없다. 참고로 데이터베이스 서버를 여럿 사용하는 경우 각 데이터베이스 서버당 하나씩의 SqlSession Factory 객체가 필요하다.
- SqlSession 및 매퍼 인스턴스 : 각 쓰레드별로 자체적인 SqlSession 인스턴스를 가져야하고, SqlSession 인스턴스는 서로 공유되지 않아야하기 때문에 쓰레드에 안전하지도 않다.

따라서 가장 좋은 스코프는 메소드 스코프라 할 수 있고, 절대 정적 필드나 전역필드로 설정하면 안된다. 또는 서블릿의 HttpSession과 같은 스코프에서 관리하는 것도 위험하다. 어떤 종류의 웹 프레임워크를 사용한다면 Http 요청과 유사한 스코프에 두는 것을 고려해야 한다. 다시 말해 요청이 들어올 때 만들고 응답을 리턴하기 전에 SqlSession을 닫는 방법을 고려해야 하고, 언제나 finally 블록에서 close를 해야지만, 정상적으로 커넥션이 풀에 반환될 수 있다.

2. MyBatis 매퍼 XML, 매퍼 인터페이스 작성

(1) XML을 이용한 매핑 구문 작성

Mybatis의 가장 큰 장점은 실질적으로 필요한 기능을 수행하기 위한 구문을 별도로 등록 및 관리하므로, JDBC 코드와 비교해 95% 이상의 코딩라인이 감소하고, 구문관리에 용이성이 보장된다는 점이다. 매퍼 XML을 이용하는 경우 CRUD를 위한 구문은 구문명과 일치하는 엘리먼트를 통해 등록할 수 있다. 매퍼 XML 의 루트 엘리먼트 mapper는 ibatis의 루트 엘리먼트 sqlMap 과 유사하게 namespace를 통해 매핑 구문들의 그룹 식별자를 지정할 수 있다. 차이점은 ibatis 의 namespace가 부가 설정이었던 반면, Mybatis에서 namespace는 필수 설정이고, Java config 와 함께 어노테이션을 사용해서 매핑 구문을 등록하는 경우 어노테이션을 가진 매퍼 인터페이스의 qualified name이 기본 namespace로 등록되기 때문에 xml설정의 mapper 엘리먼트의 namespace 속성의 값도 일반적으로 매퍼 프록시 객체의 모델이 되는 인터페이스의 qualified name을 사용한다.

(가) select

select 엘리먼트를 통해 조회 구문을 작성할 때 가장 기본적인 형식은 다음과 같다.

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
  SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

select 가 가진 다양한 속성들을 통해 조회 구문 파라미터나 결과 객체 매핑에 관한 설정이나 처리 방식에 대한 세부적인 설정을 제어할 수 있으며 주로 사용되는 속성의 종류와 기능은 다음과 같다.

	설명
id	찾기 위해 사용될 수 있는 namespace 내의 유일한 식별자, 일반적으로 해당 구문을 사용할 매퍼 인터페이스의 메소드명과 일치시킨다.
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
resultType	결과 객체 매핑 타입별칭이나 클래스명.
resultMap	외부 resultMap 의 참조명. 조회구문에는 반드시 결과 데이터 매핑을 위한 resultType 이나 resultMap이 필요하다.
flushCache	구문이 호출시마다 캐시를 지울지(flush) 여부. 디폴트는 false 이다.
useCache	구문의 결과가 캐싱 여부. 디폴트는 true 이다.
timeout	데이터베이스 요청이 발생하고 예외가 던져지기 전까지 최대 대기 시간으로, 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 지원되지 않을 수 있다.
fetchSize	지정된 수만큼의 결과를 리턴하도록 하는 드라이버 힌트 형태의 값이다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 지원되지 않을 수 있다.
statementType	구문 객체의 종류 설정.(STATEMENT, PREPARED, CALLABLE), 디폴트는 PREPARED

<표 2-9> select 구문 속성 종류

( ) nsert, update, delete

```
<insert id="insertAuthor">
  insert into Author (id,username,password,email,bio)
  values (#{id},#{username},#{password},#{email},#{bio})
</insert>
```

```
<update id="updateAuthor">
  update Author set
    username = #{username},
    password = #{password},
    email = #{email},
    bio = #{bio}
  where id = #{id}
</update>
```

```
<delete id="deleteAuthor">
  delete from Author where id = #{id}
</delete>
```

지원하는 속성들의 종류를 보면 키를 생성하기 위한 속성들이 지원되는데, Mybatis는 자동생성키 컬럼을 지원하지 않는 다른 데이터베이스를 위한 방법 또한 제공하고 있다.

	설명
id, parameterType, flushCache, timeout, statementType, databaseId	
useGeneratedKeys	((insert, update)에만 적용) 데이터베이스에서 내부적으로 생성한 키(예를 들어, MySQL 또는 SQL Server 와 같은 RDBMS 의 자동 증가 필드)를 받는 JDBC getGeneratedKeys 메소드를 사용하도록 설정하다. 디폴트는 false 이다.
keyProperty	(입력(insert, update)에만 적용) getGeneratedKeys 메소드나 insert 구문의 selectKey 하위 요소에 의해 리턴된 키를 셋팅할 프로퍼티를 지정. 디폴트는 셋팅하지 않는 것이다.
keyColumn	(입력(insert, update)에만 적용) 생성키를 가진 테이블의 칼럼명을 셋팅. 키 칼럼이 테이블이 첫번째 칼럼이 아닌 데이터베이스(PostgreSQL 처럼)에서만 필요하다.

<표 2-10> insert, update, delete 구문 속성 종류

속성명	설명
keyProperty	selectKey 구문의 결과가 셋팅될 대상 프로퍼티.
keyColumn	리턴되는 결과셋의 칼럼명은 프로퍼티에 일치한다. 여러개의 칼럼을 사용한다면 칼럼명의 목록은 콤마를 사용해서 구분한다.
resultType	결과의 타입. MyBatis 는 이 기능을 제거할 수 있지만 추가하는게 문제가 되지는 않을 것이다. MyBatis 는 String 을 포함하여 키로 사용될 수 있는 간단한 타입을 허용한다.
order	BEFORE 또는 AFTER 를 셋팅할 수 있다. BEFORE 로 셋팅하면, 키를 먼저 조회하고 그 값을 keyProperty 에 셋팅한 뒤 insert 구문을 실행한다. AFTER 로 셋팅하면, insert 구문을 실행한 뒤 selectKey 구문을 실행한다. Oracle 과 같은 데이터베이스에서는 insert 구문 내부에서 일관된 호출 형태로 처리한다.
statementType	구문객체의 종류.

<표 2-11> selectKey 구문 속성 종류

```
<insert id="insertAuthor">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    select CAST(RANDOM()*1000000 as INTEGER) a from SYSIBM.SYSDUMMY1
  </selectKey>
  insert into Author
    (id, username, password, email,bio, favourite_section)
  values
    (#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})
</insert>
```

위 구문에서 selectKey 구문이 먼저 실행되고, Author.id 에 그 값이 셋팅된 다음, insert 구문이 실행되기 때문에, 자동키 생성 컬럼을 지원하지 않는 데이터베이스에서도 사용할 수 있다.

( ) sql

이 요소는 다른 구문에서 재사용가능한 SQL구문을 정의할 때 사용된다.

```
<sql id="userColumns"> id,username,password </sql>
```

```
<select id="selectUsers" resultType="map">
  select <include refid="userColumns"/>
  from some_table
  where id = #{id}
</select>
```

( ) 인라인 파라미터와 대체 구문

Mybatis에서 인라인 파라미터는 #{propertyName} 와 같은 형태로 사용하여 파라미터 객체 내의 특정 프로퍼티의 값을 PreparedStatement 의 파라미터로 전달할 수 있고, 대체 구문은 ${columnName} 와 같은 형태로 사용하여 파라미터 객체내의 특정 프로퍼티의 값을 리터럴이 아닌 구문 자체로 사용되도록 할 수 있다.

```
#{property,javaType=int,jdbcType=NUMERIC}
```

대부분은 인라인 파라미터에 프로퍼티명만 기술하는게 일반적이지만, 특정 컬럼에 null 값이 전달되는 경우라면, jdbcType과 javaType 에 대한 기술이 반드시 필요하다 (PreparedStatement.setNull 참고). 이 외에도 인라인 파라미터내에서 숫자의 크기를 판단하기 위한, numericScale이나 프로시져에 사용하는 경우 in/out 파라미터 모드를 설정하기 위한 mode, 프로시져 결과 객체에 매핑하기 위한 resultMap 속성 등 다양한 속성을 통해 인라인 파라미터에 대해 좀 더 세부적인 설정을 추가할 수 있다.

```
#{height,javaType=double,jdbcType=NUMERIC,numericScale=2}
```

```
#{department, mode=OUT, jdbcType=CURSOR, javaType=ResultSet, resultMap=departmentResultMap}
```

BIT	FLOAT	CHAR	TIMESTAMP	OTHER	UNDEFINED
TINYINT	REAL	VARCHAR	BINARY	BLOG	NVARCHAR
SMALLINT	DOUBLE	LONGVARCHAR	VARBINARY	CLOB	NCHAR
INTEGER	NUMERIC	DATE	LONGVARBINARY	BOOLEAN	NCLOB
BIGINT	DECIMAL	TIME	NULL	CURSOR	ARRAY

<표 2-12> enum 으로 지원되는 JDBC 타입

#{ } 문법은 MyBatis 로 하여금 PreparedStatement 프로퍼티를 만들어서 PreparedStatement 파라미터(예를 들면, ?)에 값을 셋팅하도록 할 것이다. 이 방법이 안전하기는 하지만, 좀더 빠른 방법이 선호되어 가끔은 SQL 구문에 변하지 않는 값으로 삽입하길 원하기도 한다. 예를 들면, ORDER BY 와 같은 구문들이다.

```
ORDER BY ${columnName}
```

이러한 형태의 대체 구문은 리터럴로 처리하거나 이스케이프 처리를 하지 않고, 값을 그대로 구문으로 삽입하여 처리하게 되는데, 동적쿼리에 흔히 사용되기도 한다. 그러나, **사용자로부터 받은 값을 이 방법으로 변경하지 않고 구문에 전달하는 건 안전하지 않다. 이건 잠재적으로 SQL 주입 공격(SQL Injection)에 노출될 가능성이 갖고있기 때문에, 언제나 사용자 입력값에 대해서는 검증과 자체적인 이스케이**

프 처리가 필요하다.

