

멘토링 학습 정리 - 1월 4주차

박시준

목차

- REST API와 HTTP 메서드 개념 그리고 멍등성
- 로그인에 세션 적용
- 암호에 해시 적용
- Java에서 DB 연결하는 방식
- RDBMS vs NoSQL
- DB 연결, 이메일과 암호 저장
- 힙 자료구조

REST API, 멍등성, PathVariable

REST API

- REST(Representational State Transfer)
- 로이 필딩이 2000년 박사학위논문에 최초로 개재.
- 웹의 장점을 최대한 활용하는 아키텍처
- HTTP URI를 통해 자원(Resource)을 명시하고, HTTP Method(POST, GET, PUT, DELETE)를 통해 해당 자원에 대한 CRUD Operation을 적용하는 것을 의미한다.
- 즉, REST는 ROA(Resource Oriented Architecture) 설계의 중심에 Resource가 있고 HTTP Method를 통해 Resource를 처리하도록 설계된 아키텍처를 의미한다.
- 웹 사이트의 이미지, 텍스트, DB 내용 등의 모든 자원에 고유한 ID인 HTTP URI를 부여한다.

REST 구성요소

1) 자원(Resource): URI

- 모든 자원에 고유한 ID가 존재하고, 이 자원은 Server에 존재한다.
- 자원을 구별하는 ID는 '/groups/:group_id'와 같은 HTTP URI 다.
- Client는 URI를 이용해서 자원을 지정하고 해당 자원의 상태(정보)에 대한 조작을 Server에 요청한다.

2) 행위(Verb): HTTP Method

- HTTP 프로토콜의 Method를 사용한다.
- HTTP 프로토콜은 GET, POST, PUT, DELETE 와 같은 메서드를 제공한다.

REST 구성요소

3) 표현(Representation of Resource)

- Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server는 이에 적절한 응답(Representation)을 보낸다.
- REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 Representation으로 나타내어 질 수 있다.
- JSON 혹은 XML를 통해 데이터를 주고 받는 것이 일반적이다

HTTP Method

- GET: 서버로 부터 데이터를 취득
- POST: 서버에 데이터를 추가, 작성 등
- PUT: 서버의 데이터를 갱신, 작성 등
- DELETE: 서버의 데이터를 삭제

멱등성(idempotency)과 HTTP Method

- 여러 번 수행해도 결과가 같음을 의미.
- 1) GET: idempotent 해야 한다. 여러 번 요청해도 같은 값. 캐시 사용
- 2) POST: idempotent 하지 않다. 같은 POST를 계속 보내도 같은 결과물이 나오는 것을 보장하지는 않음. 캐시 없음. Body와 Content-Type
- 3) PUT: idempotent하다. 여러 번 호출해도 같은 결과. Body와 Content-Type 필요
- 4) DELETE: idempotent하다. 여러 번 호출했을 때, 404 응답이 나올 수는 있으나, 이것이 서버의 상태에 영향을 주는 건 아님. 서버에서 해당 데이터가 지워지는 상태는 일관되게 유지가 된다.

PUT VS PATCH

- PUT은 Resource의 전체를 교체. 일부 데이터만 PUT 방식으로 보내면 보내지 않은 데이터는 Null로 처리되어 에러가 난다.
- PATCH는 Resource의 일부를 교체. 일부 데이터만 보내도 알아서 해당 데이터만 수정됨.
- PUT은 idempotent하지만, PATCH는 idempotent 하지 않다.
- PATCH는 기본적으로 idempotent하지 않는 메서드였으나, 현재 사용은 idempotent하게 하고 있다.

PathVariable vs QueryParameter

- 만약 어떤 resource를 식별하고 싶으면 Path Variable을 사용하고,
- 정렬이나 필터링을 한다면 Query Parameter를 사용한다.
(idempotent하게 해야할 때 사용)
- 또는 동일한 URL에서 HTTP 메서드만 바꿀 때에도 PathVariable사용.

```
/users [GET] # 사용자 목록을 가져온다.  
/users [POST] # 새로운 사용자를 생성한다.  
/users/123 [PUT] # 사용자를 갱신한다.  
/users/123 [DELETE] # 사용자를 삭제한다.
```

테스트 프로젝트

서비스 개요 및 기능

- 영상을 편집하는 어플리케이션
- 홈 화면에서 작업 중인 영상 목록을 볼 수 있음
- 영상 목록에서 영상을 누르면 편집 화면으로 갈 수 있음
- 사용자는 로그인할 수 있음
- 로그인 한 사용자는 멤버십에 가입할 수 있음
- 멤버십에 가입한 사용자는 광고 없이 이용할 수 있음

비즈니스 요구사항 정리

- 데이터

- 1) 고객 (id, 이름, email, password, 멤버십 유무)
- 2) 영상 (id, 등록날짜)

- 기능

- 1) 로그인
- 2) 파일 등록
- 3) 파일 목록 조회
- 4) 파일 편집

로그인에 세션 적용

쿠키(Cookie)

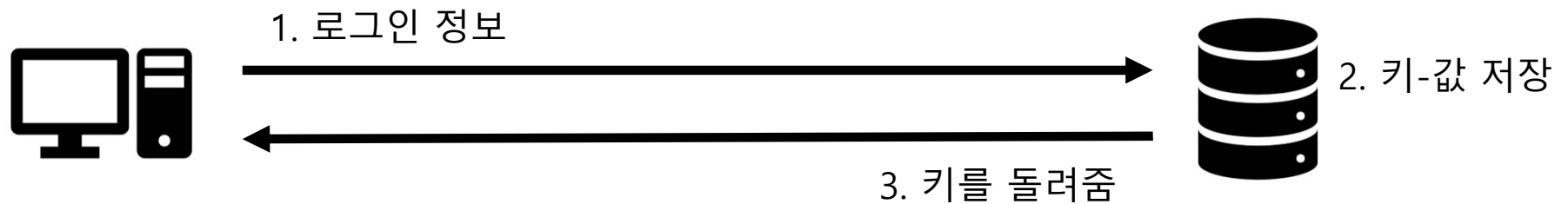
- HTTP는 Stateless 하므로 사용자를 기억 못 함.
- 기본적으로 로그인한 사용자를 기억하기 위한 수단으로 함께 사용함.
- 쿠키는 클라이언트 측에 키-값 쌍이 저장된다.
- 쿠키는 서버에서 생성되고, 클라이언트가 이것을 받고 나면 만료기간 까지 계속 들고 있다.
- 이후 HTTP 헤더에 쿠키 정보가 포함되어 서버 측으로 전송되면, 서버 측에서 사용자를 식별할 수 있음.

쿠키의 치명적인 문제

- 보안에 약하다.
- 쿠키는 클라이언트 측에서 들고 있으므로 이것을 임의로 변경해서 요청하면 다른 사용자인척 속여서 접속할 수 있음.
- 혹여 쿠키에 중요한 정보(결제 정보)가 담긴다면, 해킹 당했을 경우 막대한 피해가 발생됨.

세션

- 쿠키를 사용하는, 쿠키의 단점을 극복하기 위한 대안.
- 세션은 클라이언트가 아닌 서버 측에 데이터(키-값)가 저장된다.
- 클라이언트에서 로그인하면 그 정보(값)을 서버에 키-값쌍으로 저장하고 사용자에게는 랜덤으로 생성한 key를 쿠키에 담아서 돌려준다. 사용자는 이 쿠키에 담긴 key를 사용해서 본인을 식별하도록 하게 한다.
- 서버는 각 클라이언트에게 고유의 session ID를 부과한다.
- 브라우저가 종료되면 세션이 삭제되기 때문에 보안이 우수함.



