2장 Ajax 댓글 처리

2.1 RestController와 Ajax

- REST는 'Representational State Transfer'의 약어로 하나의 URI는 하나의 고유한 리소스를 대표하도록 설계된다는 개념이다.

- REST 방식은 특정한 URI는 반드시 그에 상응하는 데이터 자체라는 것을 의미한다.

[꿀팁] REST, REST API, RESTful의 의미

1. REST

- "Representational State Transfer" 의 약자로 자원을 이름(자원의 표현)으로 구분하여 해당 자원의 상태(정보)를 주고 받는 모든 것을 의미한다. 즉, 자원(resource)의 표현(representation)에 의한 상태전달

- 자원(resource)의 표현(representation)
  - 자원: 해당 소프트웨어가 관리하는 모든 것. 예) 문서, 그림, 데이터, 해당 소프트웨어 자체 등
  - 표현: 그 자원을 표현하기 위한 이름. 예) DB의 학생 정보가 자원일 때, 'students'를 자원의 표현으로 정한다.

- 상태(정보) 전달
  - 데이터가 요청되어지는 시점에서 자원의 상태(정보)를 전달한다. JSON 혹은 XML를 통해 데이터를 주고 받는 것이 일반적이다.

2. REST API

- API(Application Programming Interface)
  - 데이터와 기능의 집합을 제공하여 컴퓨터 프로그램간 상호작용을 촉진하며, 서로 정보를 교환가능 하도록 하는 것

- REST API의 정의
  - REST 기반으로 서비스 API를 구현한 것
  - 최근 OpenAPI(누구나 사용할 수 있도록 공개된 API: 구글 맵, 공공 데이터 등), 마이크로 서비스(하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처) 등을 제공하는 업체 대부분은 REST API를 제공한다.

3. RESTful

- 일반적으로 REST라는 아키텍처를 구현하는 웹 서비스를 나타내기 위해 사용되는 용어이다.

- 'REST API'를 제공하는 웹 서비스를 'RESTful'하다고 할 수 있다. REST를 REST답게 쓰기 위한 방법, 즉 REST 원리를 따르는 시스템은 RESTful이란 용어로 지칭된다.

2.1.1 @RestController의 소개

- 스프링 4버전부터 지원되는 '@RestController' 어노테이션의 경우 기존의 특정한 JSP와 같은 뷰를 만들어 내는 것이 목적이 아닌 REST 방식의 데이터 처리를 위해서 사용하는 어노테이션이다.

- 스프링 3버전까지 컨트롤러는 주로 '@Controller' 어노테이션망을 사용해서 처리하였고, 화면 처리를 담당하는 JSP가 아닌 데이터 자체를 서비스하려면 해당 메소드나 리턴 타입에 '@ResponseBody' 어노테이션을 추가하는 형태로 작성하였다.

2.1.2 예제 프로젝트의 생성

- 예제 프로젝트는 'ex02' 이름으로 생성한다.

2.1.3 테스트용 컨트롤러 생성하기

- 작성된 SampleController에는 @RestController와 @RequestMapping 어노테이션을 추가한다.

[ex02/src/main/java/org/zerock/controller/SampleController.java]

```
01    package org.zerock.controller;
02
03    import java.util.ArrayList;
04    import java.util.HashMap;
05    import java.util.List;
06    import java.util.Map;
07
08    import org.springframework.http.HttpStatus;
09    import org.springframework.http.ResponseEntity;
10    import org.springframework.web.bind.annotation.RequestMapping;
11    import org.springframework.web.bind.annotation.RestController;
12    import org.zerock.domain.SampleVO;
13
14    @RestController
15    @RequestMapping("/sample")
16    public class SampleController {
17
18      @RequestMapping("/hello")
19      public String sayHello() {
20        return "Hello World ";
21      }
22
23      @RequestMapping("/sendVO")
```

```
24      public SampleVO sendVO() {
25
26        SampleVO vo = new SampleVO();
27        vo.setFirstName("길동");
28        vo.setLastName("홍");
29        vo.setMno(123);
30
31        return vo;
32      }
33
34      @RequestMapping("/sendList")
35      public List<SampleVO> sendList() {
36
37        List<SampleVO> list = new ArrayList<>();
38        for (int i = 0; i < 10; i++) {
39          SampleVO vo = new SampleVO();
40          vo.setFirstName("길동");
41          vo.setLastName("홍");
42          vo.setMno(i);
43          list.add(vo);
44        }
45        return list;
46      }
47
48      @RequestMapping("/sendMap")
49      public Map<Integer, SampleVO> sendMap() {
50
51        Map<Integer, SampleVO> map = new HashMap<>();
52
53        for (int i = 0; i < 10; i++) {
54          SampleVO vo = new SampleVO();
55          vo.setFirstName("길동");
56          vo.setLastName("홍");
57          vo.setMno(i);
58          map.put(i, vo);
59        }
60        return map;
61      }
62
63      @RequestMapping("/sendErrorAuth")
64      public ResponseEntity<Void> sendListAuth() {
65
66        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
67      }
68
69      @RequestMapping("/sendErrorNot")
70      public ResponseEntity<List<SampleVO>> sendListNot() {
71
72        List<SampleVO> list = new ArrayList<>();
73        for (int i = 0; i < 10; i++) {
74          SampleVO vo = new SampleVO();
75          vo.setFirstName("길동");
76          vo.setLastName("홍");
77          vo.setMno(i);
78          list.add(vo);
79        }
80
81        return new ResponseEntity<>(list, HttpStatus.NOT_FOUND);
82      }
83
84    }
```

(1) @RestConroller의 용도

■ JSP와 같은 뷰를 만들어 내지 않는 대신에 데이터 자체를 반환하는데, 이때 주로 사용되는 것은 단순 문자열과 JSON, XML 등으로 나누어 볼 수 있다.

(2) 단순 문자열의 경우

■ @RestController에서 문자열 데이터는 기본적으로 브라우저에는 'text/html' 타입으로 처리된다.

(3) 루트 컨텍스트로 실행하기

■ http://localhost:8080/sample/hello

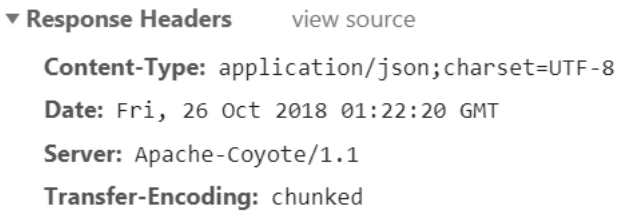
(4) 객체를 JSON으로 반환하는 경우

■ @RestController의 경우 별도의 처리가 없어도 객체는 자동으로 JSON으로 처리될 수 있다.

■ jackson-databind 라이브러리의 추가: jackson-databind 라이브러리는 객체를 JSON 타입의 데이터로 변환하거나, 반대의 작업을 할 때 사용한다. 따라서 JSON을 이용해야 하는 프로젝트에는 반드시 필요한 라이브러리이다.

```
//pom.xml의 추가
                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>2.5.4</version>
                </dependency>
```

■ 개발자 도구를 통해서 좀 더 자세한 결과를 확인해 보면 응답 헤더(Response Headers)의 내용에 응답 타입 역시 'application/json'으로 되어 있는 것을 확인할 수 있다.