( ) Result Maps

resultMap 요소는 MyBatis 에서 가장 중요하고 강력한 요소이다. ResultSet 에서 데이터를 가져올때 작성되는 JDBC 코드를 대부분 줄여주는 역할을 담당한다. 사실 join 매핑과 같은 복잡한 코드는 굉장히 많은 코드가 필요하다. ResultMap 은 간단한 구문에서는 매핑이 필요하지 않고, 좀더 복잡한 구문에서 관계를 서술하기 위해 필요하다. 조회 구문의 경우, 조회된 결과 집합을 가져오기 위해 흔히 HashMap을 많이 사용한다. 이때 컬럼명이 맵의 키가 되어 자동 매핑되지만, 타입 안정성이 보장되지 않기 때문에 좋은 도메인 모델이라 할 수는 없다. 그래서 대부분 도메인 모델로 자바빈이나 POJO를 사용하게 된다. 이때 컬럼명과 자바빈의 프로퍼티명이 같다면 자동 매핑이라는 규칙성이 적용되는데, Mybatis 가 내부적으로 자바빈의 프로퍼티명을 기준으로 ResultMap을 자동 생성하여 매핑 처리를 하는 것이다. 그런데, 만일 컬럼명과 프로퍼티명이 서로 다르다면 column alias를 사용하여 매핑되도록 하거나, 혹은 resultType 대신 resultMap을 사용할 수 있도록 커스텀 resultMap을 작성하는 방법을 사용할 수도 있다.

위의 코드는 아주 간단한 사용예라 할수 있는데, 실제 ResultMap 은 더 강력한 결과 매핑 기능을 가지고 있으며, 이를 위해 지원되는 resultMap 의 속성 및 하위 요소들의 종류는 다음과 같다.

```
<select id="selectUsers" resultType="User">
  select
    user_id             as "id",
    user_name           as "userName",
    hashed_password     as "hashedPassword"
  from some_table
  where id = #{id}
</select>
```

```
<resultMap id="userResultMap" type="User">
  <id property="id" column="user_id" />
  <result property="username" column="username"/>
  <result property="password" column="password"/>
</resultMap>
```

```
<select id="selectUsers" resultMap="userResultMap">
  select user_id, user_name, hashed_password
  from some_table
  where id = #{id}
</select>
```

	설명
id	매핑을 참조하기 위해 사용할 수 있는 식별자
type	최종 결과 객체의 타입으로, 패키지를 포함한 자바 클래스명이나 타입별칭
autoMapping	자동 매핑 처리 여부를 결정

<표 2-13> resultMap 요소의 속성

1) id, result

결과 매핑의 가장 기본적인 형태로, 둘다 모두 하나의 컬럼을 하나의 프로퍼티나 데이터 필드에 매핑하기 위한 설정이다. id는 객체 인스턴스를 비교할 때 사용하는 구분자 프로퍼티로 처리되는 특수한 경우인데, 이는 보통 캐시나 nested map (join mapping)을 사용하는 경우에 성능 향상을 가져올 수 있다.

	설명
property	칼럼에 매핑하기 위한 필드나 프로퍼티. 자바빈 프로퍼티가 해당 이름과 일치한다면, 그 프로퍼티가 사용될 것이다. 반면에 MyBatis 는 해당 이름이 필드를 찾을 것이다. 점 표기를 사용하여 복잡한 프로퍼티 검색을 사용할 수 있다. 예를 들어, "username" 과 같이 간단하게 매핑될 수 있거나 "address.street.number" 처럼 좀더 복잡하게 매핑될수도 있다.
column	데이터베이스의 칼럼명이나 칼럼 라벨. resultSet.getString(columnName) 에 전달되는 문자열이다.
javaType	패키지 경로를 포함한 클래스 전체명이거나 타입 별칭. 자바빈을 사용한다면 MyBatis 는 타입을 찾아낼 수 있다. 반면에 HashMap 으로 매핑한다면, 기대하는 처리를 명확히 하기 위해 javaType 을 명시해야 한다.
jdbcType	지원되는 타입 목록에서 설명하는 JDBC 타입. JDBC 타입은 insert, update 또는 delete 하는 null 입력이 가능한 칼럼에서만 필요하다. JDBC 의 요구사항이지 MyBatis 의 요구사항이 아니다. JDBC 로 직접 코딩을 하다보면, null 이 가능한 값에 이 타입을 지정할 필요가 있을 것이다.
typeHandler	이 프로퍼티를 사용하면, 디폴트 타입 핸들러를 오버라이드 할 수 있다. 이 값은 TypeHandler 구현체의 패키지를 포함한 전체 클래스명이나 타입 별칭이다.

<표 2-14> id, result 요소의 속성

2) constructor

거의 변하지 않는 데이터를 가진 테이블을 변하지 않는 클래스에 매핑하기 위해 setter 대신 생성자 주입방식을 사용하여, 값을 세팅하는 경우가 있는데, 이때 사용되는 요소가 constructor 이다. 생성자 주입은 public 메소드가 없어도 인스턴스화할 때 값을 셋팅하도록 해준다. 생성자 주입을 위해 constructor를 사용하면, Mybatis 는 일치하는 생성자를 호출해야할 것이고, 때문에 constructor 요소를 생성할 때는 인자의 순서에 주의하고, 반드시 데이터 타입을 명시해야 한다. 결과 데이터 중 키 역할을 하는 데이터는 idArg요소를 사용하여 주입할 수 있는데, 이런 설정을 사용하는 것이 전체적인 성능을 향상할 수 있다.

```
public class User {
   //...
   public User(int id, String username) {
     //...
  }
//...
}
```

```
<constructor>
   <idArg column="id" javaType="int"/>
   <arg column="username" javaType="String"/>
</constructor>
```

다음으로 정리할 association과 collection 은 결과 집합을 매핑할 객체와 객체간의 관계 혹은 join 의 대상이 되는 테이블과 테이블간의 관계를 형성하기 위한 내포형 결과에 사용되는 요소들이다.

3) association : has-one 관계에 사용

association 요소는 property, javaType, jdbcType 등의 속성을 통해 outer객체와 inner 객체 사이의 "has-one" 타입의 관계를 다룬다. 예를 들어, Blog가 하나의 Author를 가지므로 1:1관계를 형성하는 BLOG 테이블과 AUTHOR테이블이 있다. 이때 특정 블로그에 대한 정보를 조회하기 위해 해당 블로그 소유자에 대한 정보까지 조회해야

하는 작업이 필요하다면, 해당 구현을 위해 두 개의 테이블을 조인해야 하는 경우가 발생한다. 이러한 관계 테이블들을 조회하는 방법으로 Mybatis 는 두가지 방법을 제공하고 있다.

내포된(Nested) Select: 복잡한 타입을 리턴하는 다른 매핑된 SQL 구문을 실행하는 방법.

```
<resultMap id="blogResult" type="Blog">
  <association property="author" column="author_id" javaType="Author" select="selectAuthor"/>
</resultMap>
<select id="selectBlog" resultMap="blogResult">
  SELECT * FROM BLOG WHERE ID = #{id}
</select>
```

```
<select id="selectAuthor" resultType="Author">
  SELECT * FROM AUTHOR WHERE ID = #{id}
</select>
```

두개의 select 구문이 있다. 하나는 Blog 를 로드하고 다른 하나는 Author 를 로드한다. 그리고 Blog 의 resultMap 은 author 프로퍼티를 로드하기 위해 “selectAuthor” 구문을 사용한다. 다른 프로퍼티들은 칼럼과 프로퍼티명에 일치하는 것들로 자동 매핑될 것이다.

이 방법은 간단한 반면, “N+1 Selects 문제” 로 알려진 문제점을 야기할 수 있다. N+1 Selects 문제는 처리과정의 특이성으로 인해 야기된다.

- 레코드의 목록을 가져오기 위해 하나의 SQL 구문을 실행한다. (“+1”).
- 리턴된 레코드별로, 각각의 상세 데이터를 로드하기 위해 select 구문을 실행한다. (“N”).

이 문제는 수백 또는 수천의 SQL 구문 실행이라는 결과를 야기할 수 있다. 목록을 로드하고 내포된 데이터에 접근하기 위해 즉시 반복적으로 처리한다면, settings에서 lazyLoadingEnabled를 true 로 설정하고 사용하더라도 one 에 해당하는 객체를 대상으로 proxy 객체가 만들어지기 때문에 많은 데이터가 조회되는 경우, 성능이 떨어질 수 밖에 없다. 그래서 제공되는 다른 방법이 있다.

내포된(Nested) Results: 조인된 결과물을 반복적으로 사용하여 내포된 결과 매핑을 사용하는 방법.

Nested Select 구문은 관계를 형성하기 위해 두 개의 조회 구문을 사용하여 association 의 select 속성으로 처리했지만, Nested Result 구문은 하나의 조회구문 내에서 직접 join 하는 방식을 사용한다는 점에서 차이를 보인다.

```
<select id="selectBlog" resultMap="blogResult">
  select
    B.id            as blog_id,
    B.title         as blog_title,
    B.author_id     as blog_author_id,
    A.id            as author_id,
    A.username      as author_username,
    A.password      as author_password,
    A.email         as author_email,
    A.bio           as author_bio
  from Blog B left outer join Author A on B.author_id = A.id
  where B.id = #{id}
</select>
```

BLOG 테이블과 AUTHOR 테이블의 데이터를 하나의 구문으로 조회하고 있기 때문에 association 프로퍼티를 채우기 위해 별도의 select 구문을 사용할 필요가 없어진다.

