Part 3 Ajax 댓글 처리

Chapter 03 REST 방식의 ReplyController 작성

- @RestContoller를 이용해서 작성

URI	전송방식	설명
/replies/ + JSON 데이터	POST	새로운 댓글 등록
/replies/ + JSON 데이터	PUT, PATCH	댓글 수정
/replies/댓글 번호	DELETE	댓글 삭제

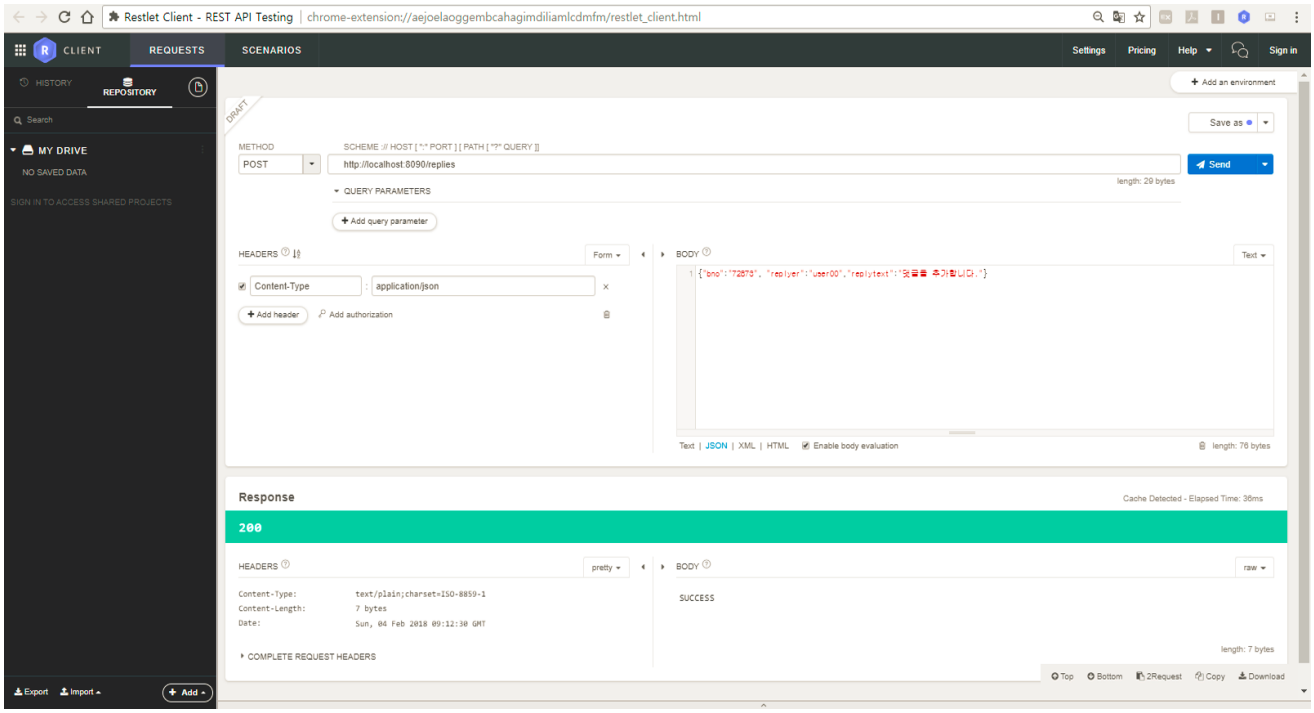
- REST 방식의 처리에서 사용하는 어노테이션
  - @PathVariable - URI의 경로에서 원하는 데이터를 추출하는 용도로 사용
  - @RequestBody - 전송된 JSON 데이터를 객체로 변환해주는 어노테이션으로 @ModelAttribute와 유사한 역할을 하지만 JSON에서 사용된다는 점이 차이

3.1 등록 처리

- 등록 작업은 '/replies' URI로 처리되고, POST 방식으로 전송된다.
- ReplyController

```
@RestController
@RequestMapping("/replies")
public class ReplyController {
    @Inject
    private ReplyService service;

    @RequestMapping(value = "", method = RequestMethod.POST)
    public ResponseEntity<String> register(@RequestBody ReplyVO vo) {
        ResponseEntity<String> entity = null;

        try {
            service.addReply(vo);
            entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);

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

- JSON 데이터 결과



3.2 특정 게시물의 전체 댓글 목록의 처리

- 전체 댓글 조회
- ReplyController (list)