(5) 컬렉션 타입의 객체를 반환하는 경우

■ 결과 데이터로 List나 Map 타입은 흔히 사용되므로, 이에 대한 확인 작업도 필요한다.
http://localhost:8080/sample/sendList

(6) ResponseEntity 타입

■ 웹의 경우는 HTTP 상태(status) 코드가 예외적인 상황 정보를 나타내는 데 사용한다. 주로 많이 사용되는 상태 코드는 아래와 같다.

- 100번대: 현재 데이터의 처리 중인 상태
- 200번대: 정상적인 응답
- 300번대: 다른 URL 처리
- 400번대: 서버에서 인식할 수 없음
- 500번대: 서버 내부의 문제

■ 스프링에서 제공하는 ResponseEntity 타입은 개발자가 직접 결과 데이터와 HTTP의 상태 코드를 직접 제어할 수 있는 클래스이다. ResponseEntity를 이용하면 개발자는 404나 500같은 HTTP 상태 코드를 전송하고 싶은 데이터와 함께 전송할 수 있기 때문에 좀더 세밀한 제어가 필요한 경우에 사용해 볼 수 있다.

```
// org.zerock.controller.SampleController.java의 일부

  @RequestMapping("/sendErrorAuth")
  public ResponseEntity<Void> sendListAuth() {

    return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
  }

  @RequestMapping("/sendErrorNot")
  public ResponseEntity<List<SampleVO>> sendListNot() {

    List<SampleVO> list = new ArrayList<>();
    for (int i = 0; i < 10; i++) {
      SampleVO vo = new SampleVO();
      vo.setFirstName("길동");
      vo.setLastName("홍");
      vo.setMno(i);
      list.add(vo);
    }

    return new ResponseEntity<>(list, HttpStatus.NOT_FOUND);
  }
```

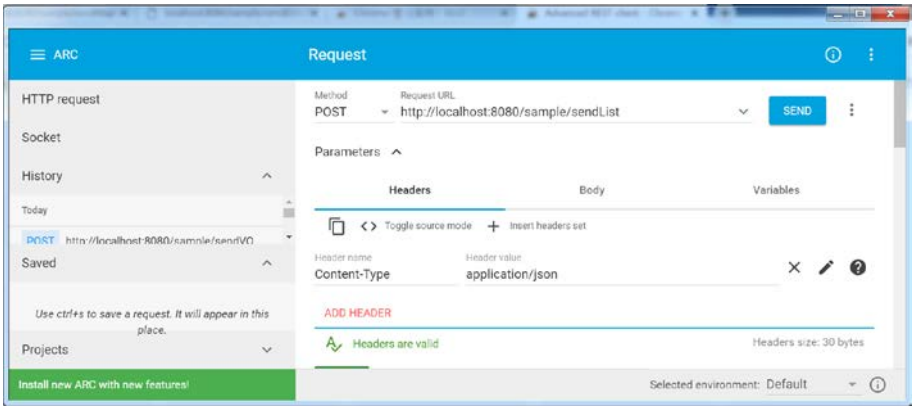
2.2 댓글 처리와 REST

■ REST 방식을 사용하는 원칙은 크게 두 가지 이다.
- URI가 원하는 리소스를 의미한다.
- URI에는 식별할 수 있는 데이터를 같이 전달하는 것이 일반적이다.

2.2.1 Advanced REST Client를 이용한 테스트

■ Chrome 웹 스토어에서 "Advanced REST Client"를 설치한다.

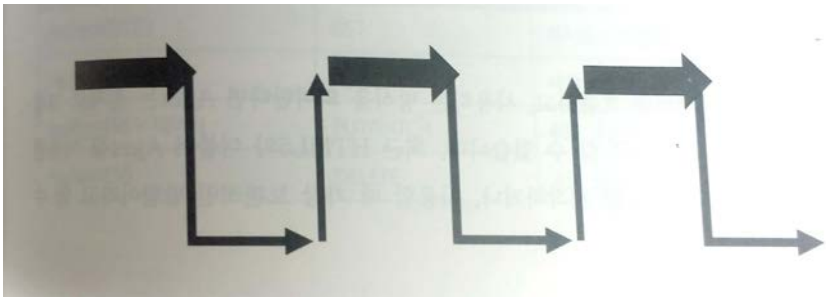
■ 실행: 구글 크럼의 주소에 "chrome://apps/"을 입력하여 "ARS"를 선택하여 실행한다.



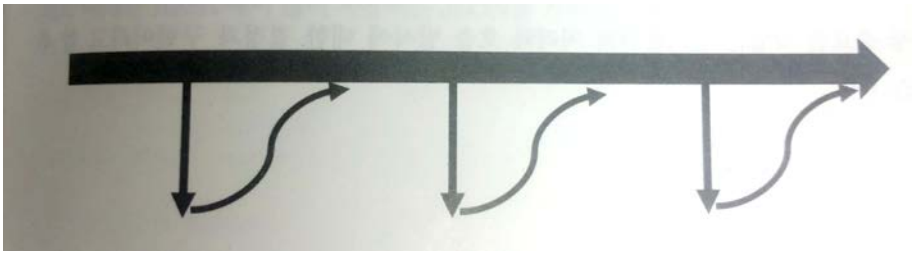
2.2.2 REST와 Ajax

■ 웹을 통해서 작업할 때 REST 방식이 가장 많이 쓰이는 형태는 Ajax와 같이 결합된 형태이다. Ajax는 'Asynchronous JavaScript and XML'의 약어로 주로 브라우저에서 대화형으로 서버와 데이터를 주고받는 형태의 메시지 전송 방식을 의미한다.

■ 동기화 방식은 순차적인 일을 실행하는 데에는 적합하다.



■ 비동기 방식의 특징은 처리된 일의 결과를 통보받은 형태로 처리된다는 점이다.



2.2.3 댓글 처리를 위한 준비

(1) 전달 방식과 처리 방식의 결정

■ 해당하는 컨트롤러를 먼저 작성하고, 그에 맞는 URI를 결정하는 것이 첫 단계이다.

■ 댓글 처리는 다음과 같은 방식으로 호출하도록 결정한다.

URI	전송방식	설명
/replies/all/123	GET	게시물 123번의 모든 댓글 리스트
/replies/ + 데이터	POST	새로운 댓글 추가
/replies/456 + 데이터	PUT/PATCH	456 댓글의 수정
/replies/456	DELETE	456 댓글의 삭제

(2) 데이터 전송 방식의 결정

■ 모바일이나 외부에서 네트워크를 이용해 원하는 데이터만 주고 받는 형태로 많이 사용하기 때문에 데이터의 전송 역시 XML이나 JSON을 사용하는 경우가 많다.