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <association property="author" column="blog_author_id" javaType="Author" resultMap="authorResult"/>
</resultMap>
```

```
<resultMap id="authorResult" type="Author">
  <id property="id" column="author_id"/>
  <result property="username" column="author_username"/>
  <result property="password" column="author_password"/>
  <result property="email" column="author_email"/>
  <result property="bio" column="author_bio"/>
</resultMap>
```

위 예제에서, Author 인스턴스를 로드하기 위한 “authorResult” resultMap 으로 위임된 Blog 의 “author” 관계를 볼 수 있을 것이다. id 요소는 내포된 결과 매핑에서 매우 중요한 역할을 담당하는데, 결과 중 유일한 것을 찾아내기 위한 한개 이상의 프로퍼티를 명시해야만 한다. 이를 위해선 두 개의 테이블간의 관계를 형성하기 위해 사용되는 식별관계를 형성하는 reference key 를 사용하는 것이 좋다. 위 예제는 관계를 매핑하기 위해 외부의 resultMap 요소를 사용했다. 이 외부 resultMap 은 Author resultMap 을 재사용가능하도록 해준다. 기호에 따라, 재사용할 필요가 없거나, 한개의 resultMap 에 결과 매핑을 함께 위치시키고자 한다면, association 결과 매핑을 내포시킬 수도 있다.

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <association property="author" javaType="Author">
    <id property="id" column="author_id"/>
    <result property="username" column="author_username"/>
    <result property="password" column="author_password"/>
    <result property="email" column="author_email"/>
    <result property="bio" column="author_bio"/>
  </association>
</resultMap>
```

association 은 optional 혹은 mandatory 관계의 테이블로부터 has one 관계의 데이터를 조회하는 경우에 사용되는데, 다음은 has-many관계의 데이터를 조회하는 방법을 살펴보도록 한다.

4) collection : has many 관계에 사용

“has many” 라는 관계를 확인하기 위해 하나의 블로그와 포스트의 관계를 생각해보자. 하나의 블로그는 하나의 소유자(author)를 갖지만(has-one), 다수의 포스트를 가질 수 있다(has-many). 이러한 객체 관계들을 갖는 블로그 도메인 모델로 작성된 자바빈 클래스는 다음과 같은 형태를 가질 것이다.

```
public class Blog {
        private String blog_id, blog_title;
        private Author author, coAuthor;
        private List<Post> posts;
        // ...
}
```

이러한 경우 Blog와 Post 의 관계를 has-many 로 정의할 수 있고, 이때 사용되는 resultMap 의 하위 요소가 collection 이다. collection 도 관계의 정의에 사용되는 요소이기 때문에 nested Select 나 nested Result를 모두 사용할 수 있으며, 사용 방법은 association 과 거의 동일하다.

① Collection 을 위한 내포된(Nested) Select

```
<resultMap id="blogResult" type="Blog">
  <collection property="posts" column="id" ofType="Post" select="selectPostsForBlog"/>
</resultMap>

<select id="selectBlog" resultMap="blogResult">
  SELECT * FROM BLOG WHERE ID = #{id}
</select>

<select id="selectPostsForBlog" resultType="Blog">
  SELECT * FROM POST WHERE BLOG_ID = #{id}
</select>
```

② Collection 을 위한 내포된(Nested) Results

```
<select id="selectBlog" resultMap="blogResult">
  select
  B.id as blog_id,
  B.title as blog_title,
  B.author_id as blog_author_id,
  P.id as post_id,
  P.subject as post_subject,
  P.body as post_body,
  from Blog B
  left outer join Post P on B.id = P.blog_id
  where B.id = #{id}
</select>
```

```
<resultMap id="blogResult" type="Blog">
  <id property="id" column="blog_id" />
  <result property="title" column="blog_title"/>
  <collection property="posts" ofType="Post">
    <id property="id" column="post_id"/>
    <result property="subject" column="post_subject"/>
    <result property="body" column="post_body"/>
  </collection>
</resultMap>
```

has-many 관계에 사용한다는 관계 정의가 달라질 뿐이지 사용하는 방식은 association과 거의 동일해서, 다중 테이블로부터 조회하는 구문을 직접 조인으로 처리하는 경우에 nested result 방식을 사용할 것이고, 마찬가지로 관계를 형성하는 키 역할을 하는 프로퍼티가 필요하기 때문에 id 요소를 사용한다. 또한 내포되던 resultMap 에 대해 재사용이 필요하다면 별도의 resultMap을 선언하고, collection요소에서 resultMap 속성으로 지정하여 사용하는 것도 가능하다.

5) discriminator :동적 결과 매핑을 위한 요소

종종 하나의 데이터베이스 쿼리는 많고 다양한 데이터 타입의 결과를 리턴한다.

```
<resultMap id="vehicleResult" type="Vehicle">
  <id property="id" column="id" />
  <result property="vin" column="vin"/>
  <result property="year" column="year"/>
  <result property="make" column="make"/>
  <result property="model" column="model"/>
  <result property="color" column="color"/>
  <discriminator javaType="int" column="vehicle_type">
    <case value="1" resultMap="carResult"/>
    <case value="2" resultMap="truckResult"/>
    <case value="3" resultMap="vanResult"/>
    <case value="4" resultMap="suvResult"/>
  </discriminator>
</resultMap>
```

discriminator 요소는 클래스 상속 관계를 포함하여 이러한 상황을 위해 고안되었다. discriminator 는 자바의 switch 와 같이 작동하기 때문에 이해하기 쉽다. discriminator 정의는 colume 과 javaType 속성을 명시한다. colume 은 MyBatis 로 하여금 비교할 값을 찾을 것이다. javaType 은 동일성 테스트와 같은 것을 실행하기 위해 필요하다. 이 예제에서, MyBatis 는 결과데이터에서 각각의 레코드를 가져와서 vehicle_type 값과 비교한다. 만약 discriminator 비교값과 같은 경우가 생기면, 이 경우에 명시된 resultMap 을 사용할 것이다. 해당되는 경우가 없다면 무시된다. 만약 일치하는 경우가 하나도 없다면, MyBatis 는 discriminator 블럭 밖에 정의된 resultMap 을 사용할 것이다. 이렇게 결과 집합 중 일부 데이터 따라 동적인 매핑을 처리하고 싶을 때 discriminator가 유용하게 쓰일수 있다.

6) cache

MyBatis 는 쉽게 설정가능하고 변경가능한 쿼리 캐싱 기능을 가지고 있다. MyBatis 3 캐시 구현체는 좀더 강력하고 쉽게 설정할 수 있도록 많은 부분이 수정되었다. 성능을 개선하고 순환하는 의존성을 해결하기 위해 필요한 로컬 세션 캐싱을 제외하고는 기본

적으로 캐시가 작동하지 않는다. 캐싱을 활성화하기 위해서, SQL 매핑 파일에 <cache /> 한줄을 추가하면 된다.

기본 캐시 설정을 추가하면 실행되는 구문에 대해 다음과 같은 특성이 반영된다.

- 매핑 구문 파일내 select 구문의 모든 결과가 캐싱된다.
- 매핑 구문 파일내 insert, update 그리고 delete 구문은 캐시 데이터를 지운다(flush).
- 기본적으로 캐싱에는 Least Recently Used (LRU) 알고리즘을 사용된다.
- 어플리케이션이 실행되는 캐시데이터가 유지되기 때문에 특정 시점에 사라지거나 하지않고, 시간 순서대로 지워진다는 보장도 없다.
- 캐시는 리스트나 객체에 대해 쿼리 메소드가 실행될때마다 최대 1024개까지 저장한다.
- 캐시는 읽기/쓰기가 모두 가능한다.

```
<cache
  eviction="FIFO"
  flushInterval="60000"
  size="512"
  readOnly="true"/>
```

더 많은 프로퍼티가 셋팅된 이 설정은 60초마다 캐시를 지우는 FIFO 캐시를 생성하고, 결과 리스트는 512개까지 저장하며, 각 객체는 읽기 전용으로 관리된다. 캐시 데이터를 변경하는 것은 개별 쓰레드에서 호출자간의 충돌을 야기할 수도 있기 때문에 읽기 전용으로 설정하는게 일반적이다. 각 설정은 다음과 같은 의미를 갖는다.

- eviction : 캐시 알고리즘 즉 캐시 데이터 제거에 사용하는 전략을 의미하는데, 기본값으로는 LRU가 사용되며, 설정 가능한 전략은 4가지 종류가 있다.

	설명
LRU	Least Recently Used , 오랫동안 사용하지 않는 캐시데이터를 먼저 제거
FIFO	First In First Out , 캐시에 들어온 순서대로 제거
SOFT	Soft Reference , 가비지 컬렉터의 상태와 강하지 않은 참조 규칙(메모리가 넉넉지 않을 때 GC의 대상이 됨)에 기초하여 객체를 제거
WEAK	Weak Reference , 가비지 컬렉터의 상태와 약한 참조 규칙(객체가 weak reference 만 가질 경우 GC 의 대상이 됨)에 기초하여 점진적으로 객체를 제거

<표 2-15> cache eviction전략의 종류

- flushInterval : 설정된 캐시를 얼마동안 유지할지 설정한다. ms 단위의 양수로 설정한다.
- size : 캐시에 저장할 객체수로 기본값은 1024이다. 가용 메모리에 따라 값을 다르게 설정한다.
- readOnly : 캐시데이터의 Read/Write 속성을 지정할며, 읽기만 가능한 경우 캐시데이터 변경은 허용되지 않으므로, 캐시 데이터를 반환할때도 원본을 반환하지만, 쓰기가 허용되는 경우 반환되는 데이터에 대한 변경이 가능하기 때문에 캐시의 복사본을 반환한다. 단순 조회는 readOnly설정이 빠르다.

