Part 3 Ajax 댓글 처리

Chapter 05 게시물 관리의 댓글 적용

- Ajax 작업은 화면에서의 구성과 처리에 많은 시간이 들기 때문에 가능하면 서버에서 만들어지는 데이터의 처리를 우선으로 하고, 화면을 구성하는 것이 좋다.
- Part 2에서 작성된 조회 화면(readPage)에 Ajax 기능을 적용해서 특정 게시물에 댓글을 처리하는 것을 구현한다.
- 게시물의 조회 화면 하단에 있는 'Replies List' 버튼을 통해서 해당 게시물에 속한 댓글의 목록을 가져올 수 있다.
- 댓글의 목록은 페이징 처리가 되도록 구성되어, 하단에 페이지 번호를 보여준다.
- 특정 댓글의 수정과 삭제는 댓글 항목 하단의 'Modify' 버튼을 통해서 이루어진다.

5.1 조회 화면의 수정

- HTML 변경
  - 댓글을 입력하는 부분
  - 댓글의 목록을 보여주는 부분
  - 댓글의 페이징 처리
- readPage.jsp
  - 댓글 등록에 필요한

```
<div class="box-footer">
    <button type="submit" class="btn btn-warning"   id="modifyBtn">Modify</button>
    <button type="submit" class="btn btn-danger"    id="removeBtn">REMOVE</button>
    <button type="submit" class="btn btn-primary"   id="goListBtn">GO LIST </button>
</div>
```

```
<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="newReplyWriter">Writer</label>
                <input class="form-control" type="text" placeholder="USER ID"
id="newReplyWriter">
                <label for="newReplyText">ReplyText</label>
                <input class="form-control" type="text" placeholder="REPLY TEXT"
id="newReplyText">
            </div>
            <!--  /.box-body -->
            <div class="box-footer">


                <button type="submit" class="btn btn-primary" id="replyAddBtn">ADD
```

```
REPLY</button>
            </div>
            <!--  /.box-footer -->
        </div>

        <ul class="timeline">
            <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>
        <!-- /.text-center -->
    </div>
    <!--/.col (left) -->
</div>
<!-- /.row -->
```

5.2 handlebars를 이용한 템플릿

- 가장 핵심적인 부분
  - 지속적을호 목록이 갱신되어서 화면에 출력되는 부분
  - 목록 출력은
  - 가 반복적으로 구성되고, 이를
    태그의 내용물로 추가하는 방식으로 동작
  - JavaScript 템플릿은 '만들어진 HTML 코드에 데이터(객체)를 넣어 찍어내는 틀'이라고 한다.
  - 템플릿의 장점
    - 데이터와 뷰를 분리해서 처리
      - 복잡한 HTML 태그를 이용해서 데이터를 구성하는 경우 문자열로 작성되는 결과를 알아보기 어렵다.
    - 템플릿을 변경하기가 쉽다.
      - 템플릿의 내용물만을 변경하기 때문에 유지보수에 더 편리하게 사용할 수 있다.
    - 템플릿의 재사용이 용이하다.
      - 템플릿만을 별도로 관리하면 필요할 때 별도의 코드를 다시 작성하지 않고도 적용이 가능하기 때문에 생산성을 높일 수 있다.
  - jQuery Template Plugin에서 발전한 JS Render나 Mustache, Mustache를 기반으로 작성된 handlebars, Hogan 등을 많이 사용한다.
    - handlebars를 사용해서 제작하도록 한다.