(3) 댓글을 위한 테이블 설정


```
-- 댓글을 위한 테이블 설정
create table tbl_reply (
          rno int NOT NULL AUTO_INCREMENT,
          bno int not null default 0,
          replytext varchar(1000) not null,
          replyer varchar(50) not null,
          regdate TIMESTAMP NOT NULL DEFAULT now(),
          updatedate TIMESTAMP NOT NULL DEFAULT now(),
          primary key(rno)
);

alter table tbl_reply add constraint fk_board
foreign key(bno) references tbl_board(bno);
```

(4) ReplyVO - 댓글을 위한 도메인 객체 설계

[org.zerock.domain.ReplyVO.java]

```
01    package org.zerock.domain;
02
03    import java.util.Date;
04
```

```
05  public class ReplyVO {
06
07    private Integer rno;
08    private Integer bno;
09    private String replytext;
10    private String replyer;
11
12    private Date regdate;
13    private Date updatedate;
14
15    public Integer getRno() {
16      return rno;
17    }
18
19    public void setRno(Integer rno) {
20      this.rno = rno;
21    }
22
23    public Integer getBno() {
24      return bno;
25    }
26
27    public void setBno(Integer bno) {
28      this.bno = bno;
29    }
30
31    public String getReplytext() {
32      return replytext;
33    }
34
35    public void setReplytext(String replytext) {
36      this.replytext = replytext;
37    }
38
39    public String getReplyer() {
40      return replyer;
41    }
42
43    public void setReplyer(String replyer) {
44      this.replyer = replyer;
45    }
46
47    public Date getRegdate() {
48      return regdate;
49    }
50
51    public void setRegdate(Date regdate) {
52      this.regdate = regdate;
53    }
54
55    public Date getUpdatedate() {
56      return updatedate;
57    }
58
59    public void setUpdatedate(Date updatedate) {
60      this.updatedate = updatedate;
61    }
62
63    @Override
64    public String toString() {
65      return "ReplyVO [rno=" + rno + ", bno=" + bno + ", replytext=" + replytext + ", replyer=" + replyer
66  + ", regdate="
67          + regdate + ", updatedate=" + updatedate + "]";
68    }
69  }
```

(5) ReplyDAO

■ ReplyDAO 인터페이스의 작성

[org.zerock.persistence.ReplyDAO.java]

```
01   package org.zerock.persistence;
02
03   import java.util.List;
04
05   import org.zerock.domain.Criteria;
06   import org.zerock.domain.ReplyVO;
07
08   public interface ReplyDAO {
09
10     public List<ReplyVO> list(Integer bno) throws Exception;
11
12     public void create(ReplyVO vo) throws Exception;
13
14     public void update(ReplyVO vo) throws Exception;
15
16     public void delete(Integer rno) throws Exception;
17
18     public List<ReplyVO> listPage(
19         Integer bno, Criteria cri) throws Exception;
20
21     public int count(Integer bno) throws Exception;
22
23   }
```

■ XML Mapper의 작성

[resources/mappers/replyMapper.xml]

```
01   <?xml version="1.0" encoding="UTF-8"?>
02   <!DOCTYPE mapper
03   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
04   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
05
06   <mapper namespace="org.zerock.mapper.ReplyMapper">
07
08            <select id="list" resultType="ReplyVO">
09                     select
10                     *
11                     from
12                     tbl_reply
13                     where bno =
14                     #{bno}
15                     order by rno desc
16            </select>
17
18            <insert id="create">
19                     insert into tbl_reply (bno, replytext, replyer)
20                     values (#{bno},#{replytext},#{replyer})
21            </insert>
22
23            <update id="update">
24                     update tbl_reply set replytext = #{replytext},
25                     updatedate = now()
```

```
26                      where rno = #{rno}
27            </update>
28
29            <delete id="delete">
30                      delete from tbl_reply where rno =#{rno}
31            </delete>
32
33
34            <select id="listPage" resultType="ReplyVO">
35                      select
36                      *
37                      from
38                      tbl_reply
39                      where
40                      bno = #{bno}
41                      order by rno desc
42                      limit #{cri.pageStart}, #{cri.perPageNum}
43            </select>
44
45            <select id="count" resultType="int">
46                      select count(bno) from tbl_reply where bno =#{bno}
47            </select>
48
49
50    </mapper>
```

■ ReplyDAOImpl의 작성

[org.zerock.persistence.ReplyDAOImpl.java]

```
01    package org.zerock.persistence;
02
03    import java.util.HashMap;
04    import java.util.List;
05    import java.util.Map;
06
07    import javax.inject.Inject;
08
09    import org.apache.ibatis.session.SqlSession;
10    import org.springframework.stereotype.Repository;
11    import org.zerock.domain.Criteria;
12    import org.zerock.domain.ReplyVO;
13
14    @Repository
15    public class ReplyDAOImpl implements ReplyDAO {
16
17      @Inject
18      private SqlSession session;
19
20      private static String namespace = "org.zerock.mapper.ReplyMapper";
21
22      @Override
23      public List<ReplyVO> list(Integer bno) throws Exception {
24
25        return session.selectList(namespace + ".list", bno);
26      }
27
28      @Override
29      public void create(ReplyVO vo) throws Exception {
30
31        session.insert(namespace + ".create", vo);
32      }
33
34      @Override
```

```
35      public void update(ReplyVO vo) throws Exception {
36
37        session.update(namespace + ".update", vo);
38      }
39
40      @Override
41      public void delete(Integer rno) throws Exception {
42
43        session.update(namespace + ".delete", rno);
44      }
45
46      @Override
47      public List<ReplyVO> listPage(Integer bno, Criteria cri)
48          throws Exception {
49
50        Map<String, Object> paramMap = new HashMap<>();
51
52        paramMap.put("bno", bno);
53        paramMap.put("cri", cri);
54
55        return session.selectList(namespace + ".listPage", paramMap);
56      }
57
58      @Override
59      public int count(Integer bno) throws Exception {
60
61        return session.selectOne(namespace + ".count", bno);
62      }
63    }
```

(6) ReplyService / ReplyServiceImpl 작성

■ ReplyService의 작성

[org.zerock.service.ReplyService.java]

```
01    package org.zerock.service;
02
03    import java.util.List;
04
05    import org.zerock.domain.Criteria;
06    import org.zerock.domain.ReplyVO;
07
08    public interface ReplyService {
09
10      public void addReply(ReplyVO vo) throws Exception;
11
12      public List<ReplyVO> listReply(Integer bno) throws Exception;
13
14      public void modifyReply(ReplyVO vo) throws Exception;
15
16      public void removeReply(Integer rno) throws Exception;
17
18      public List<ReplyVO> listReplyPage(Integer bno, Criteria cri)
19          throws Exception;
20
21      public int count(Integer bno) throws Exception;
22    }
```