캐시는 매퍼의 namespace 별로 설정하며, 다른 namespace에서 캐시 설정을 그대로

사용하고자 하는 경우는 <cache-ref namespace= “other.namspace “ />를 사용할 수 있다.

Mybatis의 캐시는 설정이 쉽고 단순하지만, 다음과 같은 제약을 갖고 있다.

- 로컬 캐시인 만큼 서버를 여러대 두고 서비스하는 경우 서버마다 캐시 내용이 다를 수 있다. 서버마다 캐시 내용을 동일하게 맞추기 위해서는 분산 캐시를 사용해야 한다.
- flushInterval을 설정하면 일정 시간 이후 캐시를 지우는 작업은 가능하지만, 스케쥴링 형태로 주기적인 캐시 데이터 교체등의 기능은 약하다.

Mybatis 이러한 제약사항 들을 해결하기 위해 Ehcache, Hazelcase, OsCache, Cacheonix 등 다른 캐시 제품과의 연동을 허용하고 있으며, 스프링과 같은 컨테이너를 사용하는 경우 컨테이너 자체로 캐시 처리를 위임하기도 한다.

참고로, EhCache와의 연동을 통한 캐시 설정 단계는 다음과 같다.

① 메이븐을 이용한 EhCache 연동 모듈 의존성 추가

```
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-ehcache</artifactId>
        <version>1.0.0</version>
</dependency>
```

<예제 2-6> ${basedir}/pom.xml 수정

② 특정 namespace 에 cache 설정 추가

```
<cache type="org.mybatis.caches.ehcache.EhcacheCache" />
```

<예제2-7> resources/kr/or/ddit/Mybatis/mappers/commons.xm

매퍼 XML에 다양한 요소들과 구문 설정에 관해 자세한 내용은 Mybatis XML 문서를 참고한다.

(2) 인터페이스를 이용한 구문 등록(Annotation을 이용한 Java Config 설정)

Mybatis에서는 XML 대신 어노테이션을 통해 필요한 설정이나 구문 등록등을 java config 로 대신 처리할 수 있는데, 이를 위한 어노테이션들과 각 특성에 대해 정리하면 다음과 같다.

Annotation	target	XML element	description
@CacheNamespace	class	<cache>	
@CacheNamespaceRef	class	<cache-ref>	
@ConstructorArgs	Method	<constructor>	결과를 자바 객체에 매핑할 때 생성자 주입방식을 사용한다.
@Arg	Method	<arg> <idArg>	<constructor> 의 하위 요소들인 <arg> 나 <idArg> 와 동일 id, column, javaType, jdbcType등의 속성을 사용
@TypeDiscriminator	Method	<disriminator>	결과 매핑 객체의 타입을 동적으로 선택.
@Case	Method	<case>	@TypeDiscriminator와 함께 사용되며, <discriminator>의 하위인 <case>와 동일
@Results	Method	<resultMap>	value로 @Result 어노테이션 배열을 갖는다.
@Result	Method	<result> <id>	one 속성은 has-one 관계 정의를 위한 <association> 기능을 하고, many 속성은 has-many 관계 정의를 위한 <collection> 기능을 한다.
@One	Method	<association>	공통적으로 select 속성을 통해 one/many 역할의 데이터를 조회하는 구문을 지정할 수 있으나, 직접 조인을 통한 resultMap 매핑은 순환 참조를 허용하지 않는 자바 어노테이션의 제약으로 인해 지원되지 않는다.
@Many	Method	<collection>	
@MapKey	Method		리턴타입이 Map 메소드에 사용되며, 결과 객체의 list를 객체의 프로퍼티중 하나를 키로 하는 Map으로 변환하기 위해 사용된다.
@Options	Method	매 핑 구 문 의 속성들	이 애노테이션은 매핑된 구문에 속성으로 존재하는 많은 분기(switch)와 설정 옵션에 접근할 수 있다. 각 구문을 복잡하게 만들기 보다, Options 애노테이션으로 일관되고 깔끔한 방법으로 설정 할 수 있게 한다. 사용가능한 속성들 : useCache=true, flushCache=false, resultSetType=FORWARD_ONLY, statementType=PREPARED, fetchSize=-1, timeout=-1, useGeneratedKeys=false, keyProperty="id", keyColumn="". 자바 애노테이션을 이해하는 것이 중요하다. 자바 애노테이션은 "null"을 셋팅 할 수 없다. 그래서 일단 Options 애노테이션을 사용하면 각각의 속성은 디폴트 값을 사용하게 된다. 디폴트 값이 기대하지 않은 결과를 만들지 않도록 주의해야 한다. keyColumn 은 키 칼럼이 테이블의 첫번째 칼럼이 아닌 특정 데이터베이스에서만(PostgreSQL 같은) 필요하다.

@Insert @Update @Delete @Select	Method	<insert> <update> <delete> <select>	애노테이션은 실행하고자 하는 SQL 을 표현한다. 각각 문자열의 배열(또는 한개의 문자열)을 가진다. 문자열의 배열이 전달되면, 각각 공백을 두고 하나로 합친다. 자바 코드에서 SQL 을 만들때 발행할 수 있는 "공백 누락" 문제를 해결하도록 도와준다.
@InsertProvider @UpdateProvider @DeleteProvider @SelectProvider	Method	<insert> <update> <delete> <select>	실행시 SQL 을 리턴할 클래스명과 메소드명을 명시하도록 해주는 대체수단의 애노테이션이다. 매핑된 구문을 실행할 때 MyBatis 는 클래스의 인스턴스를 만들고, 메소드를 실행한다. 메소드는 파라미터 객체를 받을 수도 있다. type : 클래스의 패키지 경로를 포함한 전체 이름 method: 클래스의 메소드
@Param	Parameter		매퍼 메소드가 여러개의 파라미터를 가진다면, 이 애노테이션은 이름에 일치하는 매퍼 메소드 파라미터에 적용된다. 반면에 여러개의 파라미터는 순서대로 명명된다. 예를 들어, #{param1}, #{param2} 등이 디폴트다. @Param("person") 를 사용하면, 파라미터는 #{person} 로 명명된다.
@SelectKey	Method	<selectKey>	
@ResultMap	Method		이 애노테이션은 @Select 또는 @SelectProvider 애노테이션을 위해 XML 매퍼의 <resultMap> 요소의 id 를 제공하기 위해 사용된다. XML 에 정의된 결과 매핑을 재사용하도록 해준다. 이 애노테이션은 @Results 나 @ConstructorArgs 를 오버라이드 할 것이다.
@ResultType	Method		이 애노테이션은 결과 핸들러를 사용할때 사용한다. 이 경우 리턴타입은 void이고 마이바티스는 각각의 레코드 정보를 가지는 객체의 타입을 결정하는 방법을 가져야만 한다. XML 결과매핑이 있다면 @ResultMap 애노테이션을 사용하자. 결과타입이 XML에서 <select> 엘리먼트에 명시되어 있다면 다른 애노테이션이 필요하지 않다. 결과타입이 XML에서 <select> 엘리먼트에 명시되어 있지 않은 경우에 이 애노테이션을 사용하자. 예를들어, @Select 애노테이션이 선언되어 있다면 메소드는 결과 핸들러를 사용할 것이다. 결과 타입은 void 여야만 하고 이 애노테이션(이나 @ResultMap)을 반드시 사용해야 한다. 이 애노테이션은 메소드 리턴타입이 void가 아니라면 무시한다.

<표 2-16> Mybatis Mapper Annotation 종류

Mybatis에서 다양한 어노테이션을 지원하기는 하지만, 순환참조 문제나 런타임에 어노테이션의 값 참조 등의 제약 사항이 많기 때문에 매핑에 관한 주 설정은 XML 방식을 사용하고, 어노테이션은 보조적인 설정에 사용하는 편이 효율적이다. 자세한 사항은 Mybatis Java API 문서를 참고하도록 하고, 실제 사용 방식은 학습 과정 중 실습 소스를

통해 학습하도록 한다.

```
@CacheNamespace(implementation=EhcacheCache.class)
public interface ICommonDAO {
        @Select({
                "SELECT PROPERTY_NAME, PROPERTY_VALUE, DESCRIPTION",
                "FROM DATABASE_PROPERTIES" })
        @Results({
                @Result(id=true, property="property_name",
column="PROPERTY_NAME"),
                @Result(property="property_value", column="PROPERTY_VALUE"),
                @Result(property="description", column="DESCRIPTION")})
        public List<DataBasePropertiesVO> retrieveProps( );
}
```

<예제 2-7> kr.or.ddit.chap02.CommonDAO

```
@Insert("insert into table3 (id, name) values(#{nameId}, #{name})")
@SelectKey(statement="call next value for TestSequence", keyProperty="nameId", before=true, resultType=int.
int insertTable3(Name name);
```

```
@Insert("insert into table2 (name) values(#{name})")
@SelectKey(statement="call identity()", keyProperty="nameId", before=false, resultType=int.class)
int insertTable2(Name name);
```

3. 동적 구문

데이터를 다루다보면 여러 가지 분기 처리에 의해 SQL을 동적으로 만드는 작업이 빈번히 발생한다. 자바 코드를 사용해서 동적 SQL을 만드는 일은 쉽지만, 분기 처리가 많아지면 가독성이 떨어진다는 문제가 있다. 가독성이 떨어지면 오타의 가능성을 높아지고 버그로 이어지기도 한다. 따라서 동적 SQL을 많이 다르는 것이 코드를 유지보수하는데 어려움을 주기도 한다. Mybatis가 지원하는 가장 강력한 기능인 동적 SQL을 처리하기 위해 Mybatis는 다음과 같은 방법을 제공한다.