세션으로 로그인 기능 구현

- 서블릿에 세션 만드는 편의 기능 다 구현되어있음
- 서블릿의 request 객체를 사용해서 세션 생성함.

```
HttpServletRequest request,
```

```
HttpSession session = request.getSession();
```

- 서블릿 세션의 고유 클래스명은 HttpSession
- 세션 메서드로 setAttribute()는 세션 쓰기, getAttribute()는 세션 읽기

```
session.setAttribute("name", value); // 세션에 값을 세팅하는 방법
```

```
session.getAttribute("name") // 세션에서 값을 가져오는 방법
```

세션으로 로그인 기능 구현

- setAttribute()
- "loginMember"라는 키에 loginMember 객체를 넣어서 사용
- POSTMAN에서 JSESSIONID 포착.

```
@PostMapping("/login")
public void login(
    @RequestBody LoginFrame data,
    HttpServletRequest request,
    HttpServletResponse response
) throws IOException {
    log.info("email={}, password={}", data.getEmail(), data.getPassword());

    Member loginMember = memberService.login(data.getEmail(), data.getPassword());

    if(loginMember == null){
        response.sendRedirect(location: "/login");
    }

    HttpSession session = request.getSession();
    session.setAttribute(name: "loginMember", loginMember);

    response.sendRedirect(location: "/");
}
```

Body Cookies (1) Headers (5) Test Results

Name	Value	Domain
JSESSIONID	16D5C3AB0F...	localhost

Body Cookies (1) Headers (5) Test Results

Pretty	Raw	Preview	Visualize
1	welcome		

```
: name=season, email=
: email=sionwer5@gmail
: login success
```

세션으로 로그아웃 기능 구현

- session.invalidate()를 사용하면, 서버에서 세션을 제거한다.
- 클라이언트가 들고 있는 세션은 무용지물이 됨.
- 더 이상 로그인 상태에서 보이던 정보 안 보임.

```
@PostMapping("/logout")
public void logout(HttpServletRequest request, HttpServletResponse response) {
    HttpSession session = request.getSession(create: false);
    session.invalidate();

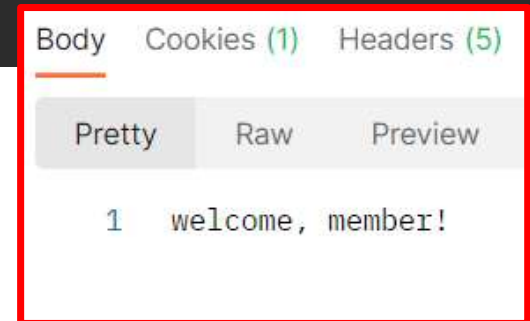
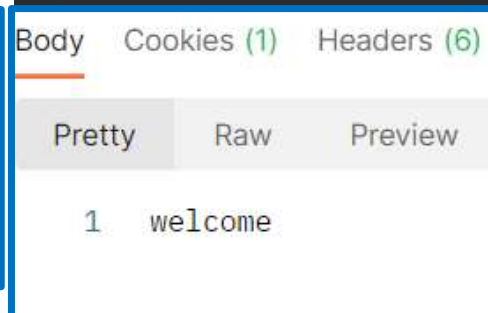
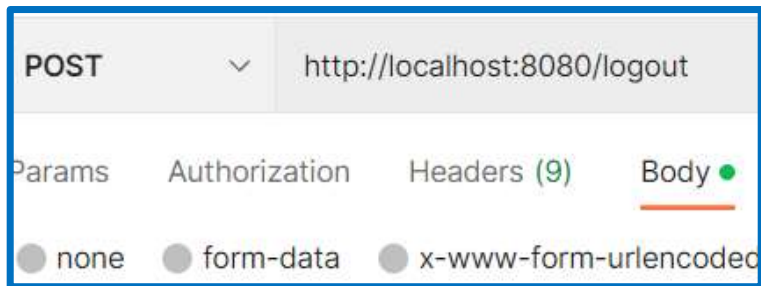
    log.info("logout success");
    response.sendRedirect(location: "/");
}
```

```
: login success
: logout success
```

세션을 통해 서버에서 사용자를 인지하는 법

- Session.getAttribute("key")
- Cookie도 생성
- 로그인에 성공하면 홈으로 바로 redirect됨

```
@GetMapping("/")  
public String home(HttpServletRequest request){  
    HttpSession session = request.getSession();  
    if(session == null){  
        log.info("no session");  
        return "welcome";  
    }  
    Member loginMember = (Member)session.getAttribute("loginMember");  
    if(loginMember == null){  
        log.info("no user info");  
        return "welcome";  
    }  
  
    return "welcome, member!";  
}
```



해시 - 패스워드 암호화

암호화를 위한 셋팅 - 스프링 자체 시큐리티

- 스프링은 자체적으로 security 패키지를 제공한다.
- Gradle에 spring-boot-starter-security 추가

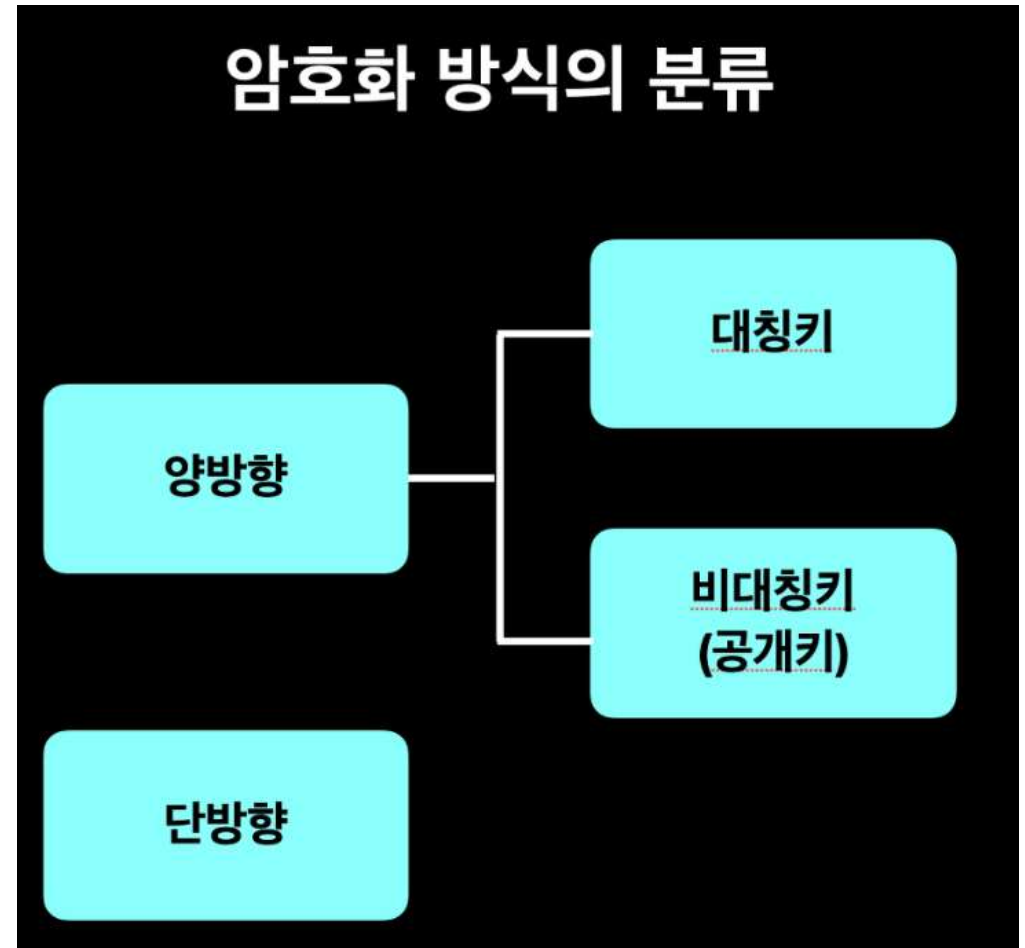
```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    compileOnly 'org.projectlombok:lombok'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

암호화 - PasswordEncoder

- PasswordEncoder는 Spring Security에서 비밀번호를 단방향 암호화하기 위한 인터페이스다.
- 구현 클래스로 BCryptPasswordEncoder가 존재하는데 BCrypt 알고리즘을 사용하며, spring에서 권장하는 클래스다.

암호화 – Bcrypt

- Bcrypt는 암호화 알고리즘의 한 종류.
- 암호화 알고리즘은 복호화가 가능한지 여부에 따라 양방향, 단방향으로 나뉘며, Bcrypt는 복호화가 불가능하기 때문에 단방향 알고리즘이다.
- 반복 횟수를 설정할 수 있어서 처리되는 시간을 임의로 늘릴 수 있다.
- Salt를 사용하여, 동일한 평문도 매번 다른 해시값으로 암호화될 수 있다.



원하지 않는 결과 - Spring Security 기본 view

- 만들지도 않은 로그인 양식의 뷰가 렌더링 되었다.
- 심지어 home -> login으로 자동 접속.
- 이것을 막기 위해서는 Configuration에서 Manual하게 disable 시켜야 한다.

localhost:8080/login

Problems - LeetCode 프로그래머

Please sign in

Username

Password

Sign in

PasswordEncoder

- @SpringBootApplication에서 SecurityAutoConfiguration 클래스를 exclude하기
- 배제하지 않으면 모든 HTTP 요청에 대해 인증/인가를 사용한다.
- 단순히 암호화 기능만을 사용할 것이므로 현재는 인증/인가 기능 끄기

```
@SpringBootApplication(exclude = SecurityAutoConfiguration.class)
public class RestapiApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestapiApplication.class, args);
    }
}
```

PasswordEncoder - encode()

- Bean으로 등록한 PasswordEncoder의 의존관계를 자동주입
- passwordEncoder.encode()를 하면 괄호 내부 데이터는 해싱된다.