■ ReplyServiceImpl의 작성

[org.zerock.service.ReplyServiceImpl.java]

```
01  package org.zerock.service;
02
03  import java.util.List;
04
05  import javax.inject.Inject;
06
07  import org.springframework.stereotype.Service;
08  import org.zerock.domain.Criteria;
09  import org.zerock.domain.ReplyVO;
10  import org.zerock.persistence.ReplyDAO;
11
12  @Service
13  public class ReplyServiceImpl implements ReplyService {
14
15    @Inject
16    private ReplyDAO dao;
17
18    @Override
19    public void addReply(ReplyVO vo) throws Exception {
20
21      dao.create(vo);
22    }
23
24    @Override
25    public List<ReplyVO> listReply(Integer bno) throws Exception {
26
27      return dao.list(bno);
28    }
29
30    @Override
31    public void modifyReply(ReplyVO vo) throws Exception {
32
33      dao.update(vo);
34    }
35
36    @Override
37    public void removeReply(Integer rno) throws Exception {
38
39      dao.delete(rno);
40    }
41
42    @Override
43    public List<ReplyVO> listReplyPage(Integer bno, Criteria cri)
44        throws Exception {
45
46      return dao.listPage(bno, cri);
47    }
48
49    @Override
50    public int count(Integer bno) throws Exception {
51
52      return dao.count(bno);
53    }
54
55  }
```

2.3 REST 방식의 ReplyController 작성

2.3.1 등록 처리

■ ReplyController.java는 아래와 같이 작성한다.

[org.zerock.controller.ReplyController.java]

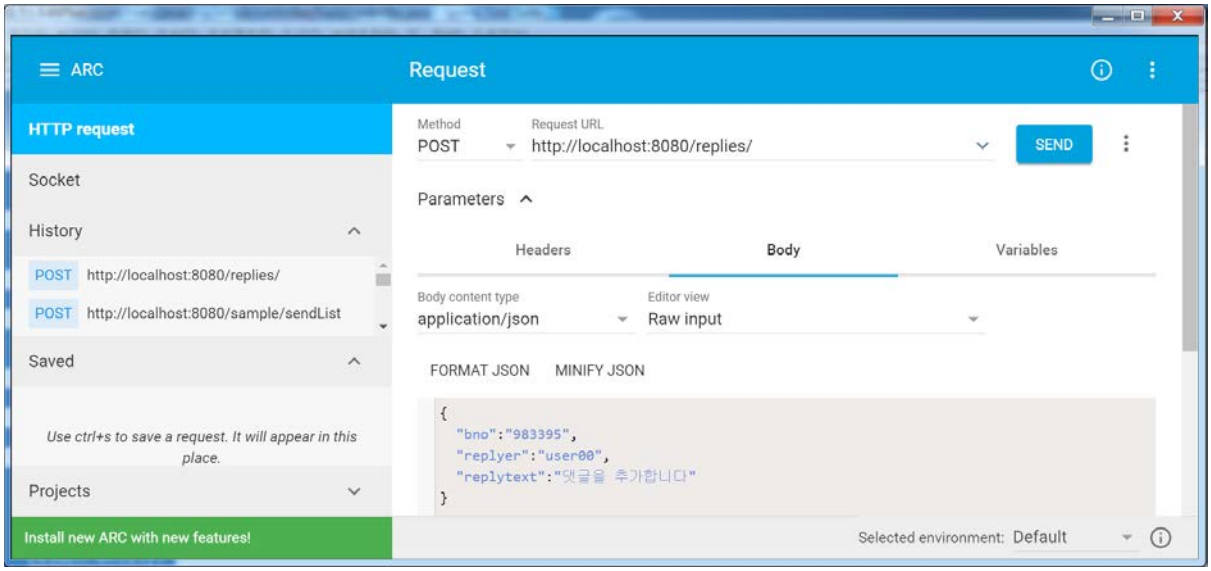
```
01  package org.zerock.controller;
02
03  import java.util.HashMap;
04  import java.util.List;
05  import java.util.Map;
06
07  import javax.inject.Inject;
08
09  import org.springframework.http.HttpStatus;
10  import org.springframework.http.ResponseEntity;
11  import org.springframework.web.bind.annotation.PathVariable;
12  import org.springframework.web.bind.annotation.RequestBody;
13  import org.springframework.web.bind.annotation.RequestMapping;
14  import org.springframework.web.bind.annotation.RequestMethod;
15  import org.springframework.web.bind.annotation.RestController;
16  import org.zerock.domain.Criteria;
17  import org.zerock.domain.PageMaker;
18  import org.zerock.domain.ReplyVO;
19  import org.zerock.service.ReplyService;
20
21  @RestController
22  @RequestMapping("/replies")
23  public class ReplyController {
24
25    @Inject
26    private ReplyService service;
27
28    @RequestMapping(value = "", method = RequestMethod.POST)
29    public ResponseEntity<String> register(@RequestBody ReplyVO vo) {
30
31      ResponseEntity<String> entity = null;
32      try {
33        service.addReply(vo);
34        entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
35      } catch (Exception e) {
36        e.printStackTrace();
37        entity = new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
38      }
39      return entity;
40    }
41
42    @RequestMapping(value = "/all/{bno}", method = RequestMethod.GET)
43    public ResponseEntity<List<ReplyVO>> list(@PathVariable("bno") Integer bno) {
44
45      ResponseEntity<List<ReplyVO>> entity = null;
46      try {
47        entity = new ResponseEntity<>(service.listReply(bno), HttpStatus.OK);
48
49      } catch (Exception e) {
50        e.printStackTrace();
51        entity = new ResponseEntity<>(HttpStatus.BAD_REQUEST);
52      }
53
54      return entity;
55    }
56
57    @RequestMapping(value = "/{rno}", method = { RequestMethod.PUT, RequestMethod.PATCH })
58    public ResponseEntity<String> update(@PathVariable("rno") Integer rno, @RequestBody ReplyVO vo) {
```

```
59
60        ResponseEntity<String> entity = null;
61        try {
62          vo.setRno(rno);
63          service.modifyReply(vo);
64
65          entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
66        } catch (Exception e) {
67          e.printStackTrace();
68          entity = new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
69        }
70        return entity;
71      }
72
73      @RequestMapping(value = "/{rno}", method = RequestMethod.DELETE)
74      public ResponseEntity<String> remove(@PathVariable("rno") Integer rno) {
75
76        ResponseEntity<String> entity = null;
77        try {
78          service.removeReply(rno);
79          entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
80        } catch (Exception e) {
81          e.printStackTrace();
82          entity = new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
83        }
84        return entity;
85      }
86
87      @RequestMapping(value = "/{bno}/{page}", method = RequestMethod.GET)
88      public ResponseEntity<Map<String, Object>> listPage(
89          @PathVariable("bno") Integer bno,
90          @PathVariable("page") Integer page) {
91
92        ResponseEntity<Map<String, Object>> entity = null;
93
94        try {
95          Criteria cri = new Criteria();
96          cri.setPage(page);
97
98          PageMaker pageMaker = new PageMaker();
99          pageMaker.setCri(cri);
100
101         Map<String, Object> map = new HashMap<String, Object>();
102         List<ReplyVO> list = service.listReplyPage(bno, cri);
103
104         map.put("list", list);
105
106         int replyCount = service.count(bno);
107         pageMaker.setTotalCount(replyCount);
108
109         map.put("pageMaker", pageMaker);
110
111         entity = new ResponseEntity<Map<String, Object>>(map, HttpStatus.OK);
112
113       } catch (Exception e) {
114         e.printStackTrace();
115         entity = new ResponseEntity<Map<String, Object>>(HttpStatus.BAD_REQUEST);
116       }
117       return entity;
118     }
119
120   }
```

■ Advanced REST Client를 이용한 테스트는 아래와 같이 작성한다.

