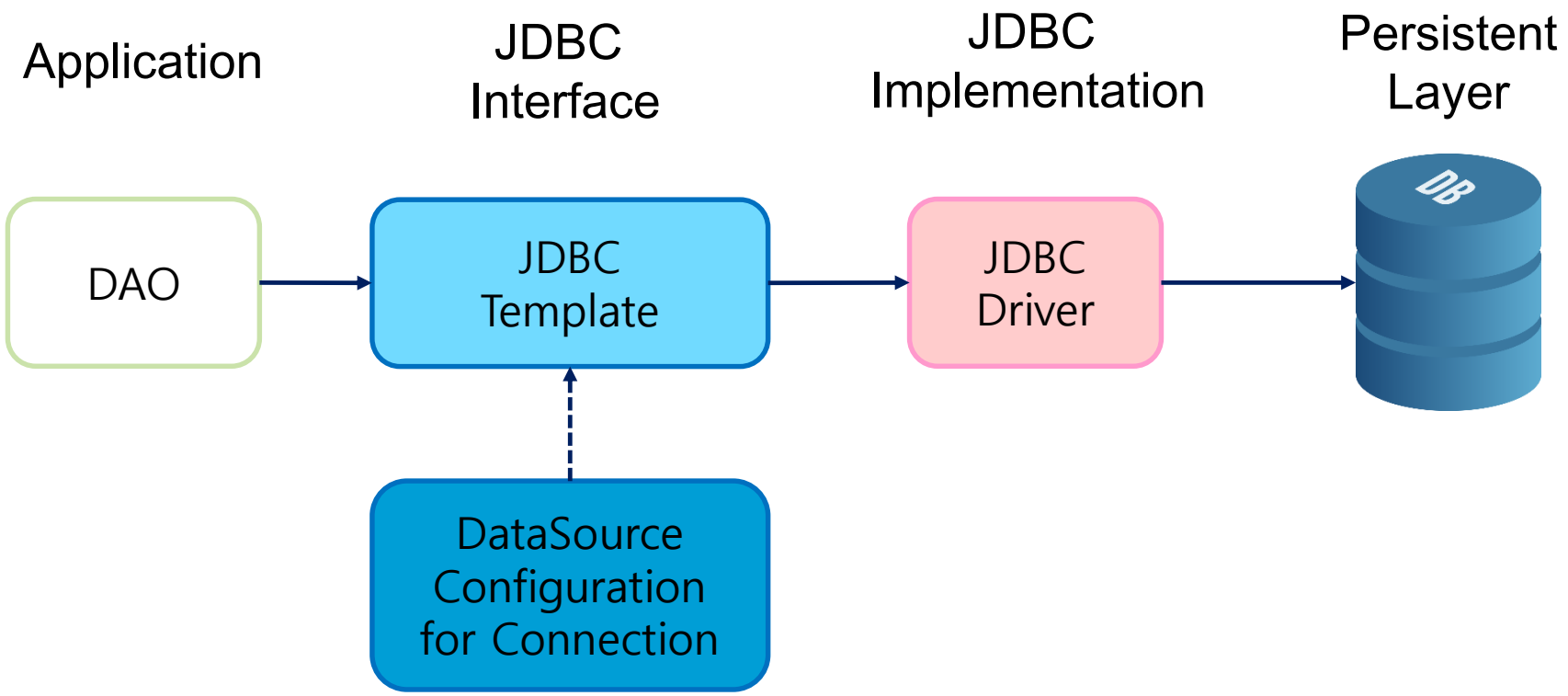
Spring JdbcTemplate, JPA

558280-1
2026년 봄학기
5/7/2026
박경신

Data Access Layer



JDBC, DAO, DTO, DataSource

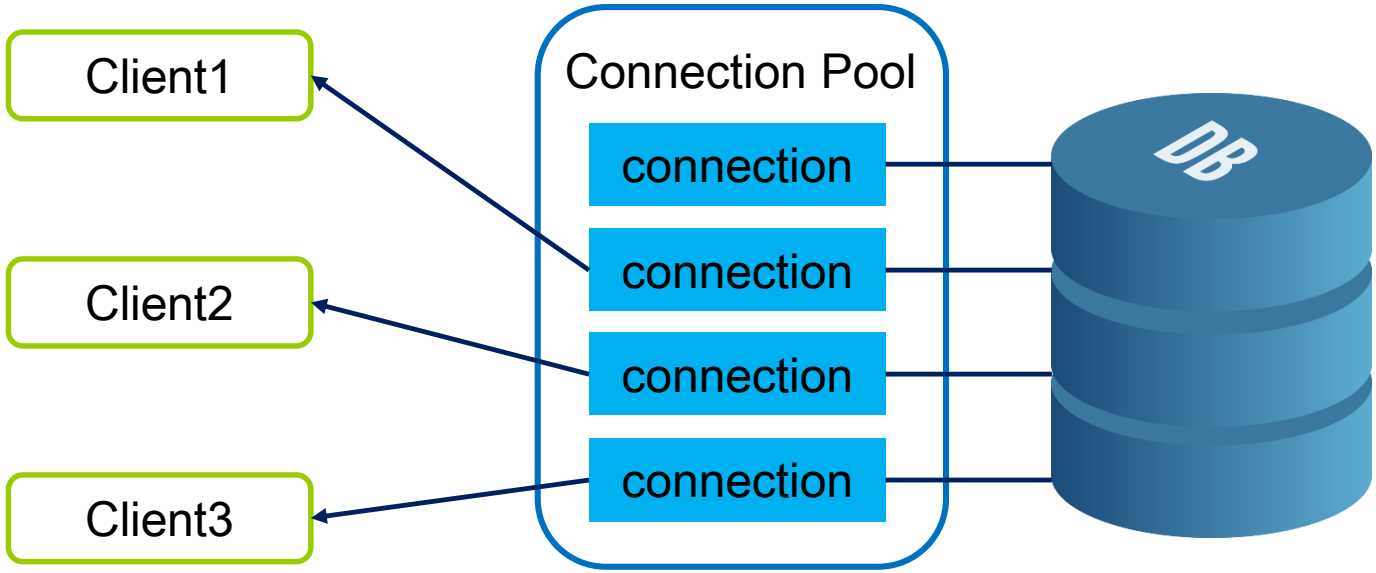
- JDBC (Java Database Connectivity)
  - 자바를 이용해 DB에 접근하기 위한 컨트롤 인터페이스
- DAO (Data Access Object)
  - 실제로 커넥션을 통해 데이터베이스에 접근하는 객체
  - 데이터베이스에 들어있는 모델을 객체로 추상화한 인터페이스
    - DB에 새로운 레코드를 생성 = 객체 생성
    - 레코드의 값 변경 = 객체.setValue
- DTO (Data Transfer Object) / VO(Value Object)
  - 데이터 그 자체로 의미를 담고 있는 객체
    - DB의 레코드에 대응되는 자바 클래스
- DataSource
  - JDBC 명세의 일부분이면서 일반화된 DB 연결 담당
    - DB와 관계된 Connection 정보를 가지고 있으며 bean으로 등록하여 인자로 넘겨주는 과정을 통해 스프링은 DataSource로 DB와 연결
    - Connection Pool, JNDI, DriverManager 방식 등

Connection

- Connection
  - 애플리케이션과 데이터베이스의 연결을 말함
  - 애플리케이션에서 새로운 DB 연결은 시스템 부하의 한 요소
  - 대규모 시스템일수록 커넥션 관리는 중요함

Connection Pool

- Connection Pool
  - DB와 커넥션을 맺고 있는 객체들을 Pool에 저장해두었다가, Client 요청이 오면 빌려주고, 처리가 끝나면 다시 Pool에 반납
  - 별도의 프로세스로 데이터베이스 커넥션을 관리
  - 일정 수준 이상의 커넥션을 유지하기 때문에 최적의 성능을 보장
  - 사용하지 않는 커넥션의 자동관리 및 최대 접속에 대한 제한
  - Apache Commons DBCP2, Tomcat-JDBC, Bone CP, HikariCP, c3p0