```
@Autowired
public MemberService(MemberRepository memberRepository, PasswordEncoder passwordEncoder) {
    this.memberRepository = memberRepository;
    this.passwordEncoder = passwordEncoder;
}

public String join(JoinForm joinData){
    Member member = new Member();
    memberRepository.findByEmail(joinData.getName())
        .ifPresent(m -> {throw new IllegalArgumentException("이미 존재하는 회원입니다.");});
    member.setName(joinData.getName());
    member.setEmail(joinData.getEmail());
    member.setPassword(passwordEncoder.encode(joinData.getPassword()));
    memberRepository.make(member);
    return member.getPassword();
}
```

PasswordEncoder - encode()

- PasswordEncoder로 암호화할 때마다 다른 값으로 해싱됨.
- Spring이 제공하는 Bcrypt 암호화는 salt를 제공하기 때문.



```
: $2a$10$H5LwB6XTHx5PcszGNcF.T0LMTMA8zkHY3bTY9n5btCbqEzny/xRy
```

```
: name=season, email=stornwet3@gmail.com, password=120120, password  
: $2a$10$uL6LrR9sJA9totsPQHE6n0V0sY6Qdw7P0dB9Ne2X9tG5/hvxKIKMW
```

```
: $2a$10$5Ei6JNsXbEauX0Gdnimx9efndQntEoMLfJI35Y1jBcriQonrIVH66
```

Salt

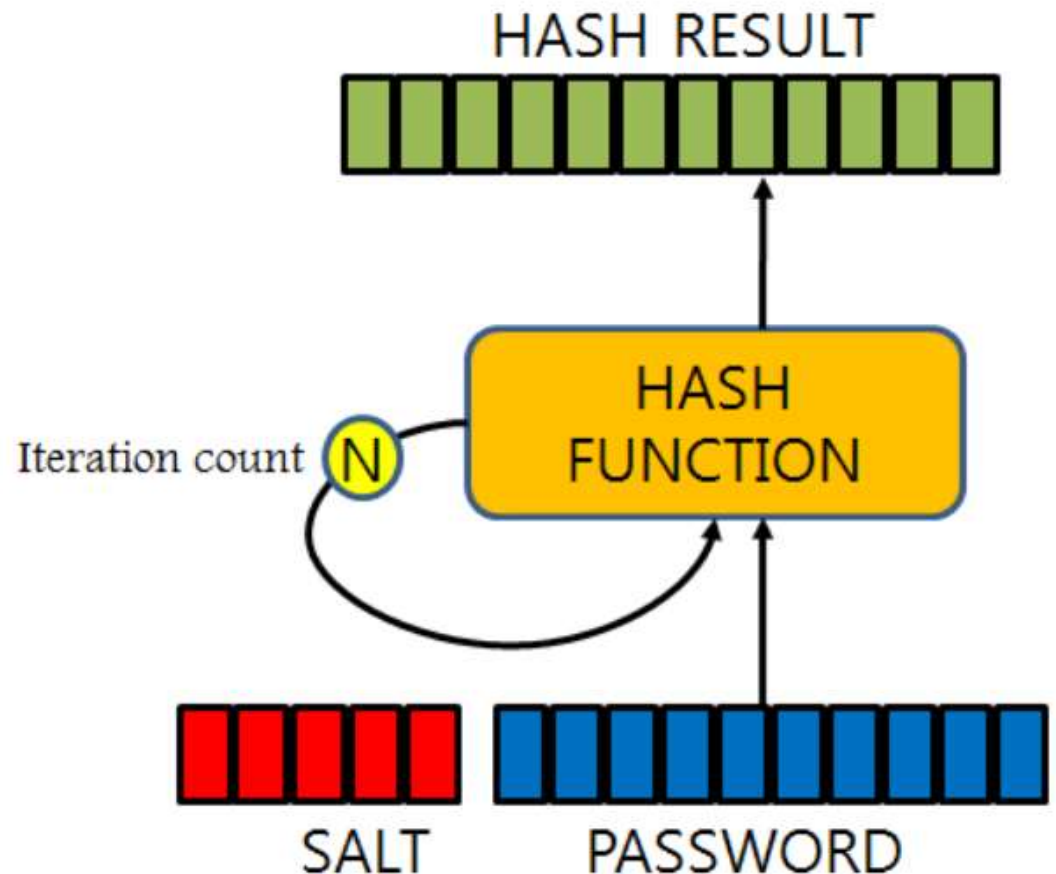
- Spring이 제공하는 Bcrypt 암호화는 salt를 제공
- 솔트(salt)는 '데이터, 비밀번호, 통과암호를 해시 처리하는 단방향 함수의 추가 입력으로 사용되는 랜덤 데이터'
- 즉, 원본 문자열에 추가적인 랜덤 생성되는 문자열을 추가한 후 해싱한다.

8zff4fgflgfd93fgdl4fgdggf4mlf45p1  redfl0wer  DIGEST

SALT PASSWORD Hash function

BCrypt 알고리즘

- Salt와 Password를 해싱하여 다이제스트를 만듦
- 이 다이제스트를 입력값으로 하여 N번 돌림
- N번 돌리는 이유는 Brute-force 공격에 대한 대비책



Join 기능 Refactoring

- 이메일 기존 존재 여부와
패스워드 일치 여부를 모두
서비스 영역에서 판단하여
Boolean으로 돌려준다.

```
public Boolean join(JoinForm joinData){
    Member member = new Member();
    memberRepository.findByEmail(joinData.getName())
        .ifPresent(m -> {throw new IllegalArgumentException("이미 존재하는 호
member.setName(joinData.getName());
member.setEmail(joinData.getEmail());
if(!joinData.getPassword().equals(joinData.getPasswordConfirm())){
    log.info("password confirm doesn't match");
    return false;
}
member.setPassword(passwordEncoder.encode(joinData.getPassword()));
log.info("password hashed={}", member.getPassword());
memberRepository.make(member);
return true;
}
```

```
@PostMapping("/join")
public JoinForm join(@RequestBody JoinForm data){
    log.info("name={}, email={}, password={}, password confirm={}",
        data.getName(), data.getEmail(), data.getPassword(), data.getPasswordConfirm());

    boolean result = memberService.join(data);
    if (!result){
        return null;
    }
    return data;
}
```


PasswordEncoder - match()

- PasswordEncoder의 match() 메서드를 사용하여 입력값과 저장값 일치 여부 검사
- 첫번째 인자는 평문, 두번째 인자는 저장된 해시값

```
public Member login(String email, String password){
    Optional<Member> user = memberRepository.findByEmail(email);
    if(user.isEmpty()) {
        log.info("such email address doesn't exist");
        return null;
    }
    Member member = user.get();
    boolean result = passwordEncoder.matches(password, member.getPassword());
    log.info("result={}", result);
    if(!passwordEncoder.matches(password, member.getPassword())){
        log.info("password wrong. password={}", password);
        return null;
    }
    return member;
}
```

PasswordEncoder - login 컨트롤러

- Service에서는 match()를 이용하여 패스워드 일치 구현
- 컨트롤러는 그저 서비스를 호출하기만 하므로
- 로그인 객체가 없으면 로그인 창으로 돌려보냄.

```
Member loginMember = memberService.login(data.getEmail(), data.getPassword());

if(loginMember == null){
    log.info("login fail");
    response.sendRedirect( location: "/login");
    return;
}

log.info("login success");
HttpSession session = request.getSession();
session.setAttribute( name: "loginMember", loginMember);
response.sendRedirect( location: "/");
```

Configuration - WebSecurityConfigurerAdapter

- Security의 기능을 어디까지 사용할 것인지 Configuration 설정을 해줘야 한다.
- 예전에는 WebSecurityConfigurerAdapter 인터페이스를 상속하여 클래스를 @Configuration으로 지정했지만, 현재는 spring-security 5.7이상에서는 Deprecated

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().disable()
            .csrf().disable()
    }
}
```

Spring Security without the WebSecurityConfigurerAdapter

ENGINEERING | ELEFThERIA STEIN-KOUSATHANA | FEBRUARY 21, 2022 99 COMMENTS

In Spring Security 5.7.0-M2 we [deprecated](#) the `WebSecurityConfigurerAdapter`, as we encourage users to move towards a component-based security configuration.