- Request Method: POST
- Request URL: http://localhost:8080/replies
- Parameters > Body 메뉴에서 Body content type = "application/json", Editor view = "Raw input"을 선택한 후, 아래 코드를 입력한다.

```
{
  "bno": "510",
  "replyer": "user00",
  "replytext": "댓글을 추가합니다"
}
```



■ 결과가 정상적으로 처리된다면 최종적으로 데이터베이스에 정상적으로 추가되었는지 확인한다. 예) SELECT * FROM book_ex.tbl_reply;

2.3.2 특정 게시물의 전체 댓글 목록의 처리

■ 특정 게시물의 전체 댓글의 목록은 GET 방식으로 처리돼야 하고, 반드시 특정 게시물의 번호(bno)를 필요로 하기 때문에 다음과 같은 형태로 작성한다.

```
// org.zerock.controller.ReplyController.java 일부

  @RequestMapping(value = "/all/{bno}", method = RequestMethod.GET)
  public ResponseEntity<List<ReplyVO>> list(@PathVariable("bno") Integer bno) {

    ResponseEntity<List<ReplyVO>> entity = null;
    try {
      entity = new ResponseEntity<>(service.listReply(bno), HttpStatus.OK);

    } catch (Exception e) {
      e.printStackTrace();
      entity = new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    return entity;
  }
```

■ http://localhost:8080/replies/all/6127

2.3.3 수정 처리

- REST 방식에서 update 작업은 PUT, PATCH 방식을 이용해서 처리한다. 일반적으로 전체 데이터를 수정하는 경우에는 PUT을 이용하고, 일부의 데이터를 수정하는 경우에는 PATCH를 이용한다.

```
// org.zerock.controller.ReplyController.java 일부

  @RequestMapping(value = "/{rno}", method = { RequestMethod.PUT, RequestMethod.PATCH })
  public ResponseEntity<String> update(@PathVariable("rno") Integer rno, @RequestBody ReplyVO vo) {

    ResponseEntity<String> entity = null;
    try {
      vo.setRno(rno);
      service.modifyReply(vo);

      entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
    } catch (Exception e) {
      e.printStackTrace();
      entity = new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
    }
    return entity;
  }
```

- http://localhost:8080/replies/6127

2.3.4 삭제 처리

- 댓글의 삭제에 대한 처리는 PUT과 유사하다. 다만, 추가적인 데이터가 없다는 부분 때문에 더 간단히 처리할 수 있다.

```
// org.zerock.controller.ReplyController.java 일부

    @RequestMapping(value = "/{rno}", method = RequestMethod.DELETE)
    public ResponseEntity<String> remove(@PathVariable("rno") Integer rno) {

      ResponseEntity<String> entity = null;
      try {
        service.removeReply(rno);
        entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
      } catch (Exception e) {
        e.printStackTrace();
        entity = new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
      }
      return entity;
    }
```

- http://localhost:8080/replies/6127

2.4 화면에서의 Ajax 호출

- @RestController를 이용하는 경우에는 데이터만 만들어져 전송되기 때문에 사실상 개발의 대

부분은 이를 처리하는 화면으로 집중된다.

- 웹에서는 HTTP 방식으로 데이터를 주고받을 수 있는 라이브러리 등을 활용해서 데이터를 처리하고, 웹에서는 주로 Ajax를 이용해서 처리한다.

2.4.1 개발의 순서 결정

- 작업의 순서는 다음과 같이 진행한다.
  - 특정 게시물을 미리 지정하고, 전체 댓글의 목록을 가져와서 출력하는 기능을 작성한다.
  - 댓글을 입력할 수 있는 화면을 구성하고 이를 이용해서 새로운 댓글을 추가한다.
  - 댓글의 목록에서 특정 게시물을 선택해서 수정과 삭제 작업이 이루어지는 화면을 구성한다.