- XML에서 동적 SQL을 위한 엘리먼트 사용
- Mybatis 의 구문빌더 API 사용
- JDBC를 사용할 때처럼 자바 코드로 SQL을 만드는 방식

교재에서는 이중 가장 일반적으로 사용되는 XML의 동적 SQL을 위한 엘리먼트를 사용하는 방식에 대해 학습하도록 한다.

동적 SQL을 위해 Mybatis 가 제공하는 엘리먼트의 종류들은 다음과 같다.

- if
- choose(when, otherwise)
- trim(where, set)
- foreach

그런데, 이러한 엘리먼트들에 대해 살펴보기 전에 먼저 알아야 할 것이 OGNL(Object Graph Navigation Language) 라는 언어이다. OGNL을 자바 객체의 프로퍼티값을 가져오거나 설정하기 위한 목적의 표현 언어로 JSTL 등에서 흔히 사용된다.

예를 들어 JSTL의 조건문 if 구문을 한번 살펴보자.

```
<c:if test="${obj.val != null }" ></c:if>
<c:if test="${obj.val == 0 }" ></c:if>
<c:if test="${obj.val1 < 0 && obj.val2 > 0 }" ></c:if>
```

이 조건문의 형식처럼 OGNL 의 객체 프로퍼티 접근 방법은 기존의 자바 개발자들의 객체 이용방식과 거의 흡사하다. 기존의 Mybatis 의 동적 SQL 지원 엘리먼트에서 바로 이 OGNL 표기법에 따라 문장을 기술하게 된다.

파트	예시
프로퍼티 접근, comment 객체의 userId 필드	comment.userId
객체의 메소드 호출, 현재 객체의 hashcode()호출	hashcode()
배열의 특정 요소 접근, comments 배열의 첫번째 요소	comments[0]

<표 2-17> OGNL 기본 문법

(1) if

동적 SQL을 처리할 때 가장 많이 사용되는 조건문에는 익숙한 키워드인 if 구문이 사용된다. OGNL의 단순한 문법을 사용하여 객체의 프로퍼티에 사용하기 때문에 작성이 간편하고 가독성이 향상된다. 그런데, if 구문을 사용하여 동적인 조건절을 완성하는 경우, 흔히 매퍼 파라미터의 존재 유무에 따라 다른 조건절이 완성되어야 하는 경우가 있다. 매퍼 파라미터가 문자열인 경우 유/무를 판단하기 위해서는 null 체크와 trim 이후의 length 까지 확인해야 하고, 이를 간단하게 처리하기 위한 유틸리티 메소드들을 만들어 사용하기도 한다. 이때 두가지 문제가 발생할 수 있는데, 첫번째는 if구문에서 OGNL 표기법을 사용하여 클래스의 정적메소드에 접근할 수 있어야 하고, 두번째는 매퍼 파라미터의 특정 프로퍼티에 접근하는 것이 아니라, 파라미터 자체의 유무를 확인하기 위해 매퍼 파라미터 자체를 참조할 수 있는 식별자가 필요하게 된다. 이러한 문제는 다음과 같은 방식으로 처리할 수 있다.

먼저, **정적 메소드의 접근은 OGNL 표기법에 따라 '@' 기호를 사용하여 처리**한다.

```
public List<Blog> findActiveBlogWithTitleLike(Blog blog);

<select id="findActiveBlogWithTitleLike" resultType="Blog">
  SELECT * FROM BLOG
  WHERE state = 'ACTIVE'
  <if test="@org.apache.commons.lang3.StringUtils@isNotEmpty(title)">
    AND title like #{title}
  </if>
</select>
```

매퍼 파라미터 자체에 대한 접근이 필요한 경우는 @Param 어노테이션을 매퍼 메소드의 파라미터에 선언하여 파라미터맵의 키를 만들어서 해결할 수 있다. 예를 들어, 아래 예제에서 title 은 findActiveBlogWithTitleLike 메소드로 전달된 Blog 객체의 프로퍼티나 혹은 맵의 키가 되는데, 만일 title자체가 findActiveBlogWithTitleLike 메소드의 파라미터로 넘어온다면 그때는 다음과 같이 @Param 어노테이션을 사용하여 매퍼 파라미터의 정확한 이름을 결정해 줘야 한다.

```
public List<Blog> findActiveBlogWithTitleLike(@Param("title") String title);

<select id="findActiveBlogWithTitleLike" parameterType="hashmap"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE state = 'ACTIVE'
  <if test="@org.apache.commons.lang3.StringUtils@isNotEmpty(title)">
    AND title like #{title}
  </if>
</select>
```

(2) choose, when, otherwise

자바의 switch~case~default 구문과 같은 다중 조건문에 사용되며 사용방법은 JSTL의 choose~when~otherwise와 동일하다.

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = 'ACTIVE'
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

(3) trim, where, set 엘리먼트

다음의 구문은 if 문에 있는 모든 조건을 만족하지 않는 경우, 혹은 두 번째 if 문의 조건만을 만족하는 경우에 구문 오류를 발생시킬 수 있다. 조건식이 없는 where 절이나 좌항이 없는 and 연산자의 사용 등 불안전한 구문으로 인한 구문 오류가 발생하는 것이다.

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE
  <if test="state != null">
    state = #{state}
  </if>
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
      state = #{state}
    </if>
    <if test="title != null">
      AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
      AND author_name like #{author.name}
    </if>
  </where>
</select>
```

이런 경우, 오른쪽의 예제처럼 직접적인 where절 대신 where 엘리먼트를 사용하여 해결할 수 있다. where 엘리먼트를 조건에 맞는 구문이 provider 에 의해 리턴되면 where 키워드를 추가해주고, 좌항이 없는 AND/OR 연산자가 삽입된다면 자동으로 지워준다.

만일 좀더 정교한 설정을 하고 싶다면 그때는 trim 엘리먼트를 사용할 수 있다.

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  ...
</trim>
```

set 엘리먼트는 update 구문의 set절에 동적 구문 완성을 위해 유용하게 쓰일 수 있다. 예를 들어 값의 유/무에 따라 컬럼의 수정 여부가 결정되는 경우, 다음과 같은 구문을 사용할 수 있다.

```
<update id="updateAuthorIfNecessary">
  update Author
    <set>
      <if test="username != null">username=#{username},</if>
      <if test="password != null">password=#{password},</if>
      <if test="email != null">email=#{email},</if>
      <if test="bio != null">bio=#{bio}</if>
    </set>
  where id=#{id}
</update>
```

(4) foreach

구문을 완성할 때 collection 의 요소들에 대해 반복적으로 접근해야 하는 경우, IN 연산자를 많이 사용하는데, 이때 다음과 같은 형태로 동적 쿼리를 작성할 수 있다.

```
<select id="selectPostIn" resultType="domain.blog.Post">
     SELECT *
     FROM POST P
     <where>
          <if test="list != null">
               ID in
               <foreach item="postId" index="index" collection="idList"
                        open="(" separator="," close=")">
                    #{postId}
               </foreach>
          </if>
     </where>
</select>
```

- collection : 값을 가지고 있는 객체에 대한 참조명. ex) 매퍼 파라미터명
- item : 컬렉션의 요소 하나를 참조하기 위한 이름.
- index : 컬렉션의 인덱스 참조
- open : 컬렉션에서 값을 가져와서 구문을 완성할 때 가장 앞에 붙일 문자
- close : 가장 마지막에 붙일 문자
- separator : 컬렉션의 요소와 요소 사이에 붙일 문자

이외에도 자주 사용하진 않지만, Sql Provider 어노테이션들과 SqlBuilder 등을 이용해 java config로 동적 구문을 등록하는 방법들도 있으니 Mybatis 구문빌더 문서를 참고하도록 한다.

학습 1	개발환경 구축
학습 2	JCF(Java Collection Framework)
학습 3	제너릭스와 어노테이션
학습 4	스레드
학습 5	입/출력
학습 6	네트워킹
학습 7	DB 연동 프로그래밍

학습 8 Servlet

8-1. Http Protocol

학습목표	• Http Protocol 대해서 설명할 수 있다.

필요 지식 /

1 Http 대하여

HTTP(HyperText Transfer Protocol는 WWW 상에서 정보를 주고받을 수 있는 프로토콜이다. 주로 HTML 문서를 주고받는 데에 쓰인다. TCP와 UDP를 사용하며, 80번 포트를 사용한다. 1996년 버전 1.0, 그리고 1999년 1.1이 각각 발표되었다.

HTTP는 클라이언트와 서버 사이에 이루어지는 요청/응답(request/response) 프로토콜이다. 예를 들면, 클라이언트인 웹 브라우저가 HTTP를 통하여 서버로부터 웹페이지나 그림 정보를 요청하면, 서버는 이 요청에 응답하여 필요한 정보를 해당 사용자에게 전달하게 된다. 이 정보가 모니터와 같은 출력 장치를 통해 사용자에게 나타나는 것이다. HTTP를 통해 전달되는 자료는 http:로 시작하는 URL(인터넷 주소)로 조회할 수 있다.

1. HTTP 프로토콜의 특징

(1) 비연결성(Connectionless) 프로토콜이다.

비연결성은 클라이언트와 서버가 연결을 맺은 후, 클라이언트 요청에 대해 서버가 응답을 하고나면, 맺고 있었던 연결을 종료시키는 것을 말한다. 잦은 연결/해제에 따른 오버헤드를 줄이고자 HTTP1.1 에서는 KeepAlive 속성을 사용할 수 있다.