Configuration - SecurityFilterChain

- SecurityFilterChain을 Bean
으로 등록하면 된다.

DB 연결 방식 종류

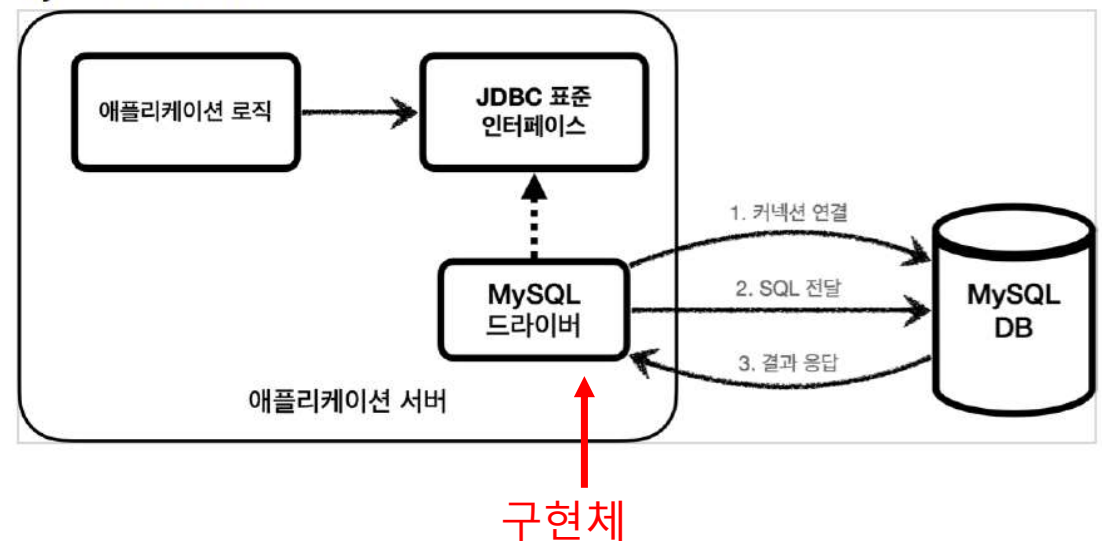
JDBC

- JDBC: 데이터베이스에 연결하게 해주는 자바 진영의 표준 API
- 등장 이유: 데이터베이스마다 사용법이 다르다. 커넥션을 연결하는 법, SQL을 전달하는 법, 결과를 응답받는 법.

- 표준 인터페이스

- 1) java.sql.Connection
- 2) java.sql.Statement
- 3) java.sql.ResultSet

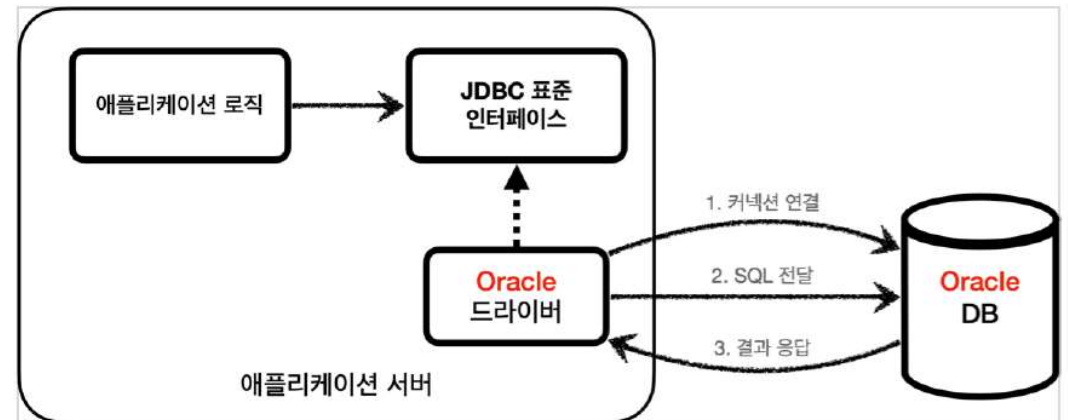
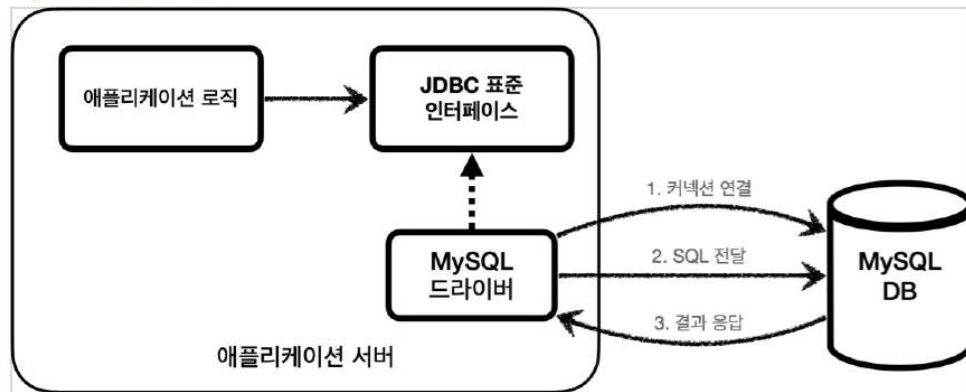
MySQL 드라이버 사용



JDBC 장점

- 데이터베이스 종류가 바뀌면 드라이버(구현체)만 바꿔주면 된다.

MySQL 드라이버 사용



JDBC의 문제점

- 데이터베이스마다 SQL 사용법이 다른 경우가 있다.
- JDBC 코드는 변경하지 않아도 되지만, SQL은 해당 데이터베이스에 맞게 변경해야 한다.
- 너무 low level 처리해야 한다.
- 계속 똑같은 코드 반복해야 한다.

JDBC의 문제점

- 무엇보다도 너무 극형 코드 반복

```
public Member save(Member member) throws SQLException {
    String sql = "insert into member(member_id, money) values (?, ?)";

    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;

    try {
        con = getConnection();
        pstmt = con.prepareStatement(sql);
        pstmt.setString( parameterIndex: 1, member.getMemberId());
        pstmt.setInt( parameterIndex: 2, member.getMoney());
        pstmt.executeUpdate();
        return member;
    } catch (SQLException e) {
        log.error("db error", e);
        throw e;
    } finally {
        close(con, pstmt, rs, null);
    }
}
```

```
public Member findById(String memberId) throws SQLException {
    String sql = "select * from member where member_id = ?";
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;

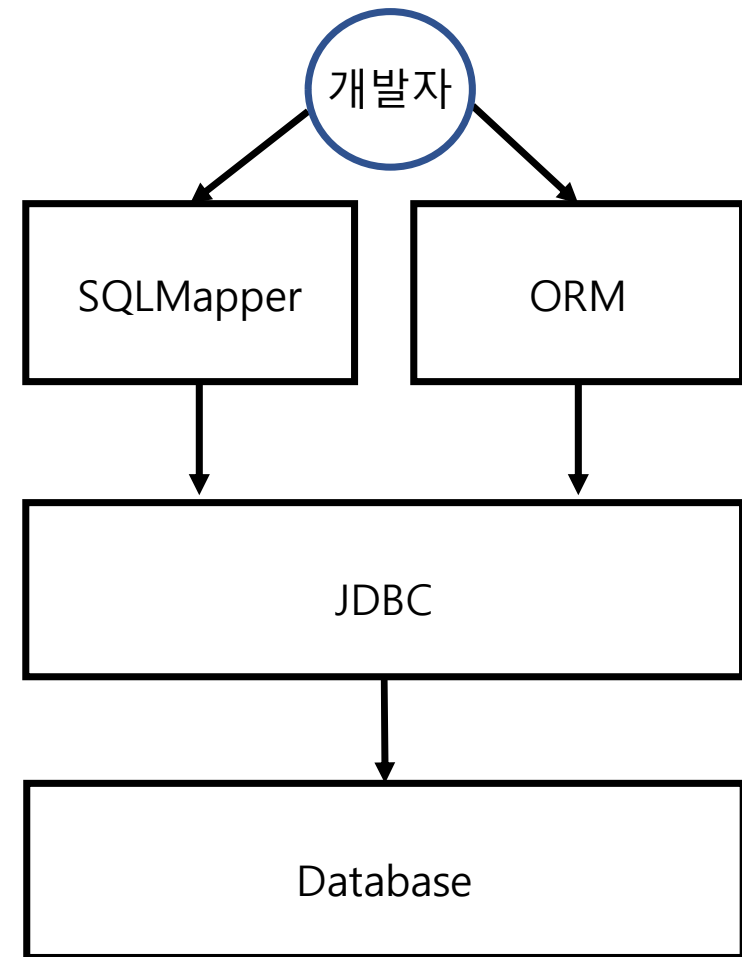
    try {
        con = getConnection();
        pstmt = con.prepareStatement(sql);
        pstmt.setString( parameterIndex: 1, memberId);
        rs = pstmt.executeQuery();
        if (rs.next()) {
            Member member = new Member();
            member.setMemberId(rs.getString( columnLabel: "member_id"));
            member.setMoney(rs.getInt( columnLabel: "money"));
            return member;
        } else {
            throw new NoSuchElementException("member not found memberId=" +
                memberId);
        }
    } catch (SQLException e) {
        log.error("db error", e);
        throw e;
    } finally {
        close(con, pstmt, rs);
    }
}
```

```
public void update(String memberId, int money) throws SQLException {
    String sql = "update member set money=? where member_id=?";
    Connection con = null;
    PreparedStatement pstmt = null;

    try {
        con = getConnection();
        pstmt = con.prepareStatement(sql);
        pstmt.setInt( parameterIndex: 1, money);
        pstmt.setString( parameterIndex: 2, memberId);
        int resultSize = pstmt.executeUpdate();
        log.info("resultSize={}", resultSize);
    } catch (SQLException e) {
        log.error("db error", e);
        throw e;
    } finally {
        close(con, pstmt, rs, null);
    }
}
```