Hikari CP

- Hikari CP
  - 데이터베이스 연결(Connection)을 관리해주는 도구
  - 커넥션풀 (Connection Pool)이 설정된 커넥션의 사이즈만큼의 연결을 허용
  - HTTP 요청에 대해 순차적으로 DB 커넥션을 처리해주는 기능을 수행
  - SpringBoot 2.0 이전에는 tomcat-jdbc 사용하다가 **2.0 이후 HikariCP를 기본옵션으로 채택**

DataSource

- DataSource
  - JNDI(Java Naming and Directory Interface)통해 DataSource를 제공
  - DataSource.getConnection()메서드를 통해 Connection 을 얻음
  - 사용한 커넥션은 close()메서드를 이용해 반환함

JDBC Connection Pool & DataSource

Property	Value	Description
Data Source URL	**jdbc:mysql:@localhost:3306/sampledb?characterEncoding=UTF-8**	DB 접속을 위한 URL 정보를 입력
JDBC Driver Class	**com.mysql.cj.jbdc.Driver**	JDBC 클래스 로딩을 위한 클래스 정보
User Name		DB 사용자 계정
Password		DB 계정의 비밀번호
Max Active Connections	**30**	커넥션 풀에서 관리할 최대 동시 연결 수 지정. Max 값 이상의 연결을 못 만듦
Max Idle Connections	**3**	최대로 유지할 유휴 연결 개수. 설정값 이상의 커넥션은 자동으로 정리됨
Max Wait for Connection	**50000**	Milliseconds (default: 50초) DB 연결을 구하기 위한 대기 시간

DB 접근 템플릿

- Connection 생성
- try {
  Connection에 쿼리 전송
  결과 객체에서 적절하게 추출, 사용
  } catch {
  알맞은 Exception Handling
  } finally {
  연결 종료
  }

Spring JdbcTemplate

- pom.xml에 JDBC와 MySQL dependency 추가

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
</dependency>
```

Spring JdbcTemplate

- JdbcTemplate 은 Spring JDBC 접근 방법 중 하나로, 템플릿 메소드/전략 패턴 사용
  - Connection 관리, Statement, ResultSet, 예외 처리 등을 자동화 해줌
- 내부적으로 Plain JDBC API를 사용하지만 Plain JDBC 문제점을 개선한 스프링에서 제공하는 클래스
- 전형적인 Spring JDBC 접근 방법
- insert, update, select 쿼리를 직접 작성하는 방법을 제공
- 개발자가 할 일
  - SQL 구문 작성
  - 결과 처리

DAO 만들기

- 이 모델이 갖춰야 할 인터페이스만 정의

```
public interface PetRepository {
    // insert into pet (name,owner,type,gender) values(?,?,?,?)
    int insert(Pet pet);
    // update pet set name=? owner=? type=? gender=? where id = ?
    int update(Pet pet);
    // delete pet where id = ?
    int deleteById(Long id);
    // select * from pet
    List<Pet> findAll();
    // select * from pet where id = ?
    Optional<Pet> findById(Long id);
    // select * from pet where name = ?
    Optional<Pet> findByName(String name);
    // select * from pet where owner = ?
    List<Pet> findByOwner(String owner);
    // select * from pet where type = ? and gender = ?
    List<Pet> findByTypeAndGender(String type, Gender gender);
}
```

DTO 구현하기

- DAO를 통해 받아올 정보를 담을 구상 객체
- 컬럼으로 id, name, owner, type, gender 가지고 있음

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Pet {
        private Long id;
        private String name;
        private String owner;
        private String type;
        private Gender gender;
        public Pet(String name, String owner, String type, Gender gender) {
                this.name = name;
                this.owner = owner;
                this.type = type;
                this.gender = gender;
        }
}
```

