Spring MVC2 11 [로그인 - 세션 사용]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.24

로그인 처리하기 - 세션 사용

쿠키의 여러 보안 이슈가 발생하는 것을 방지하기 위해 중요한 정보를 서버에 저장해야 함.

쿠키로 임의 생성된 토큰을 전송하고 랜덤 토큰을 세션 아이디로 갖는 값을 가지고 있는다.

회원과 관련된 정보는 클라이언트에 전달하지 않고 추정 불가능한 세션 **ID**만 전송**.**

로그인시 - 토큰을 생성하고 저장 - 토큰을 가지고 있다 → 홈에서 사용시 토큰으로 확인.

토큰 - 변조하기 힘들고

- 쿠키가 털렸을 시 다른 정보가 같이 노출되지 않음
- 변조가 불가능
- 털려도 유지시간이 짧음

직접 개발해보기

-생성, - 조회, -응답 쿠키 생성해 전달.

```
@Component
public class SessionManager {

   public static final String SESSION_COOKIE_NAME = "mySessionID";
   private Map<String, Object> sessionStore = new ConcurrentHashMap<>();

   /**
    * 세션 생성
    */
   public void createSession(Object value, HttpServletResponse response){

       //세션 id를 생성하고, 값을 세션에 저장
       String sessionId = UUID.randomUUID().toString();
```

```
        sessionStore.put(sessionId, value);

        //쿠키 생성
        Cookie mySessionCookie = new Cookie(SESSION_COOKIE_NAME, sessionId);
        response.addCookie(mySessionCookie);
    }

    /**
     * 세션 조회
     */
    public Object getSession(HttpServletRequest request){
        Cookie sessionCookie = findCookie(request, SESSION_COOKIE_NAME);
        if(sessionCookie == null){
            return null;
        }
        return sessionStore.get(sessionCookie.getValue());
    }
    /**
     * 세션 만료
     */
    public void expire(HttpServletRequest request){
        Cookie sessionCookie = findCookie(request, SESSION_COOKIE_NAME);
        if(sessionCookie != null){
            sessionStore.remove(sessionCookie.getValue());
        }
    }

    /**
     * 해당 쿠키 찾기
     */
    public Cookie findCookie(HttpServletRequest request, String cookeName){
        if(request.getCookies() == null){
            return null;
        }
        return Arrays.stream(request.getCookies())
                .filter(cookie -> cookie.getName().equals(cookeName))
                .findFirst().orElse(null);
    }
}
```

테스트 해보기

```
public class SessionManagerTest {

    SessionManager sessionManager = new SessionManager();
```

```
    @Test
    void sessionTest(){

        //세션 생성 → 로그인 시
        MockHttpServletResponse response = new MockHttpServletResponse();
        Member member = new Member();
        sessionManager.createSession(member, response);

        //요청에 응답 쿠키 저장 ==> 웹브라우저의 요청 ( 홈 화면 이동 )
        MockHttpServletRequest request = new MockHttpServletRequest();
        request.setCookies(response.getCookies());

        //세션 조회
        Object result = sessionManager.getSession(request);
        Assertions.assertThat(result).isEqualTo(member);

        //세션 만료
        sessionManager.expire(request);
        Object expired = sessionManager.getSession(request);
        Assertions.assertThat(expired).isNull();
    }
}
```

직접 만든 세션 사용하기

```
@PostMapping("/login")
public String loginV2(@Valid @ModelAttribute LoginForm form, BindingResult
bindingResult, HttpServletResponse response){
    if(bindingResult.hasErrors()){
        return "login/loginForm";
    }
    Member loginMember = loginService.login(form.getLoginId(), form.getPassword());
    if(loginMember == null){
        bindingResult.reject("loginFail", "아이디 또는 비밀번호가 맞지 않습니다.");
        return "login/loginForm";
    }
    //로그인 성공 처리
    log.info("login success");
    //세션 관리자를 통해 세션을 생성하고, 회원 데이터 보관
    sessionManager.createSession(loginMember, response);
    return "redirect:/";
}
```

```
@PostMapping("/logout")
public String logoutV2(HttpServletRequest request){
   sessionManager.expire(request);
   return "redirect:/";
}
```

→ 로그인 로그아웃 로직.

```
@GetMapping("/")
public String homeLoginV2(HttpServletRequest request, Model model){

   //세션 관리자에 저장된 회원 정보 조회
   Member member = (Member)sessionManager.getSession(request);

   //로그인
   if(member==null){
       return "home";
   }

   model.addAttribute("member", member);
   return "loginHome";
}
```

→ 홈 매핑 로직

서블릿이 세션을 지원한다.

서블릿도 세션 개념을 지원 한다. 세션이라는 것은 쿠키를 사용하는데 데이터를 서버에서 관리하는 것.

서블릿 **HTTP** 세션 기본

서블릿을 통해 세션을 생성하면 랜덤 쿠키 생성.