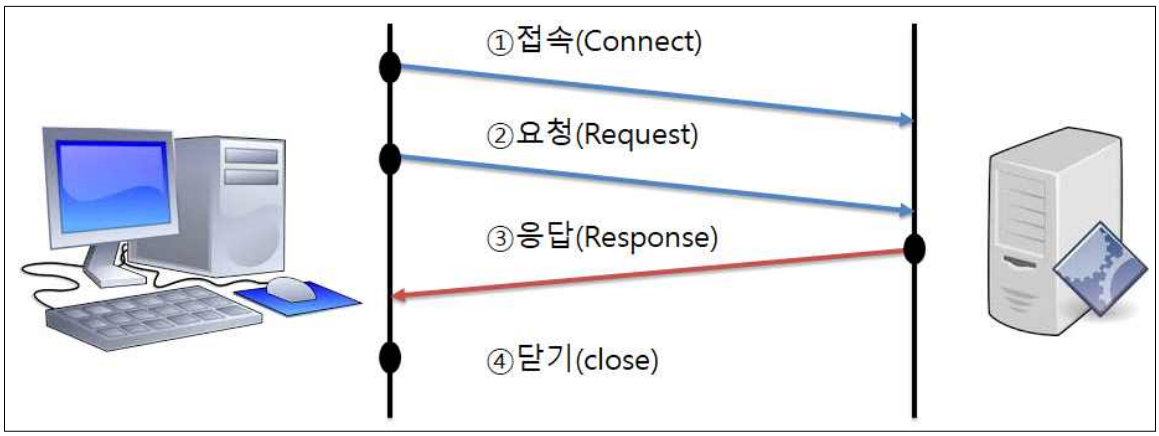
※ KeepAlive : 지정된 시간동안 서버와 클라이언트 사이에 패킷교환이 없는 경우, 상대방의 상태를 확인하기 위해 패킷을 주기적으로 보내고, 이에 대한 응답이 없으면 접속을 종료하는 것

(2) 상태가 없는(stateless) 프로토콜이다.

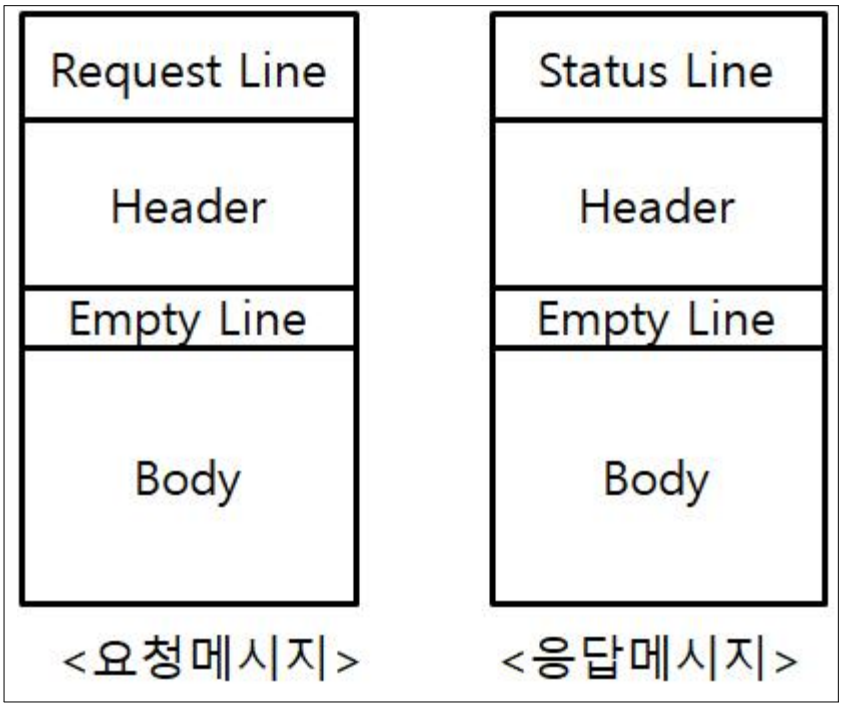
비연결성으로 인해 서버는 클라이언트를 식별할 수가 없는 상황을 말한다.

(3) TCP/IP 통신 위에서 동작 하며 기본포트는 80번이다.

2. HTTP 프로콜을 이용한 통신 과정



3. HTTP 요청(응답) 메시지 구조



(1) Request Line

예) GET /restapi/v1.0 HTTP/1.1

(2) Status Line

) HTTP/1.1 200 OK

(3) Header : 헤더정보를 의미함.

(4) Empty Line : 공백(빈줄)을 의미하고, Header와 Body를 분리 해주는 역할을 함.

(5) Body : 보내거나 받고자 하는 실제 데이터를 의미함.

※ 메서드가 GET 등의 경우에는 요청시 Body부분 생략 가능함.

4. 예제 세션

아래는 포트 80의 www.example.com에서 실행 중인 HTTP 클라이언트와 HTTP 서버 간의 샘플 변환이다. 모든 데이터는 줄 끝마다 2바이트 CR LF ('\r\n')를 사용하여 플레인 텍스트(ASCII) 인코딩을 통해 송신된다.

(1) 클라이언트 요청 예제

```
GET /restapi/v1.0 HTTP/1.1

Accept: application/json

Authorization: Bearer UExBMDFUMDRQV1MwMnzpdvtYYNWMSJ7CL8h0zM6q6a9ntw
```

(2) 서버 응답 예제

```
HTTP/1.1 200 OK

Date: Mon, 23 May 2005 22:38:34 GMT

Content-Type: text/html; charset=UTF-8

Content-Encoding: UTF-8

Content-Length: 138

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

ETag: "3f80f-1b6-3e1cb03b"

Accept-Ranges: bytes

Connection: close
```

```
<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

5. HTTP 요청 메서드

(1) GET : 존재하는 자원(리소스)에 대해 요청함.

(2) POST : 새로운 자원(리소스)을 생성을 요청함.

(3) PUT : 이미 존재하는 자원(리소스)에 대한 변경을 요청함.

(4) DELETE : 이미 존재하는 자원(리소스)에 대해 삭제를 요청함.

(5) HEAD : 이미 존재하는 자원(리소스)에 대해 헤더정보만을 요청함.

(6) OPTIONS : 서버 옵션들을 확인하기 위한 요청. CORS에서 사용함.

(7) TRACE: 클라이언트가 보낸 요청을 그대로 반환함.

(8) CONNECT: 프록시 터널링을 위해 예약된 메서드.

6. HTTP 상태 코드

	응답코드	설명
응답	100 Continue	클라이언트가 계속해서 요청을 하거나 이미 요청 완료한 경우 무시해도 됨.
	101 Switching Protocol	Upgrade 요청에 대한 응답으로 서버의 프로토콜 변경을 알려줌.
	102 Processing	서버가 수신하였지만, 아직 처리중이라 최종응답 블가상태임.
	103 Early Hints	주로 Link헤더와 사용되어 서버가 응답을 준비하는 동안 사용자 에이전트가 사전로딩(preloading)을 사작할수 있도록 한다.
성공 응답	200 OK	요청이 성공적으로 처리되었음.
	204 No Content	POST 및 PUT 요청이 성공적으로 처리되었고, 그 결과로 새로운 리소스가 생성되었음.
	205 Reset Content	요청을 완수한 이후에 사용자 에이전트에게 이 요청을 보낸 문서 뷰를 리셋하라고 알려줌.
	206 Partial Content	클라이언트에서 복수의 스트림을 분할 다운로드를 하고자 범위 헤더를 전송했기 때문에 사용됨.
리 다 이 렉 션 메시지	301 Moved Permanently	요청한 리소스의 URI가 변경되었음을 의미함. 새로운 URI가 응답에서 주어질 수 있음.
	303 See Other	클라이언트가 요청한 리소스를 다른 URI에서 GET요청을 통해 얻어야 할 때, 서버가 클라이언트로 직접 보내는 응답임.
	304 Not Modified	캐시를 목적으로 사용됨. 클라이언트의 응답이 수정되지 않았음을 알려주며, 계속해서 캐시된 응답을 사용할 수 있음.
클 라 이 언 트 에러 응답	400 Bad Request	잘못된 문법으로 인하여 서버가 요청을 이해할 수 없음.
	401 Unauthorized	인증에 실패했음을 의미함. 요청한 응답을 받기 위해 인증필요함.
	403 Forbidden	클라이언트는 콘텐츠에 접근권한이 없음. 401과 다른 점은 서버가 클라이언트가 누구인지 알고 있음.
	404 Not Found	요청받은 리소스를 찾을 수 없음. 리소스 숨기기 위해 403 대신 사용하기도 함.
	405 Method Not Allowed	요청한 메서드는 서버에서 알고 있지만, 사용할 수 없음.
	409 Conflict	요청이 현재 서버의 상태와 충돌됨.
서버 에러 응답	500 Internal Server Error	요청에 대한 서버 내부 처리 중 오류 발생함.
	501 Not Implemented	요청 메서드가 서버에서 지원되지 않아 처리할 수 없음.
	502 Bad Gateway	서버가 작업하는 동안 게이트웨이로부터 잘못된 응답을 수신함.
	503 Service Unavailable	서버가 요청처리할 준비가 되지 않았음. 서버작동이 중단된 경우나 과부하가 걸린 경우에 발생함.

< 1-1> HTTP 상태코드 종류

8-2. Servlet Programming

학습목표	• 프로그래밍에 대해서 설명할 수 있다.

필요 지식 / 프로그램(Java)언어 기본문법에 대한 이해 http 통신에 대한 이해

1 프로그래밍

1. 서블릿 기초

서블릿은 JSP 표준에 앞서 자바에서 웹 어플리케이션 개발을 위해 만들어진 표준으로, 규약에 따른 서블릿 개발 과정은 다음과 같다.

- 서블릿 규약에 따라 자바 코드를 작성(HttpServlet 상속 후 필요 메소드 재정의).
- 자바 코드를 컴파일하여 클래스 파일 생성.
- 클래스 파일을 /WEB-INF/classes 디렉토리에 패키지 구조에 따라 저장.
- web.xml 파일에 서블릿 클래스 등록 및 리퀘스트와의 매핑
  (서블릿 3.0 규약부터 @WebServlet 어노테이션으로 대체).
- 톰캣등의 컨테이너 재실행(서블릿 리로딩 기능이 있는 경우 생략)
- 웹 브라우저에서 요청 처리 결과 확인

(1) 서블릿 구현