```
@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;
}
```

- 결과

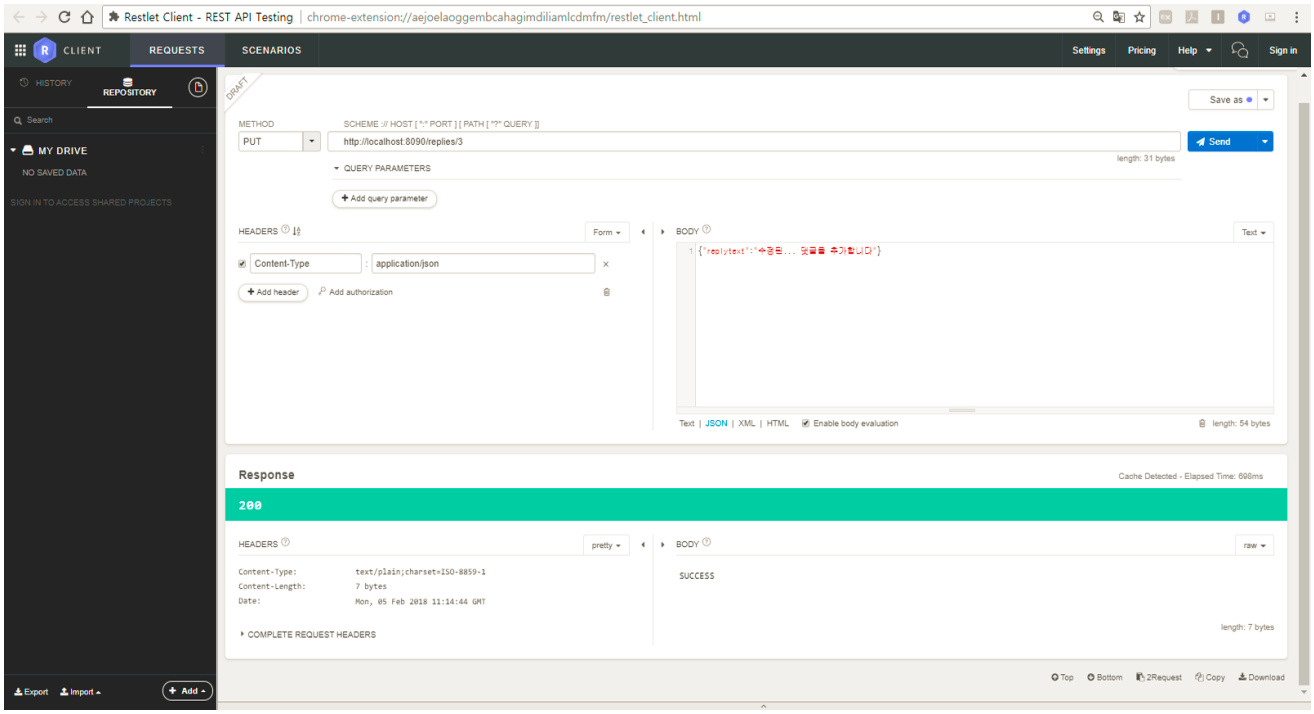
3.3 수정 처리

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

- ReplyController (update)