DAO 빈 구현하기

- PetDAOImpl하는 객체를 생성하고 이를 bean으로 등록

```
@RequiredArgsConstructor
@Repository
public class PetJdbcTemplateRepository implements PetRepository {
    private final JdbcTemplate jdbcTemplate; // 기본제공하는 jdbcTemplate 주입
    @Override
    public int insert(Pet pet) {
        return jdbcTemplate.update(
                "insert into pet (name,owner,type,gender) values(?,?,?,?)",
                pet.getName(), pet.getOwner(), pet.getType(),
pet.getGender().toString());
    }
    @Override
    public int update(Pet pet) {
        return jdbcTemplate.update(
                "update pet set name=?, owner=?, type=?, gender=? where id = ?",
                pet.getName(), pet.getOwner(), pet.getType(),
pet.getGender().toString(), pet.getId());
    }
```

각 컬럼에 대응

메시지 객체를 인자로 받음

DAO 빈 구현하기

```
@Override
public int deleteById(Long id) {
    return jdbcTemplate.update(
            "delete from pet where id = ?", id);
}
@Override
public List<Pet> findAll() {
    return jdbcTemplate.query(
            "select * from pet",
            (rs, rowNum) ->
                    new Pet(
                            rs.getLong("id"),
                            rs.getString("name"),
                            rs.getString("owner"),
                            rs.getString("type"),
                            Gender.valueOf(rs.getString("gender")))
    );
}
```

List<Pet> JdbcTemplate.query(String sql, **RowMapper<Pet> rowMapper**)

DAO 빈 구현하기

```
@Override
public Optional<Pet> findById(Long id) {
    return jdbcTemplate.queryForObject(
            "select * from pet where id = ?",
            new Object[]{id},
            (rs, rowNum) ->
                    Optional.of(new Pet(
                            rs.getLong("id"),
                            rs.getString("name"),
                            rs.getString("owner"),
                            rs.getString("type"),
                            Gender.valueOf(rs.getString("gender")))
                    )
    );
}
```

Optional<Pet>
JdbcTemplate.queryForObject(String sql,
@Nullable Object[] args,
RowMapper<Optional<Pet>> rowMapper)

DAO 빈 구현하기

- JdbcTemplate 참조
  - https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html

```
public List<Pet> findAll() {
        String sql = "select * from pet";
        List<Pet> petList = jdbcTemplate.query(sql, new RowMapper<Pet>() {
            @Override
            public Pet mapRow(ResultSet rs, int rowNum) throws SQLException {
                Pet pet = new Pet();
                pet.setId(rs.getLong("id"));
                pet.setName(rs.getString("name"));
                pet.setOwner(rs.getString("owner"));
                pet.setType(rs.getString("type")));
                pet.setGender(Gender.valueOf(rs.getString("gender")));
                return pet;
            }
        });
        return petList;
}
```

RowMapper, ResultSetExtractor 인터페이스

- RowMapper<T> 인터페이스
  - **T mapRow(ResultSet rs, int rowNum)** 구현
    - 쿼리 결과물 rs
    - 전체 row 수 rowNum
    - 쿼리 결과물을 객체 타입으로 변환해주는 역할
  - 여기서는 Pet 사용하므로
    public Pet mapRow(ResultSet rs, int rowNum) throws SQLException {
    와 같이 구현
- ResultSetExtractor<T> 인터페이스
  - **T extractData(ResultSet rs)** 구현
    - 쿼리 결과물 rs
    - 쿼리 결과물을 객체 타입으로 변환해주는 역할

빈에 접근하기

- Service
  - PetDAO 빈 인스턴스인 petJdbcTemplateRepository 자동 주입
  - 이제 DAO를 통해 데이터베이스 접근 가능

```
@RequiredArgsConstructor
@Service
public class PetService {
    private final PetRepository repository; // repository 주입
    public int insert(Pet pet) {
        return repository.insert(pet);
    }
    public int update(Pet pet) {
        return repository.update(pet);
    }
    public int deleteById(Long id) {
        return repository.deleteById(id);
    }
    public List<Pet> findAll() {
        return repository.findAll();
    } …
```