Login Mapping

```
//로그인 생성
//세션이 있으면 세션 반환, 없으면 신규 세션을 생성
HttpSession session = request.getSession();
//getSession 옵션 : true 세션이 없으면 생성 후 반환
//           옵션 : false 세션이 없으면 생성하지 않고 null 반환
```

```
//로그인 한 세션 정보 보관
session.setAttribute(SessionConst.LOGIN_MEMBER, loginMember);
```

"/login" 매핑에 로그인 완료되었을 때 세션 생성 정보를 넣어준다.

HttpServletRequest 에 속해있는 getSession으로 세션을 생성 or 불러오고

setAttribute를 통해 세션값을 집어넣을 수 있다.

getSession 옵션 : true 세션이 없으면 생성 후 반환
: false 세션이 없으면 생성하지 않고 null 반환

Logout Mapping

```
@PostMapping("/logout")
public String logoutV3(HttpServletRequest request){
   HttpSession session = request.getSession(false);
   if(session != null){
       session.invalidate();
   }
   return "redirect:/";
}
```

Home mapping 세션 불러오기.

```
@GetMapping("/")
public String homeLoginV3(HttpServletRequest request, Model model){

   HttpSession session = request.getSession(false);

   if(session == null){
       return "home";
   }

   //세션에서 회원 데이터 가져오기
   Member loginMember = (Member) session.getAttribute(SessionConst.LOGIN_MEMBER);

   //세션에 회원 데이터가 없으면 home
   if(loginMember==null){
       return "home";
   }

   model.addAttribute("member", loginMember);
   return "loginHome";
}
```

서블릿 **HTTP** 세션 **-** 스프링

스프링에서 지원하는 세션 사용해보기 - 애노테이션 사용

@ SessionAttribute를 사용

```
@GetMapping("/")
public String homeLoginV3Spring(
       @SessionAttribute(name = SessionConst.LOGIN_MEMBER, required = false) Member
loginMember
       , Model model){

   //세션에 회원 데이터가 없으면 home
   if(loginMember==null){
       return "home";
   }

   model.addAttribute("member", loginMember);
   return "loginHome";
}
```

⇒ 이 기능은 세션을 생성하지는 않는다.

TrackingMode ?

처음 로그인 하게 되면 뒤에 생성된 쿠키가 **URL**에 표출된다**.**

```
http://localhost:9595/;jsessionid=1F998C2C599BB16FA500015E9C230520
```

웹 브라우저가 쿠키를 지원하지 않는다면 URL로 세션을 유지하는 방법이다. 하지만 모든 URL에 계속 포함해서 전달해야 하기 때문에 현실적으로 힘들다.

그렇다면 왜 나오는지**?**

최초에는 쿠키를 지원하는지 판단하지 못하므로 쿠키값도 전달하고 , jsessionid도 함께 전달

없애는 법 : application.properties 에 옵션을 설정해 준다.

```
server.servlet.session.tracking-modes=cookie
```

세션 정보 & 타임아웃

세션 정보 확인

```
@GetMapping("/session-info")
public String sessionInfo(HttpServletRequest request){
   HttpSession session = request.getSession(false);
   if(session == null){
       return "session 이 없습니다.";
   }

   session.getAttributeNames().asIterator()
           .forEachRemaining(name -> log.info("session name={}, value={}", name,
session.getAttribute(name)));

   log.info("sessionId={}", session.getId());
   log.info("sessionMaxInactiveInterval={}", session.getMaxInactiveInterval());
   log.info("creationTime={}", new Date(session.getCreationTime()));
   //마지막 접근 시간
   log.info("lastAccessedTime={}", new Date(session.getLastAccessedTime()));
   //생성된 것인지 여부
   log.info("isNew={}", session.isNew());

   return "세션 출력";
}
```

세션 타임아웃

Session의 종료 시점**?**

사용자가 로그아웃 했을때 invalidate를 사용해 세션을 만료시킬 수 있지만

사용자는 주로 로그아웃을 선택하지 않고 웹 브라우저를 종료한다. HTTP 는 비연결성이므로 브라우저가 종료한 것인지 알기 힘들다. 그러므로 세션을 언제 삭제해야 하는지 알기 힘들다.

세션은 메모리에 생성이 되고 메모리 크기가 무한하지 않기 때문에 꼭 필요한 경우에만 생성하고 불필요하면 삭제해야 한다. 또한 악의 적인 사용을 계속 오랜 시간 유지 시킬 이유는 없다.

종료 시점?

생성 시점에서 30분? → 30분 이후에 계속 로그인등을 사용해 세션을 계속 갱신해야한다. (불 편)

최근 요청한 시점부터 30분 → 사용할 때마다 30분 (HttpSession 의 생명주기)

```
server.servlet.session.timeout=60  [ 초 단위 ]
```

application.properties 에 timeout 설정으로 들어가고

아니면 setMaxInactiveInterval 메소드로 세부 설정도 가능하다.

세션은 최소한으로 사용해야 한다! 저장한 데이터 용량 * 사용자 수로 세션의 메모리 사용량이 급격히 늘 수 있고 장애로 이어질 수 있다.