```
@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;
}
```

- 결과



3.3.1 HiddenMethod의 활용

- 문제
  - 브라우저에 따라 PUT, PATCH, DELETE 방식을 지원하지 않는 경우가 발생할 수 있다.
  - 많은 브라우저가 GET, POST 방식만을 지원하기 때문에 REST 방식을 제대로 사용하려면 브라우저가 지원하지 않는 상황에 고려해야 한다.
- 해결책

    - Overloaded POST: 브라우저에서 POST 방식으로 전송, 추가적인 정보를 이용해서 PUT, PATCH, DELETE와 같은 정보를 같이 전송
    - Ajax를 이용해서 전송
        - 'X-HTTP-Method-Override' 정보를 이용
        - 태그를 이용해서 데이터를 전송하는 경우에는 POST 방식으로 전송하되, '_Method'라는 추가적인 정보를 이용한다.
    - 스프링은 이를 위해 HiddenHttpMethodFilter라는 것을 제공한다.
        - 태그 내에서 과 같은 형태로 사용해서 GET/POST 방식만을 지원하는 브라우저에서 REST 방식을 사용할 수 있도록 설정할 때 사용한다.
        - POST + _method value='put' = PUT 방식
        - POST + _method value='delete' = DELETE 방식

- web.xml (Filter추가)

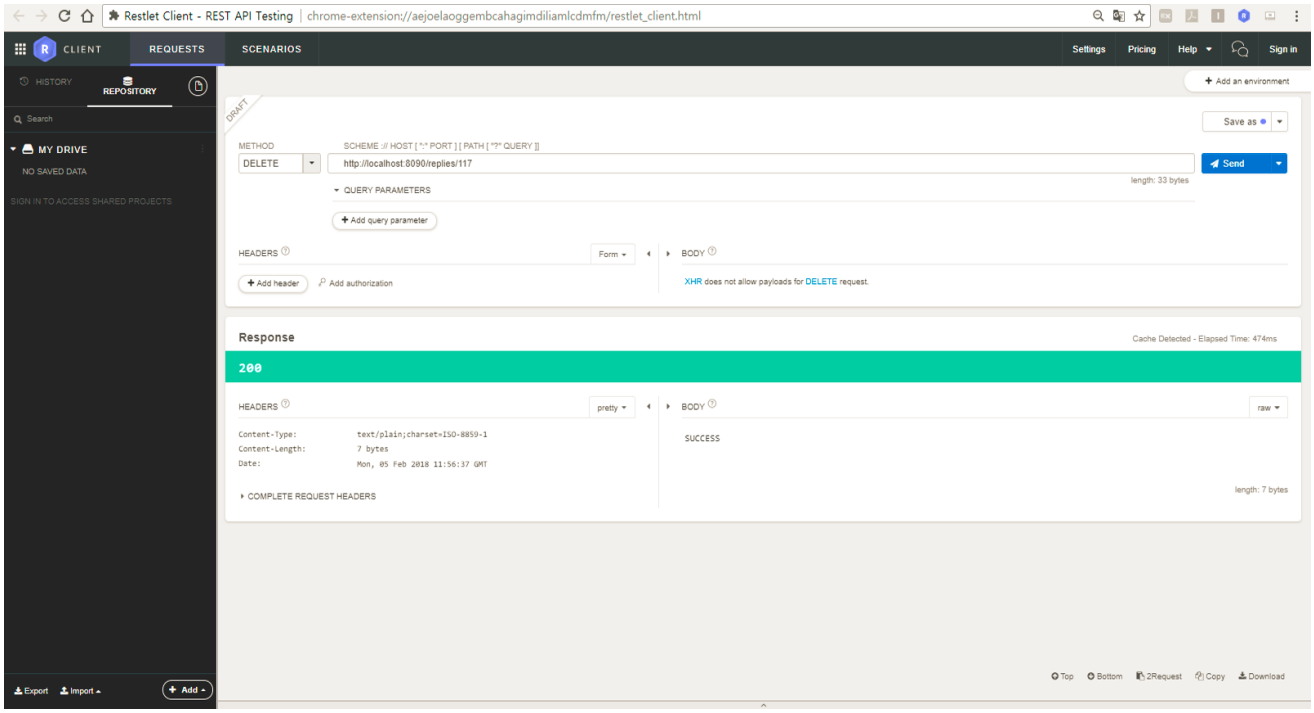
```
<filter>
  <filter-name>hiddenHttpMethodFilter</filter-name>
  <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>hiddenHttpMethodFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.4 삭제 처리

- ReplyController

```
@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;
}
```

- 삭제 처리 결과



3.5 페이징 처리

- Part 2에서 작성한 Criteria 필요
- REST 방식의 경우 전통적인 Model 객체에 데이터를 담지 않고 객체를 처리할 수 있다.

3.5.1 ReplyDAO 처리

- ReplyDAO
  - listPage(), count() 기능 추가

```
public List<ReplyVO> listPage(Integer bno, Criteria cri) throws Exception;

public int count(Integer bno) throws Exception;
```

3.5.2 XML Mapper처리

- replyMapper.xml
  - listPAge(), count()에 대한 SQL문 추가

```
<select id="listPage" resultType="ReplyVO">
  SELECT
    *
  FROM
    tbl_reply
  WHERE

    bno = #{bno}
```

```
  ORDER BY rno DESC
  LIMIT #{cri.pageStart}, #{cri.perPageNum}
</select>
<select id="count" resultType="int">
  SELECT
    count(bno)
  FROM
    tbl_reply
  WHERE
    bno = #{bno}
</select>
```

3.5.3 ReplyDAOImpl 처리

- ReplyDAOImpl

```
@Override
public List<ReplyVO> listPage(Integer bno, Criteria cri) throws Exception {
  Map<String, Object> paramMap = new HashMap<>();
  paramMap.put("bno", bno);
  paramMap.put("cri", cri);

  return session.selectList(namespace + ".listPage", paramMap);
}

@Override
public int count(Integer bno) throws Exception {
  return session.selectOne(namespace + ".count", bno);
}
```

3.5.4 ReplyService 페이징 처리

- ReplyService

```
public List<ReplyVO> listReplyPage(Integer bno, Criteria cri) throws Exception;

public int count(Integer bno) throws Exception;
```