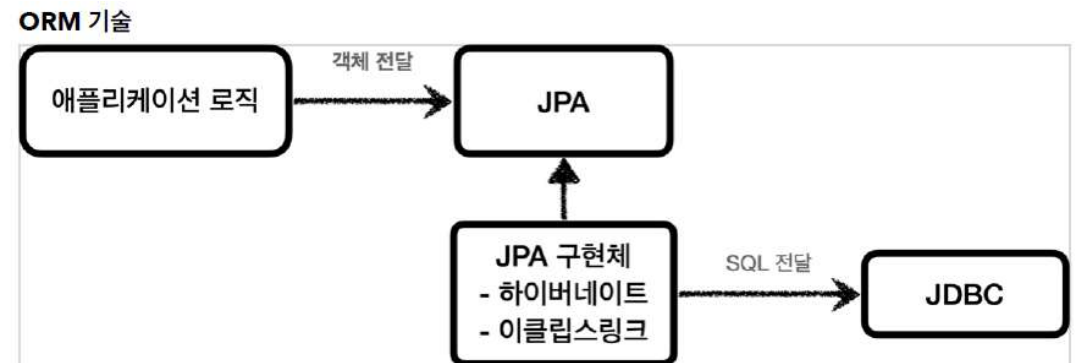
JDBC의 대안 – SQLMapper 또는 ORM

- JDBC를 직접 사용하기 보다는 SQLMapper 또는 ORM을 사용하여 JDBC 컨트롤을 대신 한다.
- SQLMapper와 ORM은 병렬 구조. 둘 중 하나를 선택해서 사용하면 된다.



SQLMapper vs ORM

- SQLMapper: SQL을 작성하면 해당 SQL의 결과를 객체로 매핑.
- ORM – JPA: 기본적인 SQL은 JPA가 대신 작성하고 처리함. 개발자는 그냥 컬렉션에 저장하고 조회하면, 나머지 ORM이 알아서 데이터 베이스에 객체를 저장하고 조회해준다. 개발자는 SQL 작성 X

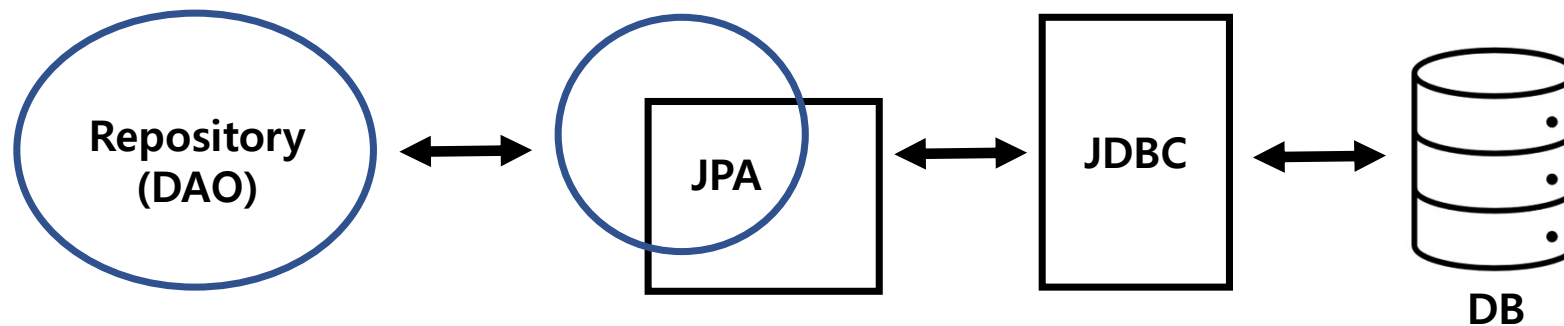


SQLMapper

- JDBC를 직접 사용할 때 발생하는 대부분의 반복작업을 대신 처리함.
- SQL을 직접 작성하는 경우에 사용한다.
- JDBC Template
 - 동적 SQL(where, and 쿼리 조건 작성)을 해결하는 건 어렵다.
- MyBatis
 - 동적 SQL 작성하기 용이하다.
 - XML에 작성한다.

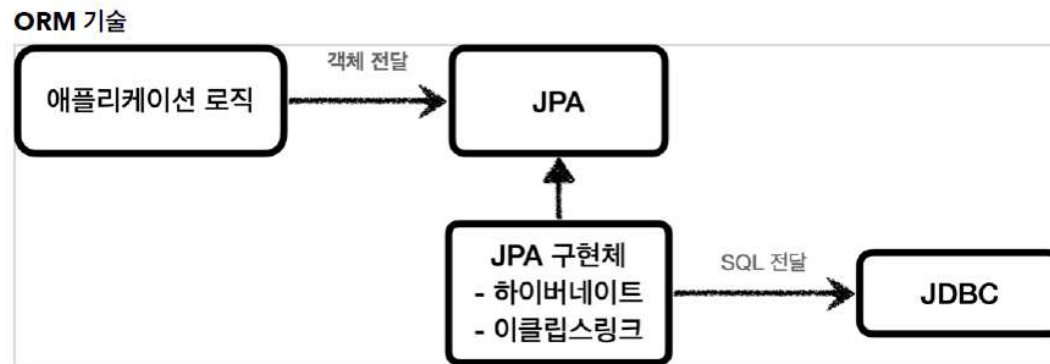
ORM(Object-Relational Mapping)

- 이런 RDBMS를 객체와 손쉽게 매핑해주는 기술이 ORM.
- ORM이 개발자 대신 SQL을 동적으로 만들어준다. (완전 편리!)
- SQL 중심적인 개발이 아닌 객체 중심 개발을 할 수 있다.



ORM - JPA? Hibernate?

- JPA는 ORM 표준 인터페이스
- Hibernate는 JPA의 구현체. 가장 많이 사용한다. 개빈 킹이 만든 오픈 소스 프로젝트.

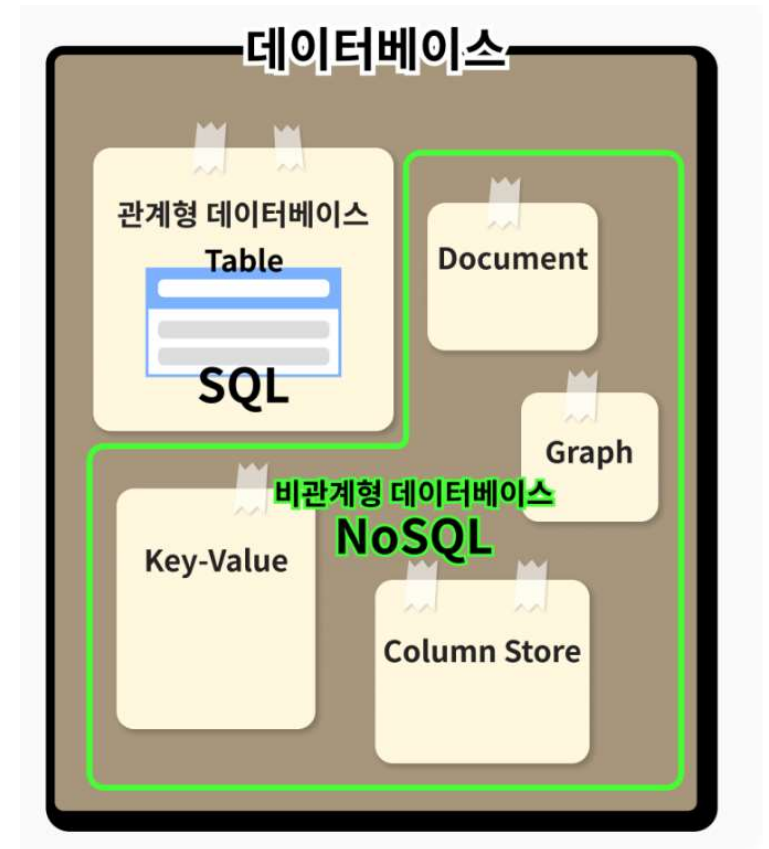


- JPA 쓰면, DB 접근 할 때 JAVA 컬렉션 접근하듯이 하면 된다.