빈에 접근하기

```
@RequiredArgsConstructor
@RestController
@RequestMapping("/pet")
public class PetController {
    private final PetService service; // service 주입
    // get a list of pet
    @GetMapping("/list")
    public List<Pet> getAll() {
        return service.findAll();
    }
    // get a pet by id
    @GetMapping("/id/{id}")
    public Pet getById(@PathVariable("id") long id) {
        return service.findById(id).orElse(null);
    }
    // get a pet by name
    @GetMapping("/name/{name}")
    public Pet getByName(@PathVariable("name") String name) {
        return service.findByName(name).orElse(null);
    } …
```

실행 결과

```
C:\Users\park>curl -s http://localhost:8080/pet/list
[{"id":1,"name":"Fluffy","owner":"Harold","type":"cat","gender":"FEMALE"},{"id":2,"
name":"Claws","owner":"Gwen","type":"cat","gender":"MALE"},{"id":3,"name":"Buf
fy","owner":"Harold","type":"dog","gender":"FEMALE"},{"id":4,"name":"Fang","ow
ner":"Benny","type":"dog","gender":"MALE"},{"id":5,"name":"Bowser","owner":"Di
ane","type":"dog","gender":"MALE"},{"id":6,"name":"Chirpy","owner":"Gwen","typ
e":"bird","gender":"FEMALE"},{"id":7,"name":"Whistler","owner":"Gwen","type":"bi
rd","gender":"FEMALE"},{"id":8,"name":"Slim","owner":"Benny","type":"snake","g
ender":"MALE"},{"id":9,"name":"Kong","owner":"Ranhee","type":"turtle","gender":"
FEMALE"},{"id":10,"name":"Puffball","owner":"Diane","type":"hamster","gender":"
FEMALE"},{"id":11,"name":"Kiki","owner":"Park","type":"dog","gender":"MALE"}]
C:\Users\park>curl -s http://localhost:8080/pet/id/3
{"id":3,"name":"Buffy","owner":"Harold","type":"dog","gender":"FEMALE"}
C:\Users\park>curl -s http://localhost:8080/pet/name/Kiki
{"id":11,"name":"Kiki","owner":"Park","type":"dog","gender":"MALE"}
C:\Users\park>curl -s http://localhost:8080/pet/owner/Gwen
[{"id":2,"name":"Claws","owner":"Gwen","type":"cat","gender":"MALE"},{"id":6,"na
me":"Chirpy","owner":"Gwen","type":"bird","gender":"FEMALE"},{"id":7,"name":"
Whistler","owner":"Gwen","type":"bird","gender":"FEMALE"}]
```

cURL

- cURL(client url)
  - 프로토콜을 이용해 URL로 데이터를 전송하여 HTTP 요청, REST API 테스트 (GET, POST, PUT, DELETE, …)
  - 이밖에 HTTP / HTTPS / FTP / LDAP / SCP / TELNET / SMTP / POP3 등 다양하고 주요한 프로토콜을 지원
  - cURL 명령어 Option
    - -X (GET,POST,PUT,DELETE,…) 요청시 http method 설정
    - -H "Content-Type: application/json" 헤더 옵션 설정
    - -d HTTP POST 요청 데이터 "내용 (body)" 설정
    - -s 진행 내용이나 메시지들을 출력하지 않음 HTTP response만 가져올때 유리
    - -I 헤더만 가져오기
    - -i 헤더와 바디까지 가져오기
    - -v 상세 모드를 활성화하고 요청/응답 진행상황을 자세히 출력

cURL

- cURL(client url) POST
  - -d 옵션으로 body 파라미터를 앞에 쓰고, 그 뒤에 POST를 처리하는 주소를 넣음
  - 파라미터는 무조건 먼저 인코딩된 상태여야 함
  - POST의 기본 Content-Type 은 application/x-www-form-urlencoded
  - url 형식 데이터