- ReplyServiceImpl

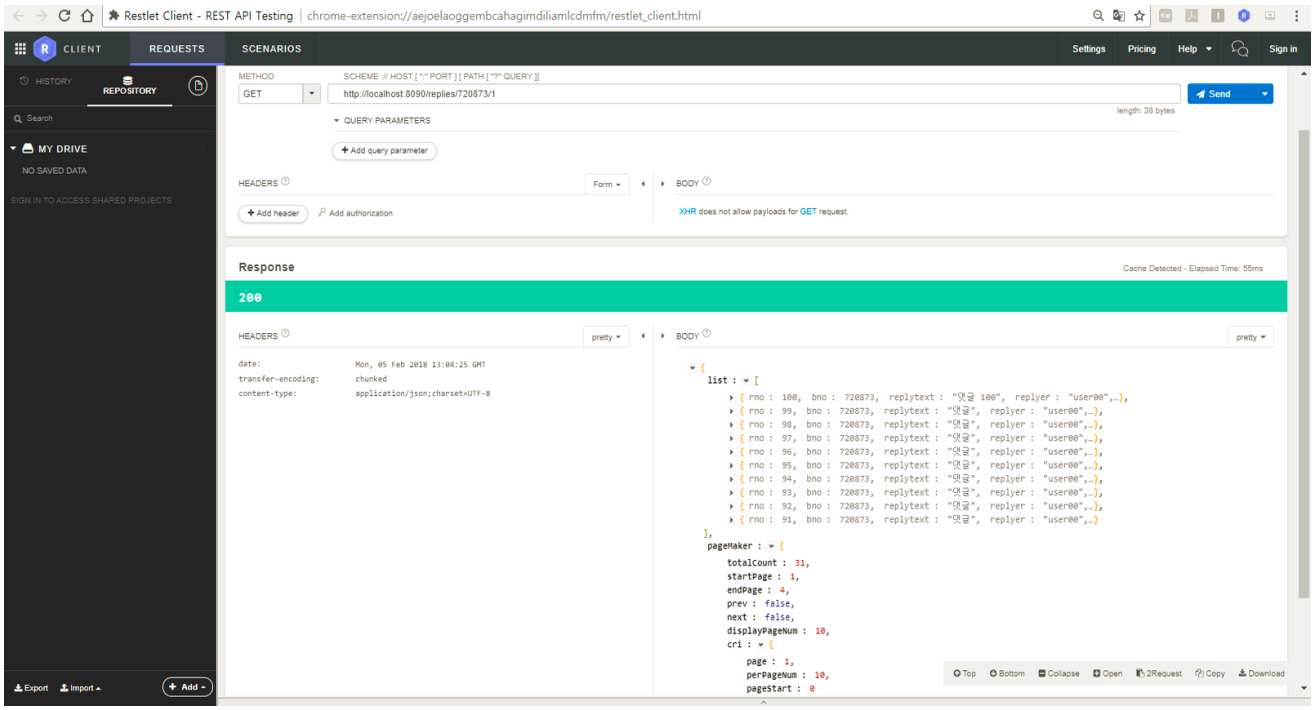
```
@Override
public List<ReplyVO> listReplyPage(Integer bno, Criteria cri) throws Exception {
  return dao.listPage(bno, cri);
}

@Override
public int count(Integer bno) throws Exception {
  return dao.count(bno);
}
```

3.5.5 ReplyController에서의 페이징 처리

- ReplyController
  - 컨트롤러의 경우 두 개의 @PathVariable을 이용해서 처리한다.
    - '/replies/게시물 번호/페이지 번호'
    - GET 방식의 처리
- 페이징 처리를 위해서 Part 2에서 작성된 PageMaker 필요

```
@RequestMapping(value = "/{bno}/{page}", method = RequestMethod.GET)
public ResponseEntity<Map<String, Object>> listPage(@PathVariable("bno") Integer bno,
@PathVariable("page") Integer page) {
  ResponseEntity<Map<String, Object>> entity = null;
  try {

    Criteria cri = new Criteria();
    cri.setPage(page);

    PageMaker pageMaker = new PageMaker();
    pageMaker.setCri(cri);

    Map<String, Object> map = new HashMap<String, Object>();
    List<ReplyVO> list = service.listReplyPage(bno, cri);

    map.put("list", list);

    int replyCount = service.count(bno);
    pageMaker.setTotalCount(replyCount);

    map.put("pageMaker", pageMaker);

    entity = new ResponseEntity<Map<String, Object>>(map, HttpStatus.OK);

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

    entity = new ResponseEntity<Map<String, Object>>(HttpStatus.BAD_REQUEST);
  }
  return entity;

}
```

- 테스트 결과



- listPage() 메서드는 페이징 처리를 위해 '/게시물 번호/페이지 번호' 패턴으로 처리 한다.
- Ajax로 호출될 것이므로 Model을 사용하지 못한다.
- 전달하는 데이터들을 담기 위해 Map타입의 객체를 별도로 생성해야 한다.
- 화면으로 전달되는 Map 데이터는 페이징 처리된 댓글의 목록(list)과 PageMaker클래스의 객체를 담는다.
- 댓글은 페이지당 보여주는 수가 변경되는 경우가 드물기 때문에 10, 20, 50, 100 등으로 결정해 주는 것이 일반적이다.