```
public class NowServlet extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest request, HttpServletResponse
response)
				throws ServletException, IOException {
		response.setContentType("text/html;charset=UTF-8");
		response.isCommitted();
		PrintWriter out = response.getWriter();
		out.println("<html>");
		out.println("<head><title>현재시간</title></head>");
		out.println("<body>");
		out.println("현재 시간은 ");
		out.println(new Date());
		out.println("입니다.");
		out.println("</body></html>");
		out.flush();
		out.close();
	}
}
```

<예제 1-1-1> ddit/WEB-INF/src/chap02/NowServlet.java

- HTTP (method)별 구현 메소드

  GET : protected void doGet(HttpServletRequest, HttpServletResponse)
  thorws ServletException, IOException

  POST : protected void doPost(HttpServletRequest, HttpServletResponse)
  thorws ServletException, IOException

  요청의 의미와 요청데이터 패키징 방식을 구분하기 위한 방법으로 정의된 Http method 는 GET, POST, PUT, DELETE, HEAD, OPTION, TRACE 등이 있으며, 이중 대부분의 웹서버와 브라우저에서 지원하며 일반적인 웹에서 사용되는 method 는 GET 과 POST 가 있다. 각 메소드의 특징과 사용방법은 Chapter 04에서 다루게 된다.

- 서블릿의 callback 메소드 종류

  - lifeCycle 관련 메소드 : 생명주기내에서 단 한번씩 호출되는 메소드(init, destroy)
  - request 처리 관련 메소드 : 요청 발생시마다 각 쓰레드 내에서 반복 호출되는 메소드 (service 및 do□□□ 계열의 메소드)

  각 콜백 메소드의 호출 순서는 다음과 같다.

  객체 생성 직후 init 호출 → 리퀘스트 발생시 service 호출 → service 메소드내에서 현재 리퀘스트의 http method 구분후 해당 do□□□ 메소드 호출 → 객체 소멸 직전 destroy 호출

(2) 컴파일 및 /WEB-INF/classes 에 저장

```
%catalina_home%\webapps\ddit\WEB-INF\src> javac -d ..\classes
.\chap03\NowServlet.java
```

<예제 2-1-2> 컴파일 ddit/WEB-INF/classes/kr/or/ddit/chap02/NowServlet.class

(3) 서블릿 등록 및 매핑(web.xml, @WebServlet)

- web.xml 에 직접 등록 및 매핑(Servlet 2.5 까지의 방식)

  서블릿 등록(servlet 엘리먼트 사용) 설정 종류

  : 서블릿을 등록하는 과정에 설정된 모든 정보는 서블릿의 콜백 메소드내에서 ServletConfig 객체를 통해 접근 가능하다.

  - servlet-name : 컨테이너 내에서 관리되는 서블릿 인스턴스의 이름을 지정
  - servlet-class : 컨테이너 내에서 관리될 서블릿 클래스를 지정
  - init-param : 서블릿의 초기화 파라미터 설정(param-name, param-value 로 구성)
  - load-on-startup : 서블릿의 인스턴스 생성 순위 지정

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:jsp="http://java.sun.com/xml/ns/javaee/jsp"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="ddit" version="3.0">
        <servlet>
                <servlet-name>nowServlet</servlet-name>
                <servlet-class>kr.or.ddit.chap02.NowServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>nowServlet</servlet-name>
                <url-pattern>/now</url-pattern>
        </servlet-mapping>
</web-app>
```

<예제 2-1-3> ddit/WEB-INF/web.xml

- @WebServlet 사용한 자동 등록과 매핑

```java
@WebServlet("/streaming")
public class StreamingServlet extends HttpServlet {
        @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse
response)
                        throws ServletException, IOException {
                response.setContentType("audio/mpeg");
                String publicDir = System.getenv("PUBLIC");
                File source = new File(publicDir+ "₩₩Music₩₩Sample
Music", "Kalimba.mp3");
                InputStream is = new FileInputStream(source);
                OutputStream os = response.getOutputStream();
                byte[] buffer = new byte[1024];
                while(is.read(buffer)!=-1){
                        os.write(buffer);
                }
                is.close();
                os.flush();
                os.close();
        }
}
```

<예제 2-2> 어노테이션을 이용한 등록과 매핑

(ddit/WEB-INF/src/chap02/StreamingServlet.java)

** MIME(Multipurpose Internet Mail Extensions)

: 원래 목적은 이메일의 첨부되는 데이터의 타입(확장자)를 지정하기 위한 텍스트였으며, 현재는 메일 뿐만 아니라 HTTP 등의 프로토콜에서 응답 데이터의 타입을 설명하기 위한 목적으로 사용되고있다.

참고 : MIME 목록 (http://www.iana.org/assignments/media-types/index.html)

2. URL 패턴 매핑 규칙

웹 컨테이너에 등록된 서블릿은 웹상의 리퀘스트를 직접 처리할 수 있도록 url 과의 매핑

과정이 필요하다. web.xml 에 직접 등록과정을 거치는 경우, <url-pattern> 엘리먼트를 통해 매핑 설정을 하고, @WebServlet 을 사용하는 경우, 어노테이션의 속성으로 매핑 설정을 할수있다.

(1) 규칙의 종류

(가) '/' 로 시작하고 '/*' 로 끝나는 패턴은 경로 매핑을 위해 사용.

(나) '*.' 로 시작하는 패턴은 확장자에 대한 매핑을 할때 사용.

(다) 오직 '/' 만 포함하는 경우 어플리케이션의 기본 서블릿으로 매핑

(라) 이 규칙 외, 나머지 다른 문자열은 정확한 매핑을 위해 사용

4. URL pattern	5. 매핑 서블릿
/foo/bar/*	servlet1
/baz/*	servlet2
/catalog	servlet3
*.bop	servlet4

6. < 2-1> 예시 서블릿 매핑 규칙

7. 요청 경로	8. 일치 URL 패턴	9. 요청 처리 서블릿
/foo/bar/index.html	/foo/bar/*	servlet1
/foo/bar/index.bop	/foo/bar/*	servlet1
/baz	/baz/*	servlet2
/baz/index.html	/baz/*	servlet2
/catalog	/catalog	servlet3
/catalog/racecar.bop	*.bop	servlet4
/index.bop	*.bop	servlet4

10. <표 2-2> 경로 및 확장자에 따른 서블릿 매핑

3. 서블릿 동작 과정(Servlet Container 의 역할)

(1) 서블릿 요청 처리 과정

등록 및 매핑 과정을 거친 서블릿이 매핑된 url을 사용한 요청을 처리하는 과정은 다음과 같다.

매핑 url 패턴에 맞는 리퀘스트 발생 → 서블릿 컨테이너에서 매핑 서블릿의 인스턴스 검색

있다면 : 해당 서블릿의 콜백 메소드 호출

없다면 : 해당 서블릿의 인스턴스 생성 → 서블릿의 콜백 메소드 호출

(2) 서블릿 동작 과정에서 서블릿 컨테이너의 역할

Java EE 표준에 따라 구현된 웹 어플리케이션 서버는 그 명세나 구현체가 지나치게 무거워지고 구현 방법이 필요이상으로 복잡해졌다는 회의론이 등장했다, 그러한 추세에서 스프링과 비슷한 경량 프레임워크들이 등장했으나 이러한 프레임워크들 역시 Java EE 중 웹 어플

리케이션 기술 위에서 동작해야 하며, 웹 어플리케이션 기술에 대한 구현체가 곧 서블릿 컨테이너이다.

서블릿 컨테이너는 웹 어플리케이션 서버에서 Http 요청을 받아 처리하는 기초 역할을 담당한다. 앞서 살펴본 서블릿의 동작 절차를 보면, 등록된 url-pattern에 매핑되는 리퀘스트를 서블릿에게 처리 위임하기 위해 인스턴스를 생성 및 관리하며 적절한 형태의 콜백 메소드(service)를 호출해주는 역할을 담당하는 것이 곧 서블릿 컨테이너라 할 수 있다.

4. 서블릿의 단점 및 JSP 사용 이유

앞서 서블릿을 실행 코드 방식으로 분류했듯이 서블릿은 개발하는데 몇가지 단점을 가지고 있다. 첫째, 스크립트 방식에 비해 구현 후 반드시 컴파일 및 등록 과 매핑이라는 과정을 거쳐야 한다. 둘째, 서블릿 리로딩 기능이 없는 WAS 의 경우에, 등록 후 웹 리퀘스트를 받기 위해서는 컨테이너를 재시작 해야 하는 단점이 있다. 셋째, 어플리케이션 구현 소스 내에 UI 구성 스크립트 소스가 들어가므로 웹 디자이너와 웹 개발자 사이의 역할 분담 및 협업이 자연스럽지 못한 단점이 있다. 이러한 단점들을 해결하기 위해 클라이언트 사이드 스크립트 코드와 섞어서 개발하기 쉽고, 스크립트 방식을 사용하고 있는 JSP의 장점이 부각되었다.

8-3. Servlet과 JDBC

학습목표	• Servlet 이용한 JDBC 프로그램을 작성할 수 있다.

필요 지식 /

1 Servlet JDBC

1. Servlet을 이용한 JDBC 프로그래밍

기존에 iBatis를 이용해 작성한 Java애플리케이션인 회원관리 프로그램을 Serlvet 프로그램으로 변경해 보도록 한다.

(1) 프로젝트 생성하기

Dynamic Web 프로젝트를 생성 후 기존에 작성한 회원관리 프로그램 소스 중 DAO 및 Service관련 소스를 적절한 위치에 import 시킨다.

(2) 필요한 DB 정보 생성

기존에 만들어 둔 테이블을 그대로 사용할 예정이므로 별도의 추가 작업은 필요하지 않다.