    ```
    $ curl -d "key1=value1&key2=value2" ₩ -H "Content-Type: application/x-www-form-urlencoded" ₩ -X POST http://localhost:8000/data
    ```

  - Json 형식 데이터

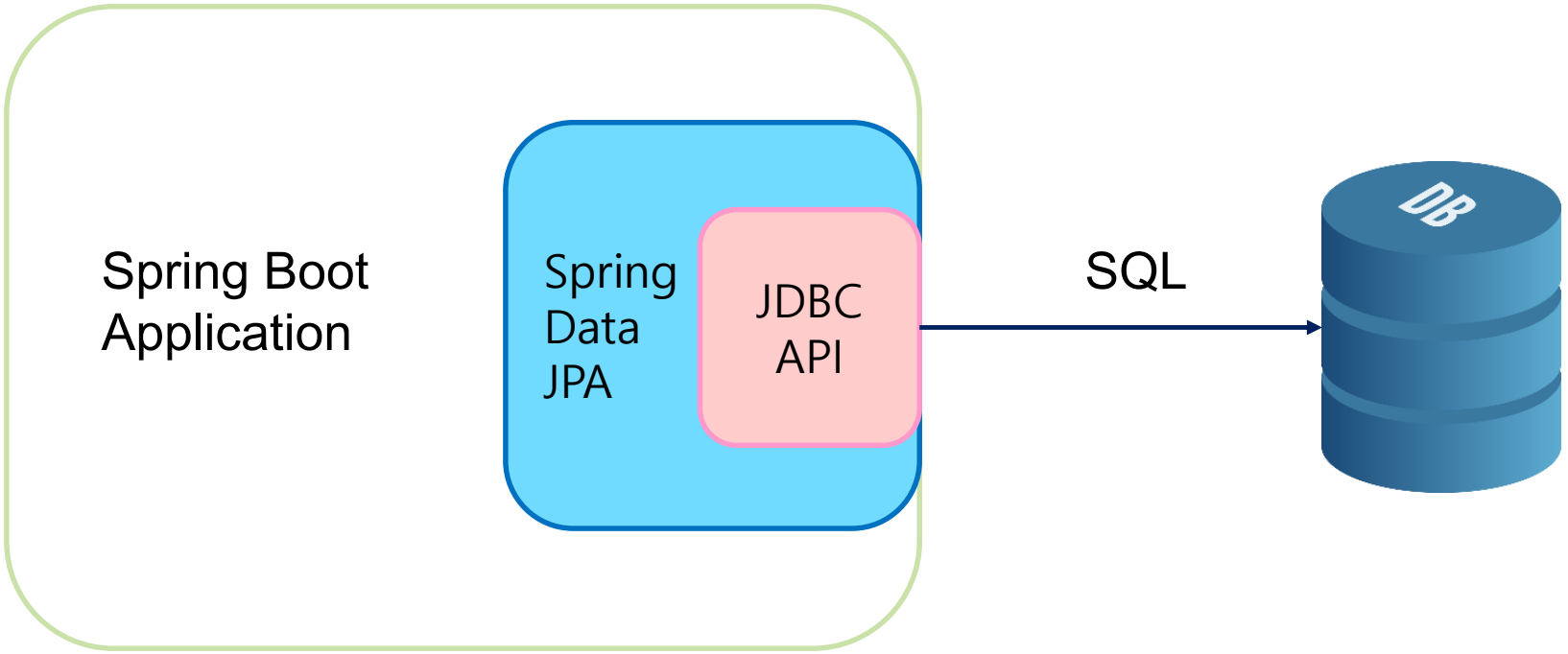
    ```
    $ curl -d '{"key1":"value1", "key2":"value2"}' ₩-H "Content-Type: application/json" ₩-X POST http://localhost:8000/data
    ```