RDBMS vs NoSQL

RDBMS vs NoSQL

- RDBMS는 행과 열로 구성된 테이블에 데이터 저장. 테이블 구조와 타입을 사전에 지정한다.
- NoSQL은 데이터가 고정되어있지 않은 데이터베이스 구조
- NoSQL에는 Key-Value 방식, Document방식, 그래프 방식 등이 있다.



RDBMS vs NoSQL

- RDBMS는 쿼리를 사용하며 고정된 형식의 스키마가 필요하다. SQL을 기반으로 수직적으로 확장한다. 하드웨어 성능을 많이 사용함.
- NoSQL는 동적으로 스키마 형태를 관리할 수 있고, 더 빨리 읽고 쓰고, 수평적으로 확장한다. 하드웨어 적게 사용해서 저렴.
- 데이터베이스의 ACID를 준수하기 위해서는 RDBMS를 사용해야 한다.
- ACID: 트랜잭션(Transaction)에 의한 상태의 변화를 수행하는 과정에서, 안전성을 보장하기 위해 필요한 성질이다.

DB에 연결, 이메일과 암호 저장

JPA – Entity

- 데이터 모델에 @Entity 붙이기만 하면 끝
- Primary Key만 @Id 붙여야 하고 나머지 키들은 자동으로 Column으로 등록됨
- JPA에선 기본생성자 필요

```
@Data
@Entity
public class Member {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;
    private String password;
    private Boolean membership;

    public Member(){

    }
}
```

Generation.IDENTITY

- 기존의 메모리 저장소에서 id 시퀀스 자동으로 올려주던 것을 구현하기 위한 편의 기능
- 이것을 엔티티의 id에 올려놓으면 id를 DB에서 생성해서 올려준다.
- DB가 알아서 Auto Increment

memoryRepository

```
private static Map<Long, Member> store = new HashMap<>();  
private static long sequence = 0L;
```

Entity

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;
```

JPA – Repository

- DAO에서 JPA 쓸려면 EntityManager를 Import해야함.
- entityManager의 persist(): 저장
- entityManager의 find(): 조회
- @Transactional 필수. JPA의 모든 변경은 트랜잭션 안에서 해결.
- 거의 memory저장소에서 객체에 저장하듯이 사용하면 된다.

```
private final EntityManager entityManager;  
  
public JpaMemberRepository(EntityManager entityManager) {  
    this.entityManager = entityManager;  
}
```

```
@Override  
public void make(Member member) {  
    entityManager.persist(member);  
}  
  
@Override  
public Optional<Member> findById(Long id) {  
    Member member = entityManager.find(Member.class, id);  
    return Optional.ofNullable(member);  
}
```

왜 클래스 변수에는 ".class"를 붙여줘야 할까?

- EntityManager의 find 메서드는 첫번째 인자의 형태로 클래스 변수를 원함

- 왜 ".class"가 굳이 붙는 거지?

```
@Override
public Optional<Member> findById(Long id) {
    Member member = entityManager.find(Member.class, id);
    return Optional.ofNullable(member);
}
```

⇒클래스 변수는 static 영역에 저장되는 변수다.

⇒사용할 수 있는 인스턴스가 없는 경우 ".class"를 붙여서 클래스 자체를 활용함

ERROR - UnsatisfiedDependencyException

- Dependency 빈 생성 관련 에러

```
ERROR 26664 --- [main] o.s.boot.SpringApplication : Application run failed

org.springframework.beans.factory.support.ConstructorResolver$UnsatisfiedDependencyException: Error creating bean with name 'dataSourceScriptDatabaseInitializer' defined in class path resource [com/example/dao/DataSourceScriptDatabaseInitializer.class]: Unsatisfied dependency found through bean 'dataSourceScriptDatabaseInitializer': org.springframework.jdbc.datasource.init.ScriptUtils cannot be instantiated; nested exception is java.lang.NoClassDefFoundError: org/apache/commons/dbcp2/BasicDataSourceFactory
    at org.springframework.beans.factory.support.ConstructorResolver.createArgumentArray(ConstructorResolver.java:798) ~[spring-beans-6.0.3.jar:6.0.3]
```

```
factory.BeanCreationException: Create breakpoint : Error creating bean with name 'dataSource' defined in class path
    at org.springframework.support.ConstructorResolver.instantiate(ConstructorResolver.java:657) ~[spring-beans-6.0.3.jar:6.0.3]
    at org.springframework.support.ConstructorResolver.instantiateUsingFactoryMethod(ConstructorResolver.java:645) ~[spring-beans-6.0.3.jar:6.0.3]
```

- @Entity와 @Repository에서 별 짓 다했으나 해결 불가
- Build의 dependency 관련 문제로 예상하고 해결

ERROR – 빌드 깨짐

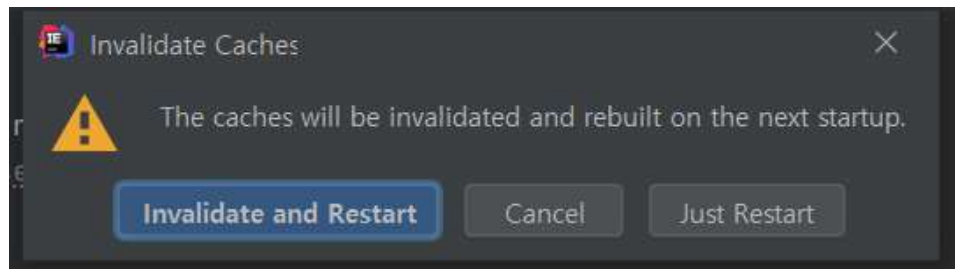
- Git으로 JPA 사용 전으로(메모리 사용으로 정상동작까지 확인된 코드) 돌렸음에도 불구하고 빌드가 깨지는 마법이 발생.

```
D:\RESTAPI_Training\back\restapi\src\main\java\training\restapi\AppConfig.java:3:46  
java: package org.springframework.context.annotation does not exist
```

- 모든 dependency는 그대로 잘 있음에도 annotation 기능을 활용하지 못한다.
- 조사결과, IntelliJ 자체 문제. Git을 revert하면서 어디선가 꼬인 것 같다.

ERROR – 빌드 깨짐

- IntelliJ에서 cache를 삭제하고 재시작하여 다시 빌드함.



- 캐시를 삭제하면 정상동작 하는 것을 보니 IntelliJ와 Gradle이 충돌하는 경우가 많은 것 같다. (평소는 괜찮은데, git revert만 하면 이러한 문제가 발생함.)

ERROR - UnsatisfiedDependencyException

- Dependency 삽입과 application.properties 까지는 문제 없음.

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    compileOnly 'org.projectlombok:lombok'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    runtimeOnly 'com.h2database:h2'  
}
```

```
spring.datasource.url=jdbc:h2:tcp://localhost/~ /test  
spring.datasource.username=sa  
logging.level.org.springframework.jdbc=debug  
logging.level.org.hibernate.SQL=DEBUG  
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE|
```

ERROR - UnsatisfiedDependencyException

- 갑자기 동작 잘 됨. 문제가 뭐였는지 파악 불가

```
2023-01-31T12:56:20.874+09:00 INFO 53888 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2023-01-31T12:56:20.882+09:00 INFO 53888 --- [main] training.restapi.RestapiApplication : Started RestapiApplication in 3.495 seconds (process running fo
2023-01-31T12:56:27.347+09:00 INFO 53888 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-01-31T12:56:27.347+09:00 INFO 53888 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-01-31T12:56:27.348+09:00 INFO 53888 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2023-01-31T12:56:27.510+09:00 INFO 53888 --- [nio-8080-exec-1] t.restapi.controller.HomeController : no user info
```