(1) 테스트를 위한 컨트롤러와 JSP

```
// org.zerock.controller.HomeController의 일부
@RequestMapping(value = "/test", method = RequestMethod.GET)
public void ajaxTest() {
}
```

```
// WEB-INF/views/test.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>

<style>
#modDiv {
          width: 300px;
          height: 100px;
          background-color: gray;
          position: absolute;
          top: 50%;
          left: 50%;
          margin-top: -50px;
          margin-left: -150px;
          padding: 10px;
          z-index: 1000;
}

.pagination {
  width: 100%;
}

.pagination li{
  list-style: none;
  float: left;
  padding: 3px;
  border: 1px solid blue;
  margin:3px;
```

```
}

.pagination li a{
  margin: 3px;
  text-decoration: none;
}

</style>
</head>

<body>


          <div id='modDiv' style="display: none;">
                    <div class='modal-title'></div>
                    <div>
                              <input type='text' id='replytext'>
                    </div>
                    <div>
                              <button type="button" id="replyModBtn">Modify</button>
                              <button type="button" id="replyDelBtn">DELETE</button>
                              <button type="button" id='closeBtn'>Close</button>
                    </div>
          </div>

          <h2>Ajax Test Page</h2>

          <div>
                    <div>
                              REPLYER <input type='text' name='replyer' id='newReplyWriter'>
                    </div>
                    <div>
                              REPLY TEXT <input type='text' name='replytext' id='newReplyText'>
                    </div>
                    <button id="replyAddBtn">ADD REPLY</button>
          </div>



          <ul id="replies">
          </ul>

          <ul class='pagination'>
          </ul>


          <!-- jQuery 2.1.4 -->
          <script src="/resources/plugins/jQuery/jQuery-2.1.4.min.js"></script>

          <script>

                    /* var bno = 123239; */
                    var bno = 983015;

                    getPageList(1);

                    function getAllList() {

                              $
                                        .getJSON(
                                                  "/replies/all/" + bno,
                                                  function(data) {

                                                            //console.log(data.length);
```

```
                                                                var str = "";

                                                                $(data)
                                                                                        .each(

        function() {

                str += "<li data-rno='"+this.rno+"' class='replyLi'>"

                                        + this.rno

                                        + ":"

                                        + this.replytext

                                        + "<button>MOD</button></li>";

        });

                                                                $("#replies").html(str);
                                                        });
        }

        $("#replyAddBtn").on("click", function() {

                var replyer = $("#newReplyWriter").val();
                var replytext = $("#newReplyText").val();

                $.ajax({
                        type : 'post',
                        url : '/replies',
                        headers : {
                                "Content-Type" : "application/json",
                                "X-HTTP-Method-Override" : "POST"
                        },
                        dataType : 'text',
                        data : JSON.stringify({
                                bno : bno,
                                replyer : replyer,
                                replytext : replytext
                        }),
                        success : function(result) {

                                if (result == 'SUCCESS') {

                                        alert("등록 되었습니다.");
                                        getAllList();

                                }
                        }
                });
        });

        $("#replies").on("click", ".replyLi button", function() {

                var reply = $(this).parent();

                var rno = reply.attr("data-rno");
                var replytext = reply.text();

                $(".modal-title").html(rno);
                $("#replytext").val(replytext);
                $("#modDiv").show("slow");

        });
```

```
$("#replyDelBtn").on("click", function() {

          var rno = $(".modal-title").html();
          var replytext = $("#replytext").val();

          $.ajax({
                    type : 'delete',
                    url : '/replies/' + rno,
                    headers : {
                              "Content-Type" : "application/json",
                              "X-HTTP-Method-Override" : "DELETE"
                    },
                    dataType : 'text',
                    success : function(result) {
                              console.log("result: " + result);
                              if (result == 'SUCCESS') {
                                        alert("삭제 되었습니다.");
                                        $("#modDiv").hide("slow");
                                        getAllList();
                              }
                    }
          });
});

$("#replyModBtn").on("click",function(){

            var rno = $(".modal-title").html();
            var replytext = $("#replytext").val();

            $.ajax({
                              type:'put',
                              url:'/replies/'+rno,
                              headers: {
                                    "Content-Type": "application/json",
                                    "X-HTTP-Method-Override": "PUT" },
                              data:JSON.stringify({replytext:replytext}),
                              dataType:'text',
                              success:function(result){
                                        console.log("result: " + result);
                                        if(result == 'SUCCESS'){
                                                  alert("수정 되었습니다.");
                                                   $("#modDiv").hide("slow");
                                                  //getAllList();
                                                   getPageList(replyPage);
                                        }
                    }});
});

function getPageList(page){

  $.getJSON("/replies/"+bno+"/"+page , function(data){

            console.log(data.list.length);

            var str ="";

            $(data.list).each(function(){
                      str+= "<li data-rno='"+this.rno+"' class='replyLi'>"
                      +this.rno+":"+ this.replytext+
                      "<button>MOD</button></li>";
            });

            $("#replies").html(str);
```

```
                        printPaging(data.pageMaker);

                });
        }


                function printPaging(pageMaker){

                        var str = "";

                        if(pageMaker.prev){
                                str += "<li><a href='"+(pageMaker.startPage-1)+"'> << </a></li>";
                        }

                        for(var i=pageMaker.startPage, len = pageMaker.endPage; i <= len; i++){

                                var strClass= pageMaker.cri.page == i?'class=active':'';
                          str += "<li "+strClass+"><a href='"+i+"'>"+i+"</a></li>";
                        }

                        if(pageMaker.next){
                                str += "<li><a href='"+(pageMaker.endPage + 1)+"'> >> </a></li>";
                        }
                        $('.pagination').html(str);
                }

                var replyPage = 1;

                $(".pagination").on("click", "li a", function(event){

                        event.preventDefault();

                        replyPage = $(this).attr("href");

                        getPageList(replyPage);

                });



        </script>

</body>
</html>
```

2.4.2 전체 댓글 목록의 테스트

- 개발자 도구에서 Network 탭을 선택해서 ReplyController를 호출한 부분을 선택하면 전달된 데이터의 상세한 내용을 확인한다. 예) http://localhost:8080/test
- 현재 댓글 목록의 경우 <ul id="replies"> 요소에 출력되므로 문자열을 구성해서 화면에 보이게 작성한다.
- 전체 목록을 갱신하는 부분은 여러 번 사용되기 때문에 별도의 함수로 만들어서 처리한다. Ajax 호출 부분을 getAllList() 함수로 작성한다.

```
// WEB-INF/views/test.jsp의 일부
```

```
        <script>
                var bno = 983015;

                getPageList(1);

                function getAllList() {

                        $.getJSON("/replies/all/" + bno, function(data) {
                            //console.log(data.length);
                            var str = "";

                            $(data).each(
                                  function() {
                                        str += "<li data-rno='"+this.rno+"' class='replyLi'>"
                                            + this.rno + ":" + this.replytext
                                            + "<button>MOD</button></li>";
                                  });

                            $("#replies").html(str);
                        });
                }
```

2.4.3 댓글 등록 화면 및 테스트

```
// /WEB-INF/views/test.jsp의 일부

        <h2>Ajax Test Page</h2>

        <div>
                <div>
                        REPLYER <input type='text' name='replyer' id='newReplyWriter'>
                </div>
                <div>
                        REPLY TEXT <input type='text' name='replytext' id='newReplyText'>
                </div>
                <button id="replyAddBtn">ADD REPLY</button>
        </div>

        ...(생략)...
        <script>
                $("#replyAddBtn").on("click", function() {

                        var replyer = $("#newReplyWriter").val();
                        var replytext = $("#newReplyText").val();

                        $.ajax({
                                type : 'post',
                                url : '/replies',
                                headers : {
                                        "Content-Type" : "application/json",
                                        "X-HTTP-Method-Override" : "POST"
                                },
                                dataType : 'text',
                                data : JSON.stringify({
                                        bno : bno,
                                        replyer : replyer,
                                        replytext : replytext
                                }),
                                success : function(result) {

                                        if (result == 'SUCCESS') {
```

```
                                                alert("등록 되었습니다.");
                                                getAllList();

                                        }
                                }
                        });
                });
        </script>
```

2.4.4 댓글 조회 및 수정/삭제

(1) 수정과 삭제를 위한

```
// WEB-INF/views/test.jsp의 일부

        <div id='modDiv' style="display: none;">
                <div class='modal-title'></div>
                <div>
                        <input type='text' id='replytext'>
                </div>
                <div>
                        <button type="button" id="replyModBtn">Modify</button>
                        <button type="button" id="replyDelBtn">DELETE</button>
                        <button type="button" id='closeBtn'>Close</button>
                </div>
        </div>

...(생략)...

<style>
#modDiv {
        width: 300px;
        height: 100px;
        background-color: gray;
        position: absolute;
        top: 50%;
        left: 50%;
        margin-top: -50px;
        margin-left: -150px;
        padding: 10px;
        z-index: 1000;
}
</style>
```

(2) 에 댓글 보이기

■ 항목을 선택했을 때 댓글의 번호와 내용을 를 이용해서 보여지도록 다음과 같이 수정한다.

```
// WEB-INF/views/test.jsp의 일부

                $("#replies").on("click", ".replyLi button", function() {

                        var reply = $(this).parent();

                        var rno = reply.attr("data-rno");
```

```
                        var replytext = reply.text();

                        $(".modal-title").html(rno);
                        $("#replytext").val(replytext);
                        $("#modDiv").show("slow");

                });
```