실행 결과

```
C:\Users\park>curl -d "name=Baby&owner=Park&type=cat&gender=MALE" -H
"Content-Type: application/x-www-form-urlencoded" -X POST
http://localhost:8080/pet/add
{"id":null,"name":"Baby","owner":"Park","type":"cat","gender":"MALE"}
C:\Users\park>curl -d
"id=12&name=Baby&owner=Lim&type=dog&gender=MALE" -H "Content-Type:
application/x-www-form-urlencoded" -X POST http://localhost:8080/pet/update
{"id":12,"name":"Baby","owner":"Lim","type":"dog","gender":"MALE"}
C:\Users\park>curl -X POST http://localhost:8080/pet/delete/12

C:\Users\park>curl -s http://localhost:8080/pet/list
[{"id":1,"name":"Fluffy","owner":"Harold","type":"cat","gender":"FEMALE"},{"id":2,"
name":"Claws","owner":"Gwen","type":"cat","gender":"MALE"},{"id":3,"name":"Buf
fy","owner":"Harold","type":"dog","gender":"FEMALE"},{"id":4,"name":"Fang","ow
ner":"Benny","type":"dog","gender":"MALE"},{"id":5,"name":"Bowser","owner":"Di
ane","type":"dog","gender":"MALE"},{"id":6,"name":"Chirpy","owner":"Gwen","typ
e":"bird","gender":"FEMALE"},{"id":7,"name":"Whistler","owner":"Gwen","type":"bi
rd","gender":"FEMALE"},{"id":8,"name":"Slim","owner":"Benny","type":"snake","g
ender":"MALE"},{"id":9,"name":"Kong","owner":"Ranhee","type":"turtle","gender":"
FEMALE"},{"id":10,"name":"Puffball","owner":"Diane","type":"hamster","gender":"
FEMALE"},{"id":11,"name":"Kiki","owner":"Park","type":"dog","gender":"MALE"}]
```

JPA는 JDBC를 통해 DB에 접근



Spring JPA

- pom.xml에 JPA, JDBC와 MySQL dependency 추가

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
</dependency>
```

Entity

```
@Entity
@Getter
@NoArgsConstructor // for JPA only
@AllArgsConstructor
@Table
public class Pet {
    @Id // Primary Key
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="ID")
    private Long id;
    @Column(length=45, nullable = false)
    private String name;
    @Column(length=45, nullable = false)
    private String owner;
    @Column(length=45, nullable = false)
    private String type;
    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Gender gender;
}
```

JpaRepository

```
public interface PetRepository extends JpaRepository<Pet, Long> {
    //@Query("SELECT p FROM Pet p WHERE p.name = ?1")
    Optional<Pet> findByName(String name);
    //@Query("SELECT p FROM Pet p WHERE p.owner = ?1")
    List<Pet> findByOwner(String owner);
    //@Query("SELECT p FROM Pet p WHERE p.type = ?1 and p.gender = ?2")
    List<Pet> findByTypeAndGender(String type, Gender gender);
}
```

Service

```
@RequiredArgsConstructor
@Service
public class PetService {
    private final PetRepository repository; // repository 주입
   public Pet insert(Pet pet) {
      return repository.save(pet);
   }
   public Pet update(Pet pet) {
      return repository.save(pet);
   }
   public void deleteById(Long id) {
      repository.deleteById(id);
   }
    public List<Person> findAll() {
        return repository.findAll();
    } …
}
```

Controller

```
@RequiredArgsConstructor
@RestController
@RequestMapping("/pet")
public class PetController {
    private final PetService service; // service 주입
    @GetMapping("/list")
    public List<Pet> getAll() {
        return service.findAll();
    }
    @GetMapping("/name/{name}")
    public Pet getByName(@PathVariable("name") String name) {
        return service.findByName(name).orElse(null);
    }
    @GetMapping("/id/{id}")
    public Pet getById(@PathVariable("id") long id) {
        return service.findById(id).orElse(null);
    } …
```