DB에 연결 후 POST 맨으로 Join

- EntityManager의 persist를 사용하여 데이터 저장을 테스트 함.
- MEMBER 테이블은 H2에서 미리 만들어 둬

POST http://localhost:8080/join

Params Authorization Headers (9) Body ● P

● none ● form-data ● x-www-form-urlencoded

```
1 {}
2   .... "name": "season",
3   .... "email": "sionwer5@gmail.com",
4   .... "password": "123123",
5   .... "passwordConfirm": "123123"
6 }
```

실행 Run Selected 자동 완성 지우기 SQL 문:

```
SELECT * FROM MEMBER
```

SELECT * FROM MEMBER;

ID	NAME	EMAIL	PASSWORD	MEMBERSHIP
----	------	-------	----------	------------

(행 없음, 0 ms)

편집

ERROR - NumberFormatException

- Int 타입을 넣어야 하는 곳에 String 타입을 넣었다고 에러 발생

```
java.lang.NumberFormatException: Create breakpoint : For input string: "sionwer5@gmail.com"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67) ~[na:na]
    at java.base/java.lang.Long.parseLong(Long.java:708) ~[na:na]
    at java.base/java.lang.Long.parseLong(Long.java:831) ~[na:na]
    at org.hibernate.type.descriptor.java.LongJavaType.lambda$coerce$0(LongJavaType.java:150) ~[hibernate-co
```

ERROR - NumberFormatException

- Log를 사용하여 추적 결과 repository 메서드 중 findByEmail에서 에러가 발생한 것

```
public Boolean join(JoinForm joinData){  
    Member member = new Member();  
    log.info("join try");  
    memberRepository.findByEmail(joinData.getEmail())  
        .ifPresent(m -> {throw new IllegalArgumentException("이미 존재하는 회원입니다.");});  
    log.info("set data");  
    member.setName(joinData.getName());  
    member.setEmail(joinData.getEmail());  
    member.setMembership(false);  
}
```

```
@Override  
public Optional<Member> findByEmail(String email) {  
    Member member = entityManager.find(Member.class, email);  
    return Optional.ofNullable(member);  
}
```

ERROR - NumberFormatException

- Entity Manager의 find의 두번째 인자는 primary key만 받는다.
- 이 primary key는 Long타입으로 지정되어있음

```
Class<Member> entityClass, Object primaryKey  
Class<Member> entityClass, Object primaryKey, Map<String, Object> properties  
Class<Member> entityClass, Object primaryKey, LockModeType lockMode  
Class<Member> entityClass, Object primaryKey, LockModeType lockMode, Map<String, Object> properties  
public Optional<Member> findById(Long primaryKey) {  
    Member member = entityManager.find();  
}
```

ERROR - NumberFormatException

- JPA의 find()에는 primary key만 들어갈 수 있고, 다른 key를 넣을 수 있는 유사한 메서드는 없다.
- 다른 key를 넣기 위해 해결책 검색한 결과, 순수 JPA 코드 해결책은 찾기 어려움.
- 대부분 **spring data JPA**를 사용 중이었음

```
public interface EmotionRepository extends JpaRepository<Emotion, Integer> {  
    Optional<Emotion> findById(String uid);  
}
```

```
@Repository  
public interface CalendarMonthRepository extends CrudRepository<Calendar_Month, Long> {  
    List<Calendar_Month> findByCalendar_Month(String calendar_Month)  
}
```


Spring data JPA

- JpaRepository를 상속해서 새로운 인터페이스를 만들어 사용한다.
- 개발자가 임의로 메서드 만들면 그 메서드 이름을 분석해서 필요한 JPQL을 만들고 실행해줌.
- 아무 단어나 쓰면 되는 건 아니고, 규칙이 있음
- [query method](#)

```
public interface MemberRepository extends JpaRepository<Member, Long> {  
    List<Member> findByUsernameAndAgeGreaterThan(String username, int age);  
}
```

Spring data JPA

- findByEmail 새로운 메서드를 인터페이스에 작성해준다.

```
public interface SpringDataJpaMemberRepository extends JpaRepository<Member, Long> {  
    Optional<Member> findByEmail(String email);  
}
```

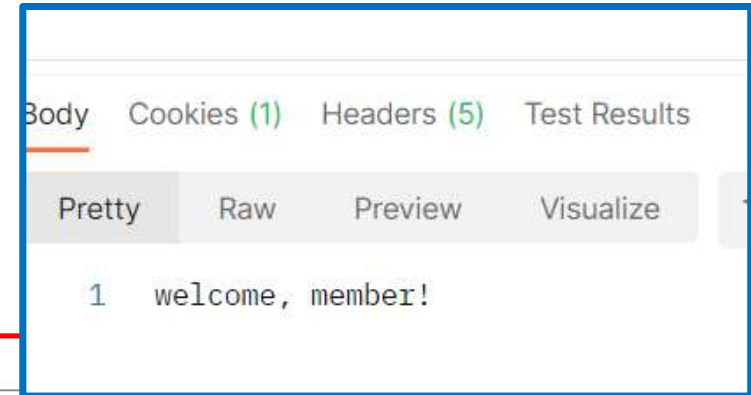
- Repository에 인터페이스를 주입하고, 만든 메서드를 그냥 쓴다.

```
private final SpringDataJpaMemberRepository repository;  
  
public SpringDataJpaRepository(SpringDataJpaMemberRepository springDataJpaMemberRepository) {  
    this.repository = springDataJpaMemberRepository;  
}
```

```
public Optional<Member> findByEmail(String email) {  
    return repository.findByEmail(email);  
}
```

Spring data JPA

- POSTMAN으로 /join에 post
- Spring Data JPA로 DB에 데이터 넣기 성공



ERROR – NumberFormatException 순수 JPA로 해결

- 나중에 찾고 보니, 순수 JPA로 이를 해결하기 위해서는 그냥 Query를 작성해야 했음. <- JPA를 쓰는 메리트가 없다.
- EntityManager의 createQuery 메서드 = 직접 쿼리를 넣을 수 있게 해준다.
(jpql 문법)

```
public List<Member> findByUsernameAndAgeGreaterThan(String username, int age) {  
    return em.createQuery("select m from Member m where m.username = :username  
and m.age > :age")  
        .setParameter("username", username)  
        .setParameter("age", age)  
        .getResultList();  
}
```

힙 자료구조

우선순위 큐

- 우선순위 큐는 먼저 들어온 데이터가 먼저 나가는 것이 아니라, 우선순위가 높은 데이터가 먼저 나가는 구조다.
- 우선순위 큐는 Heap으로 구현한다. (다른 수단으로 구현해도 되지만, 힙이 input, output에 따른 시간복잡도가 $O(\log(n))$ 으로 가장 안정적)
- Enqueue(), Dequeue(), peak() 기능이 필요하다.
- 여기서 peak()는 최대 우선순위 요소를 반환하는 것으로 Heap구조에선 루트를 반환하면 됨.