(3) 삭제 호출하기

■ 화면에 보이는 에서 'DELETE' 버튼을 선택하면 삭제가 되게 처리한다. 삭제는 HTTP의 DELETE 방식으로 동작하도록 설계해야만 한다.

```
// WEB-INF/views/test.jsp의 일부

                $("#replyDelBtn").on("click", function() {

                        var rno = $(".modal-title").html();
                        var replytext = $("#replytext").val();

                        $.ajax({
                                type : 'delete',
                                url : '/replies/' + rno,
                                headers : {
                                        "Content-Type" : "application/json",
                                        "X-HTTP-Method-Override" : "DELETE"
                                },
                                dataType : 'text',
                                success : function(result) {
                                        console.log("result: " + result);
                                        if (result == 'SUCCESS') {
                                                alert("삭제 되었습니다.");
                                                $("#modDiv").hide("slow");
                                                getAllList();
                                        }
                                }
                        });
                });
```

(4) 수정 작업 처리하기

■ 댓글 수정의 처리에는 PUT 방식이 사용되고, 수정되는 게시물의 번호는 URI에 추가해서 전송한다.

```
// WEB-INF/views/test.jsp의 일부


                $("#replyModBtn").on("click",function(){

                          var rno = $(".modal-title").html();
                          var replytext = $("#replytext").val();

                          $.ajax({
                                          type:'put',
                                          url:'/replies/'+rno,
                                          headers: {
                                                "Content-Type": "application/json",
                                                "X-HTTP-Method-Override": "PUT" },
```

```
                                        data:JSON.stringify({replytext:replytext}),
                                        dataType:'text',
                                        success:function(result){
                                                console.log("result: " + result);
                                                if(result == 'SUCCESS'){
                                                        alert("수정 되었습니다.");
                                                         $("#modDiv").hide("slow");
                                                        //getAllList();
                                                         getPageList(replyPage);
                                                }
                                });
                        });
```

2.5 게시물 관리의 댓글 적용

2.5.1 조회 화면의 수정

■ 기존의 게시물 조회 화면에서 댓글 관련 기능을 추가해야 하기 때문에 HTML의 변경이 필요하다. 화면에 추가되는 영역은 크게 세가지이다.
- 댓글을 입력하는 부분
- 댓글의 목록을 보여주는 부분
- 댓글의 페이징 처리

```
// WEB-INF/views/sboard/readPage.jsp의 일부

          <div class="row">
                    <div class="col-md-12">

                              <div class="box box-success">
                                        <div class="box-header">
                                                  <h3 class="box-title">ADD NEW REPLY</h3>
                                        </div>
                                        <div class="box-body">
                                                  <label for="exampleInputEmail1">Writer</label> <input
                                                            class="form-control" type="text"
placeholder="USER ID"
                                                            id="newReplyWriter"> <label
for="exampleInputEmail1">Reply
                                                            Text</label> <input class="form-control"
type="text"
                                                            placeholder="REPLY TEXT" id="newReplyText">

                                        </div>
                                        <!-- /.box-body -->
                                        <div class="box-footer">
                                                  <button type="button" class="btn btn-primary"
id="replyAddBtn">ADD
                                                            REPLY</button>
                                        </div>
                              </div>


                              <!-- The time line -->
                              <ul class="timeline">
                                        <!-- timeline time label -->
                                        <li class="time-label" id="repliesDiv"><span class="bg-green">
                                                            Replies List </span></li>
                              </ul>
```

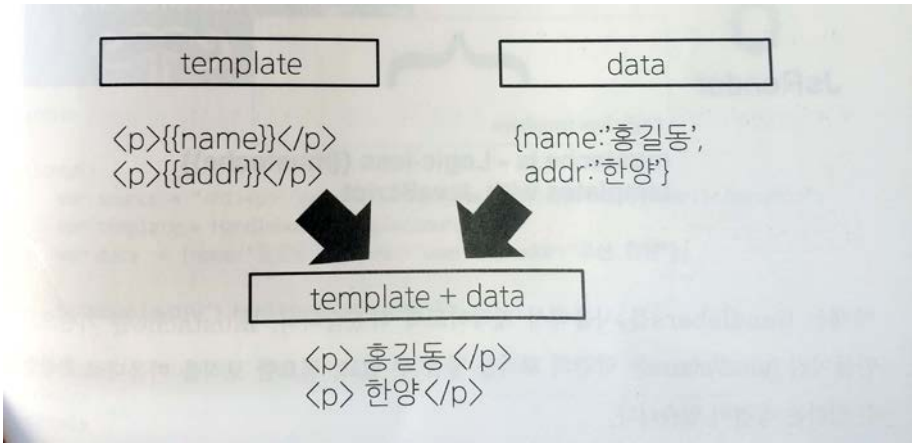
```
                        <div class='text-center'>
                                <ul id="pagination" class="pagination pagination-sm no-margin ">

                                </ul>
                        </div>

                </div>
                <!-- /.col -->
        </div>
        <!-- /.row -->
```

2.5.2 handlebars를 이용한 템플릿

- JavaScript의 템플릿을 적용하여 '만들어진 HTML 코드에 데이터(객체)를 넣어 찍어내는 틀'로 사용한다.



- 현재 개발자 사이에서 많이 사용되는 템플릿은 jQuery Template Plugin에서 발전한 JS Render, Mustache, Mustache를 기반으로 작성된 handlebars, Hogan 등이 있다.

(1) handlebars 사용 연습

```
// webapp/resources/ex00.html

<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/3.B.1/handlebars.js"></script>


<div id="displayDiv">

</div>

<script>
    var source = "<h1><p>{{name}}</p> <p>{{userid}}</p> <p>{{addr}}</p></h1>";
```

```
    var template = Handlebars.compile(source);
    var data  = {name:"홍길동", userid:"user00", addr:"조선 한양"};

    $("#displayDiv").html(template(data));

</script>
</body>
</html>
```

```
// webapp/resources/ex01.html

<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/3.B.1/handlebars.js"></script>


<div id="displayDiv">

</div>

<script id="template" type="text/x-handlebars-template">
    <span> {{ name }}</span>
    <div>
        <span> {{ userid }}</span>
        <span> {{ addr }}</span>
    </div>
</script>
<script>
    var source = $("#template").html();
    var template = Handlebars.compile(source);
    var data  = {name:"홍길동", userid:"user00", addr:"조선 한양"};

    $("#displayDiv").html(template(data));

</script>
</body>
</html>
```

```
// webapp/resources/ex02.html

<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/3.B.1/handlebars.js"></script>
```

```
<ul id="replies">

</ul>

<script id="template" type="text/x-handlebars-template">
    {{#each .}}
    <li class="replyLi">
        <div>{{rno}}</div>
        <div>{{replytext}}</div>
        <div>{{replydate}}</div>
    </li>
    {{/each}}
</script>
<script>
    var source = $("#template").html();
    var template = Handlebars.compile(source);
    var data  = [
        {rno:1, replytext:'1번 댓글....', replydate:new Date()},
        {rno:2, replytext:'2번 댓글....', replydate:new Date()},
        {rno:3, replytext:'3번 댓글....', replydate:new Date()},
        {rno:4, replytext:'4번 댓글....', replydate:new Date()},
        {rno:5, replytext:'5번 댓글....', replydate:new Date()}
    ];
    $("#replies").html(template(data));

</script>
</body>
</html>
```