(3) 전체 회원조회를 위한 서블릿 생성하기(SelectAllMemberServlet.java)

```
String msg = req.getParameter("msg") == null ? "" : req.getParameter("msg");

// 1. 서비스 객체 생성하기
IMemberService memberService = MemberServiceImpl.getInstance();

// 2. 회원정보 조회
List<MemberVO> memList = memberService.getAllMemberList();

// 3. 결과를 화면(브라우져)에 출력하기
resp.setCharacterEncoding("utf-8");
resp.setContentType("text/html");
```

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

out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>회원 목록</title>");
out.println("</head>");
out.println("<body>");
out.println("<table border=\"1\">");
out.println("<tr>");
out.println("<td colspan=\"4\"><a href=\"insertMemberForm\">[회원 등록]</a></td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>ID</td>");
out.println("<td>이름</td>");
out.println("<td>전화번호</td>");
out.println("<td>주소</td>");
out.println("</tr>");

int memSize = memList.size();
if (memSize > 0) {
    for (int i = 0; i < memSize; i++) {
        out.println("<tr>");
        out.println("<td>" + memList.get(i).getMem_id() + "</td>");
        out.println("<td>");
        out.println("<a href=\"viewMember?memId=" + memList.get(i).getMem id() + "\">
" + memList.get(i).getMem_name() + "</a>");
        out.println("</td>");
        out.println("<td>" + memList.get(i).getMem_tel() + "</td>");
        out.println("<td>" + memList.get(i).getMem_addr() + "</td>");
        out.println("</tr>");
    }
} else {// 회원정보가 존재하지 않으면...
    out.println("<tr>");
    out.println("<td colspan=\"4\">회원정보가 없습니다.</td>");
    out.println("</tr>");
}

out.println("</table>");

if (msg.equals("성공")) { // 성공메시지가 전달되면...
    out.println("<script>");
    out.println("alert('정상적으로 처리되었습니다.')");
    out.println("</script>");
}

out.println("</body>");
out.println("</html>");
```

(4) 위한 서블릿 생성하기

(가) 회원등록 폼 서블릿 생성(InsertMemberFormServlet.java)

```
// 1. 회원등록 폼을 화면(브라우저)에 출력하기
resp.setCharacterEncoding("utf-8");
resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>신규회원 등록</title>");
out.println("</head>");
out.println("<body>");
out.println("<form action=\"insertMember\" method=\"post\">");
out.println("<table>");
out.println("<tr>");
out.println("<td>I D:</td>");
out.println("<td><input type=\"text\" name=\"memId\" value=\"\"></td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>이름:</td>");
out.println("<td><input type=\"text\" name=\"memName\" value=\"\"></td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>전화번호:</td>");
out.println("<td><input type=\"text\" name=\"memTel\" value=\"\"></td>");
out.println(" </tr>");
out.println("<tr>");
out.println(" <td>주소:</td>");
out.println("<td><textarea name=\"memAddr\" ></textarea></td>");
out.println("</tr>");
out.println("</table>");
out.println("<input type=\"submit\" value=\"회원 등록\">");
out.println("</form>");
out.println("</body>");
out.println("</html>");
```

( ) 회원등록 서블릿 생성(InsertMemberServlet.java)

```
// 1. 요청파라미터 정보 가져오기
String memId = req.getParameter("memId");
String memName = req.getParameter("memName");
String memTel = req.getParameter("memTel");
String memAddr = req.getParameter("memAddr");

// 2. 서비스 객체 생성하기
IMemberService memberService = MemberServiceImpl.getInstance();

// 3. 회원정보 등록
MemberVO mv = new MemberVO();
mv.setMem_id(memId);
mv.setMem_name(memName);
mv.setMem_tel(memTel);
mv.setMem_addr(memAddr);

int cnt = memberService.insertMember(mv); // 회원등록

String msg = "";
if(cnt > 0) {
    msg = "성공";
}else {
    msg = "실패";
}

// 4. 목록 조회화면으로 이동
String redirectUrl = req.getContextPath() + "/displayMemberAll?msg=" + URLEncoder
.encode(msg, "utf-8");
resp.sendRedirect(redirectUrl); // 목록조회화면으로 리다이렉트
```

(4) 위한 서블릿 생성하기

(가) 회원수정 폼 서블릿 생성(UpdateMemberFormServlet.java)

```
String memId = req.getParameter("memId");

// 1. 서비스 객체 생성하기
IMemberService memberService = MemberServiceImpl.getInstance();

// 2. 회원정보 조회
MemberVO mv = new MemberVO();
mv.setMem_id(memId);
List<MemberVO> memList = memberService.getSearchMember(mv);

// 3. 회원등록 폼을 화면(브라우져)에 출력하기
resp.setCharacterEncoding("utf-8");
resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>회원정보 수정</title>");
out.println("</head>");
out.println("<body>");
out.println("<form action=\"updateMember\" method=\"post\">");
out.println("<table>");
out.println("<tr>");
out.println("<td>I D:</td>");
out.println("<td><input type=\"text\" name=\"memId\" value=\""
+ memList.get(0).getMem_id() + "\"></td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>이름:</td>");
out.println("<td><input type=\"text\" name=\"memName\" value=\""
+ memList.get(0).getMem_name() + "\"></td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>전화번호:</td>");
out.println("<td><input type=\"text\" name=\"memTel\" value=\""
+ memList.get(0).getMem_tel() + "\"></td>");
out.println(" </tr>");
out.println("<tr>");
out.println(" <td>주소:</td>");
out.println("<td><textarea name=\"memAddr\" >"
+ memList.get(0).getMem_addr() + "</textarea></td>");
out.println("</tr>");
out.println("</table>");
out.println("<input type=\"submit\" value=\"회원정보 수정\">");
out.println("</form>");
out.println("</body>");
out.println("</html>");
```

( ) 회원수정 서블릿 생성(UpdateMemberServlet.java)

```
// 1. 요청파라미터 정보 가져오기
String memId = req.getParameter("memId");
String memName = req.getParameter("memName");
String memTel = req.getParameter("memTel");
String memAddr = req.getParameter("memAddr");

// 2. 서비스 객체 생성하기
IMemberService memberService = MemberServiceImpl.getInstance();

// 3. 회원정보 수정
MemberVO mv = new MemberVO();
mv.setMem_id(memId);
mv.setMem_name(memName);
mv.setMem_tel(memTel);
mv.setMem_addr(memAddr);

int cnt = memberService.updateMember(mv); // 회원정보 수정

String msg = "";
if(cnt > 0) {
    msg = "성공";
}else {
    msg = "실패";
}

// 4. 목록 조회화면으로 이동
String redirectUrl = req.getContextPath() + "/displayMemberAll?msg=" + URLEncoder
.encode(msg, "utf-8");
resp.sendRedirect(redirectUrl); // 목록조회화면으로 리다이렉트
```

(5) 회원정보 확인을 위한 서블릿 생성(ViewMemberServlet.java)

```
String memId = req.getParameter("memId");

// 1. 서비스 객체 생성하기
IMemberService memberService = MemberServiceImpl.getInstance();

// 2. 회원정보 조회
MemberVO mv = new MemberVO();
mv.setMem_id(memId);
List<MemberVO> memList = memberService.getSearchMember(mv);

// 3. 결과를 화면(브라우저)에 출력하기
resp.setCharacterEncoding("utf-8");
resp.setContentType("text/html");
```

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

out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>회원 정보</title>");
out.println("</head>");
out.println("<body>");
out.println("<form action=\"insertMember\" method=\"post\">");
out.println("<table border='1'>");
out.println("<tr>");
out.println("<td>I D:</td>");
out.println("<td>" + memList.get(0).getMem_id() + "</td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>이름:</td>");
out.println("<td>" + memList.get(0).getMem_name() + "</td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td>전화번호:</td>");
out.println("<td>" + memList.get(0).getMem_tel() + "</td>");
out.println(" </tr>");
out.println("<tr>");
out.println(" <td>주소:</td>");

String memAddr = memList.get(0).getMem_addr();
// lineSeparator를 이용하여 <br>태크로 변경함.
memAddr = memAddr.replaceAll(System.lineSeparator(), "<br>");

out.println("<td>" + memAddr + "</td>");
out.println("</tr>");
out.println("<tr>");
out.println("<td colspan=\"2\">");
out.println("<a href=\"displayMemberAll\">[목록]</a>");
out.println("<a href=\"updateMemberForm?memId="
            + memList.get(0).getMem_id() + "\">[회원정보수정]</a>");
out.println("<a href=\"deleteMember?memId="
           + memList.get(0).getMem_id() + "\">[회원정보삭제]</a>");
out.println("</td>");
out.println("</tr>");
out.println("</table>");
out.println("</form>");
out.println("</body>");
out.println("</html>");
```

(6) 삭제를 위한 서블릿 생성 (DeleteMemberServlet.java)

```
// 1. 요청파라미터 정보 가져오기
String memId = req.getParameter("memId");

// 2. 서비스 객체 생성하기
IMemberService memberService = MemberServiceImpl.getInstance();

// 3. 회원정보 삭제
int cnt = memberService.deleteMember(memId); // 회원정보 삭제

String msg = "";
if(cnt > 0) {
    msg = "성공";
}else {
    msg = "실패";
}

// 4. 목록 조회화면으로 이동
String redirectUrl = req.getContextPath() + "/displayMemberAll?msg=" + URLEncoder
.encode(msg, "utf-8");
resp.sendRedirect(redirectUrl); // 목록조회화면으로 리다이렉트
```

참고자료

아래의 자료를 제공하신 원저자들에게 깊은 감사를 표합니다.

1. 자바 프로그래밍 100% 실전가이드, 심상원 저.
2. https://ko.wikipedia.org/ 위키피디아 사이트
3. 기타 웹문서들

NCS 학습모듈 개발진

(산업체) (교육훈련기관) (연구기관)

*표시는 NCS 개발진임

NCS 학습모듈 검토진

(산업체) (교육훈련기관)

*표시는 NCS 개발진임