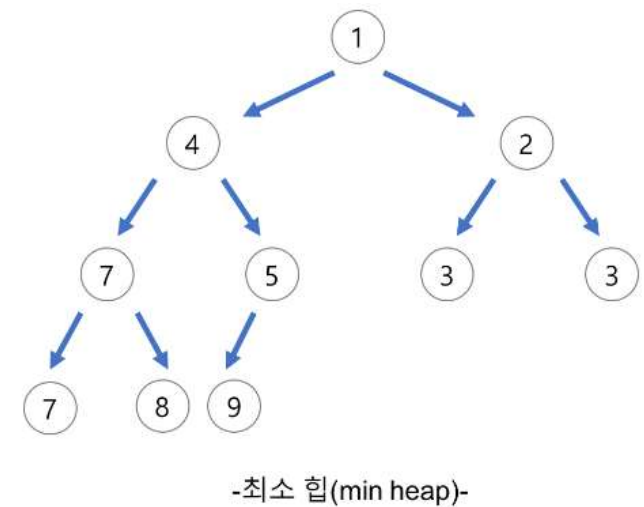
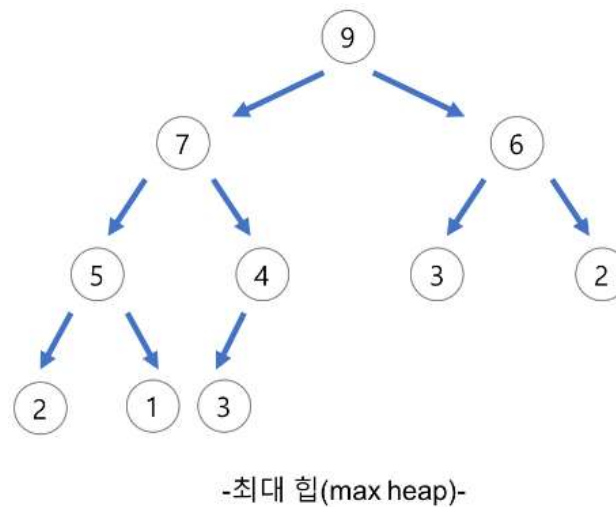
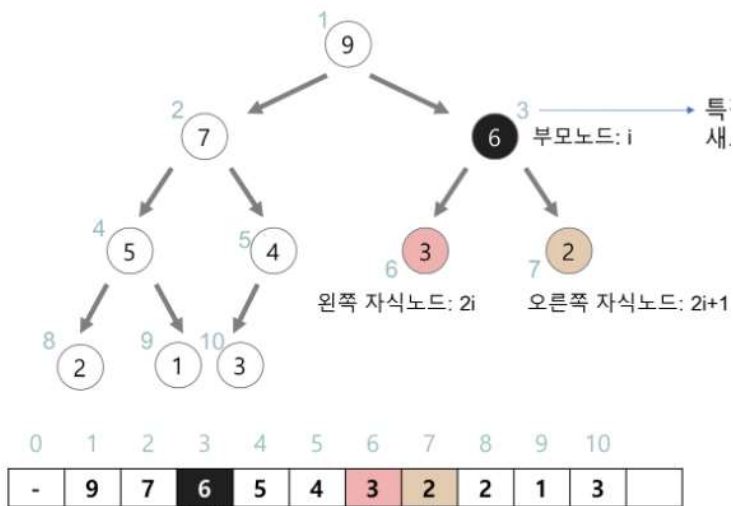
우선순위 큐 - Heap

- 우선순위 큐는 Heap으로 구현한다. Heap이 가장 안정적.

우선순위 큐를 구현하는 표현 방법	삽입	삭제
순서 없는 배열	$O(1)$	$O(n)$
순서 없는 연결 리스트	$O(1)$	$O(n)$
정렬된 배열	$O(n)$	$O(1)$
정렬된 연결 리스트	$O(n)$	$O(1)$
힙(heap)	$O(\log n)$	$O(\log n)$

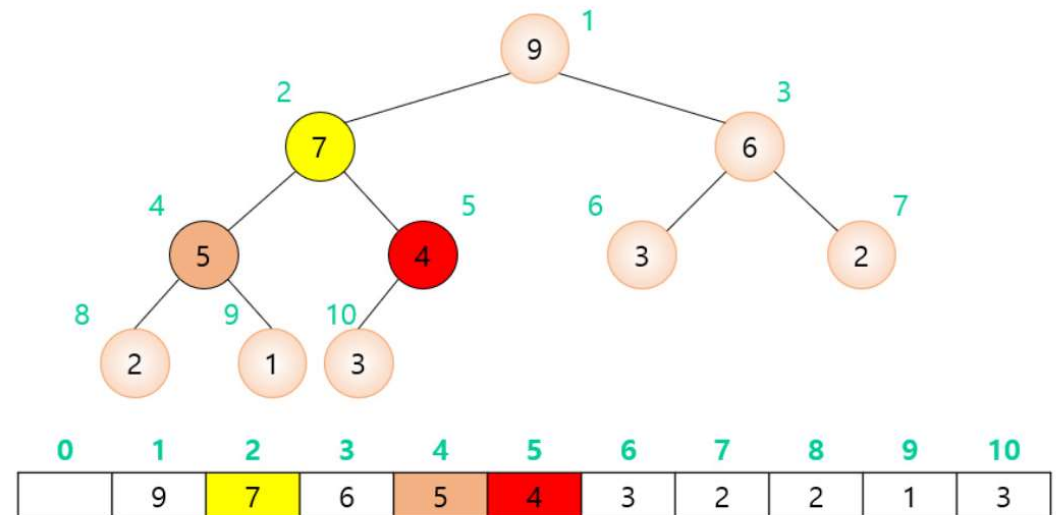
Heap

- 힙은 완전 이진 트리이므로 배열로 구현한다. (빈 노드가 없기 때문)
- 일반적으로 0번 인덱스는 사용하지 않는다.
- 상하 관계는 보장되지만, 자식들 간의 좌우관계는 정의되지 않음. 수평으로 크기 순서는 알 수 없다.



Heap – parent와 child의 관계

- Left child index = (parent index) * 2
- Right child index = (parent index) * 2 + 1
- parent index = (child index) // 2



Heap – 삽입, 삭제 연산

- 삽입은 일단 마지막 index에 넣고, 정렬한다
- 삭제는 첫번째 index(루트 노드)를 삭제하고 정렬한다.
- 맨 뒤로 넣고 맨 앞으로 꺼내는 것은 일반적인 Queue와 같음
- 하지만 우선순위에 따라 정렬하는 행위가 추가됨.
- 구현된 코드는 Maximum Heap

```
def enqueue(arr, value):
    arr.append(value) # put it in first
    i = len(arr) - 1
    while i > 1: # while it is not root
        if arr[i // 2] > arr[i]: # if parent is bigger than child
            break
        swap(arr, i // 2, i)
        i = i // 2

def dequeue(arr, value):
    if len(arr) == 1: # check if it is empty
        return arr

    root = arr[1]
    arr[1] = arr[len(arr) - 1]
    arr.pop() # pop the root

    i = 1
    while(i * 2 <= len(arr) - 1): # while it has child
        if (arr[i] > arr[i*2]) and (arr[i] > arr[i*2 + 1]):
            break
        elif arr[i*2] > arr[i*2 + 1]:
            swap(arr, i, i*2)
            i = i*2
        else:
            swap(arr, i, i*2 + 1)
            i = i*2 + 1

    return root # pop the root

def swap(arr, i, j):
    temp = arr[i]
    arr[i] = arr[j]
    arr[j] = temp
```

Heap – 삽입 enqueue()

- 삽입은 일단 마지막 index에 넣고,
부모 자식을 비교해서 정렬한다

```
def main():
    arr = [0] # Initialization essential
    enqueue(arr, 3)
    enqueue(arr, 4)
    enqueue(arr, 9)
    enqueue(arr, 5)
    enqueue(arr, 13)
    enqueue(arr, 11)
    enqueue(arr, 6)
    print(arr)

if __name__ == "__main__":
    main()
```

```
def enqueue(arr, value):
    arr.append(value) # put it in first
    i = len(arr) - 1
    while i > 1: # while it is not root
        if arr[i // 2] > arr[i]: # if parent is bigger than child
            break
        swap(arr, i // 2, i)
        i = i // 2
```

```
def swap(arr, i, j):
    temp = arr[i]
    arr[i] = arr[j]
    arr[j] = temp
```

실행 결과

```
===== RESTART: D:/Algorithm/method/priority_queue_heap.py
[0, 13, 9, 11, 3, 5, 4, 6]
>>> |
```

Heap – 삭제 dequeue()

- 일단 루트 노드는 빼고, 마지막 노드를 루트에 올린 후 정렬 알고리즘

```
def main():
    arr = [0] # Initialization essential
    enqueue(arr, 3)
    enqueue(arr, 4)
    enqueue(arr, 9)
    enqueue(arr, 5)
    enqueue(arr, 13)
    enqueue(arr, 11)
    enqueue(arr, 6)
    print(arr)
    dequeue(arr)
    print(arr)
    dequeue(arr)
    print(arr)
```

```
===== RESTART: D:/Algorithm/method/priority_queue_heap.py
[0, 13, 9, 11, 3, 5, 4, 6]
[0, 11, 9, 6, 3, 5, 4]
[0, 9, 5, 6, 3, 4]
```

```
def dequeue(arr):
    if len(arr) == 1: # check if it is empty
        return arr

    root = arr[1]
    arr[1] = arr[len(arr) - 1]
    arr.pop() # pop the root

    i = 1
    while(i * 2 <= len(arr) - 1): # while it has at least one child
        # one child case
        if len(arr) - 1 < (i * 2 + 1):
            if arr[i] > arr[i*2]:
                break
            swap(arr, i, i*2)
            i = i*2
            continue

        #two child case
        if (arr[i] > arr[i*2]) and (arr[i] > arr[i*2 + 1]):
            break
        elif arr[i*2] > arr[i*2 + 1]:
            swap(arr, i, i*2)
            i = i*2
        else:
            swap(arr, i, i*2 + 1)
            i = i*2 + 1

    return root # pop the root
```

다음 학습: 필터와 인터셉터,
파일 업로드, Security 인증/인가