2.5.3 댓글 목록 처리

■ 가장 먼저 처리하는 작업은 댓글을 페이징 처리해서 화면에 목록을 보여주는 작업이다.

```
// WEB-INF/views/sboard/readPage.jsp의 일부

<script id="template" type="text/x-handlebars-template">
{{#each .}}
<li class="replyLi" data-rno={{rno}}>
<i class="fa fa-comments bg-blue"></i>
 <div class="timeline-item" >
  <span class="time">
    <i class="fa fa-clock-o"></i>{{prettifyDate regdate}}
  </span>
  <h3 class="timeline-header"><strong>{{rno}}</strong> -{{replyer}}</h3>
  <div class="timeline-body">{{replytext}} </div>
    <div class="timeline-footer">
     <a class="btn btn-primary btn-xs"
             data-toggle="modal" data-target="#modifyModal">Modify</a>
    </div>
  </div>
</li>
{{/each}}
</script>
```

■ 'prettifyDate regdate'에는 handlebar의 기능을 확장하는 방법의 예를 사용된다. handlebars는 helper라는 기능을 이용해서 데이터의 상세한 처리에 필요한 기능을 처리한다. 기능이 없는 경우에는 registerHelper()를 이용해서 사용자가 새로운 기능을 추가할 수 있다.

```
// WEB-INF/views/sboard/readPage.jsp의 일부

<script>
	Handlebars.registerHelper("prettifyDate", function(timeValue) {
		var dateObj = new Date(timeValue);
		var year = dateObj.getFullYear();
		var month = dateObj.getMonth() + 1;
		var date = dateObj.getDate();
		return year + "/" + month + "/" + date;
	});

	var printData = function(replyArr, target, templateObject) {

		var template = Handlebars.compile(templateObject.html());

		var html = template(replyArr);
		$(".replyLi").remove();
		target.after(html);

	}
</script>
```

- getPage()는 특정한 게시물에 대한 페이징 처리를 위해서 호출되는 함수이다. 내부적으로 jQuery를 이용해서 JSON 타입의 데이터를 처리한다.

```
// WEB-INF/views/sboard/readPage.jsp의 일부

	var bno = ${boardVO.bno};

	var replyPage = 1;

	function getPage(pageInfo) {

		$.getJSON(pageInfo, function(data) {
			printData(data.list, $("#repliesDiv"), $('#template'));
			printPaging(data.pageMaker, $(".pagination"));

			$("#modifyModal").modal('hide');

		});
	}

	var printPaging = function(pageMaker, target) {

		var str = "";

		if (pageMaker.prev) {
			str += "<li><a href='" + (pageMaker.startPage - 1)
					+ "'> << </a></li>";
		}

		for (var i = pageMaker.startPage, len = pageMaker.endPage; i <= len; i++) {
			var strClass = pageMaker.cri.page == i ? 'class=active' : '';
			str += "<li "+strClass+"><a href='"+i+"'>" + i + "</a></li>";
		}

		if (pageMaker.next) {
			str += "<li><a href='" + (pageMaker.endPage + 1)
					+ "'> >> </a></li>";
		}

		target.html(str);
	};
```

2.5.4 새로운 댓글의 등록

■ 댓글의 등록 작업은 'replyAddBtn'을 누르는 순간 처리되도록 한다.

```
// WEB-INF/views/sboard/readPage.jsp의 일부

          $("#replyAddBtn").on("click",function(){

                    var replyerObj = $("#newReplyWriter");
                    var replytextObj = $("#newReplyText");
                    var replyer = replyerObj.val();
                    var replytext = replytextObj.val();


                     $.ajax({
                                        type:'post',
                                        url:'/replies/',
                                        headers: {
                                              "Content-Type": "application/json",
                                              "X-HTTP-Method-Override": "POST" },
                                        dataType:'text',
                                        data: JSON.stringify({bno:bno, replyer:replyer,
replytext:replytext}),
                                        success:function(result){
                                                  console.log("result: " + result);
                                                  if(result == 'SUCCESS'){
                                                            alert("등록 되었습니다.");
                                                            replyPage = 1;
                                                            getPage("/replies/"+bno+"/"+replyPage );
                                                            replyerObj.val("");
                                                            replytextObj.val("");
                                                  }
                                  }});
          });
```

2.5.5 수정과 삭제를 위한 Modal창

■ 게시물의 수정과 삭제 작업은 별도의 Modal(파업과 유사하지만 다른 작업을 할 수 없도록 제한된 팝업)을 이용해서 처리하도록 한다.

```
// WEB-INF/views/sboard/readPage.jsp의 일부

<!-- Modal -->
<div id="modifyModal" class="modal modal-primary fade" role="dialog">
  <div class="modal-dialog">
    <!-- Modal content-->
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal">&times;</button>
        <h4 class="modal-title"></h4>
      </div>
      <div class="modal-body" data-rno>
        <p><input type="text" id="replytext" class="form-control"></p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-info" id="replyModBtn">Modify</button>
        <button type="button" class="btn btn-danger" id="replyDelBtn">DELETE</button>
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
```

```
      </div>
    </div>
  </div>
</div>
```

2.5.6 수정과 삭제의 처리

■ 수정 작업의 처리는 HTTP의 PUT 방식을 통해서 처리하고, 삭제 처리는 HTTP의 DELETE 방식을 이용해서 처리한다.

```
// WEB-INF/views/sboard/readPage.jsp의 일부

         $("#replyModBtn").on("click",function(){

                     var rno = $(".modal-title").html();
                     var replytext = $("#replytext").val();

                     $.ajax({
                                      type:'put',
                                      url:'/replies/'+rno,
                                      headers: {
                                            "Content-Type": "application/json",
                                            "X-HTTP-Method-Override": "PUT" },
                                      data:JSON.stringify({replytext:replytext}),
                                      dataType:'text',
                                      success:function(result){
                                                console.log("result: " + result);
                                                if(result == 'SUCCESS'){
                                                          alert("수정 되었습니다.");
                                                          getPage("/replies/"+bno+"/"+replyPage );
                                                }
                              }});
         });

         $("#replyDelBtn").on("click",function(){

                     var rno = $(".modal-title").html();
                     var replytext = $("#replytext").val();

                     $.ajax({
                                      type:'delete',
                                      url:'/replies/'+rno,
                                      headers: {
                                            "Content-Type": "application/json",
                                            "X-HTTP-Method-Override": "DELETE" },
                                      dataType:'text',
                                      success:function(result){
                                                console.log("result: " + result);
                                                if(result == 'SUCCESS'){
                                                          alert("삭제 되었습니다.");
                                                          getPage("/replies/"+bno+"/"+replyPage );
                                                }
                              }});
         });
```