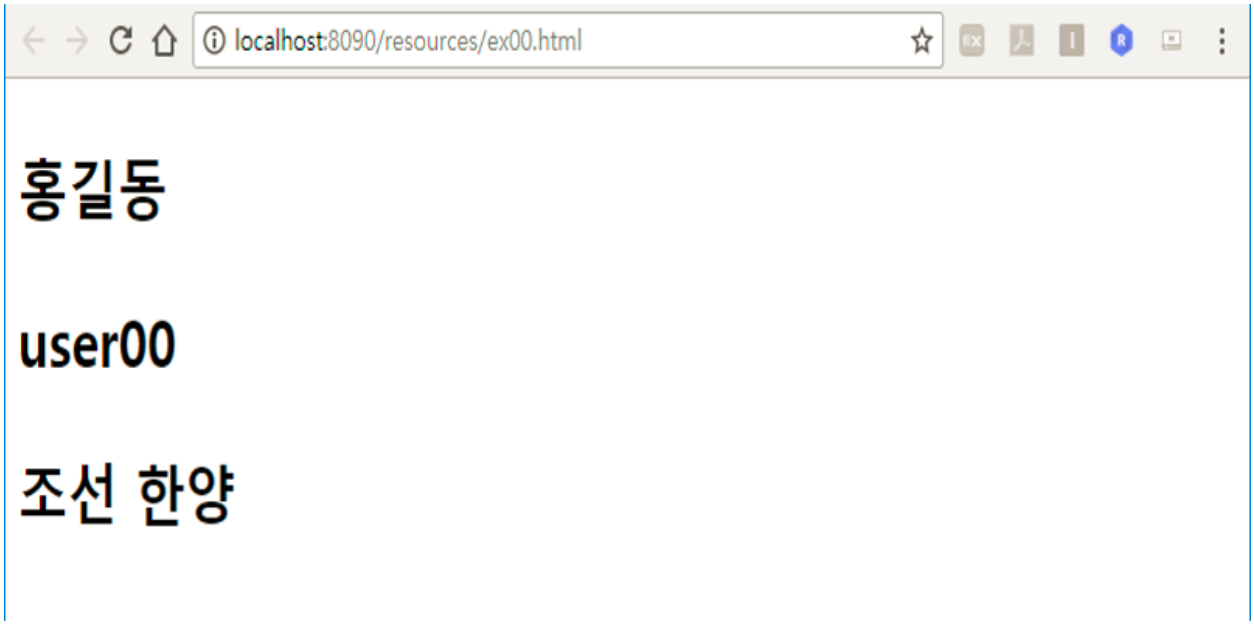
5.2.1 handlebars 사용 연습

- ex00.html

- handlebars를 사용하려면 반드시 jQuery 라이브러리가 필요하므로 먼저 jQuery 라이브러리를 로딩한 후 사용한다.
- script 태그에 정의된 var source는 적용될 HTML의 코드
- 다음 라인에서 이를 complie이라는 기능을 이용해서 미리 처리하고 이를 template으로 활용한다.
- handlebar의 경우 템플릿에서 대체되어야 하는 내용은 '{{}}' 문법을 이용해서 처리한다.
- Process
  - var data에는 실제 데이터가 구성
  - template(data)을 이용해서 처리하면 최종 HTML 코드가 완성된다.
  - 이를 $("#displayDiv")에 반영

```
<!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/3.0.1/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>
```

- 템플릿 결과



- ex01.html
  - id='template'은 HTML로 작성된 템플릿 코드

```
<!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/3.0.1/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>
```

- ex02.html
  - 배열로 된 데이터를 처리하는 방법
  - 템플릿의 선언부를 보면 handlebar의 '{{each.}}'를 이용
    - 'each'는 배열의 루프는 순환하는 것을 의미
    - '.'은 배열을 도는 동안의 해당 객체를 의미

```
<!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/3.0.1/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>
```

5.2 댓글 목록 처리

- handlebars를 사용해서 화면에 반복적으로 처리되는 템플릿 코드를 처리할 수 있다.
- 서버에서는 댓글의 목록과 댓글의 리스트 데이터를 한 번에 전송해 줄 수 있다.
  - 가장 먼저 처리하는 작업은 댓글을 페이징 처리해서 화면에 목록을 보여주는 작업을 한다.
- readPage.jsp
  - 템플릿 코드는 화면상에서 하나의 댓글을 구성하는 부분

```
<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 예로 사용되었다.
- 'prettifyDate'에 대한 JavaScript 처리

```
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);
}
```

- handlebars는 helper라는 기능을 이용해서 데이터의 상세한 처리에 필요한 기능들을 처리

- 만일 원하는 기능이 없는 경우에는 registerHelper()를 이용해서 사용자가 새로운 기능 추가
- readPage.jsp (paging)

```
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, targer){
    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);
}
```

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

5.3.1 댓글 목록의 이벤트 처리

- Replies List라는 버튼을 클릭했을 때 동작하도록 설계
- readPage.jsp
  - 목록의 size()를 체크하는 코드는 목록을 가져오는 버튼이 보여지는
  - 만 있는경우 1페이지의 댓글 목록을 가져오기 위한 코드

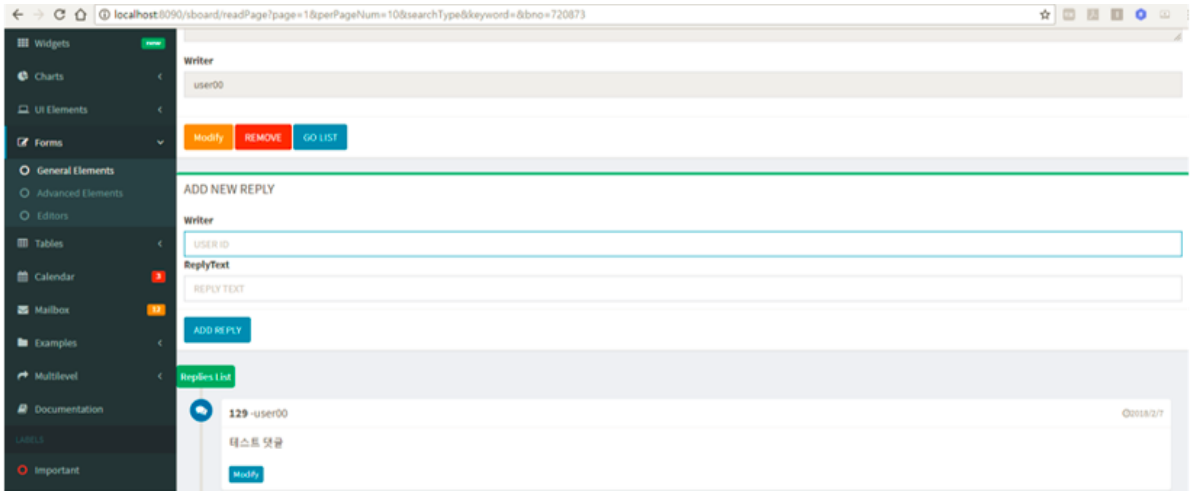
```
$("#repliesDiv").on("click", function() {
    if($(".timeline li").size() > 1){
        return;
    }
    getPage("/replies/" + bno + "/1");
});
```

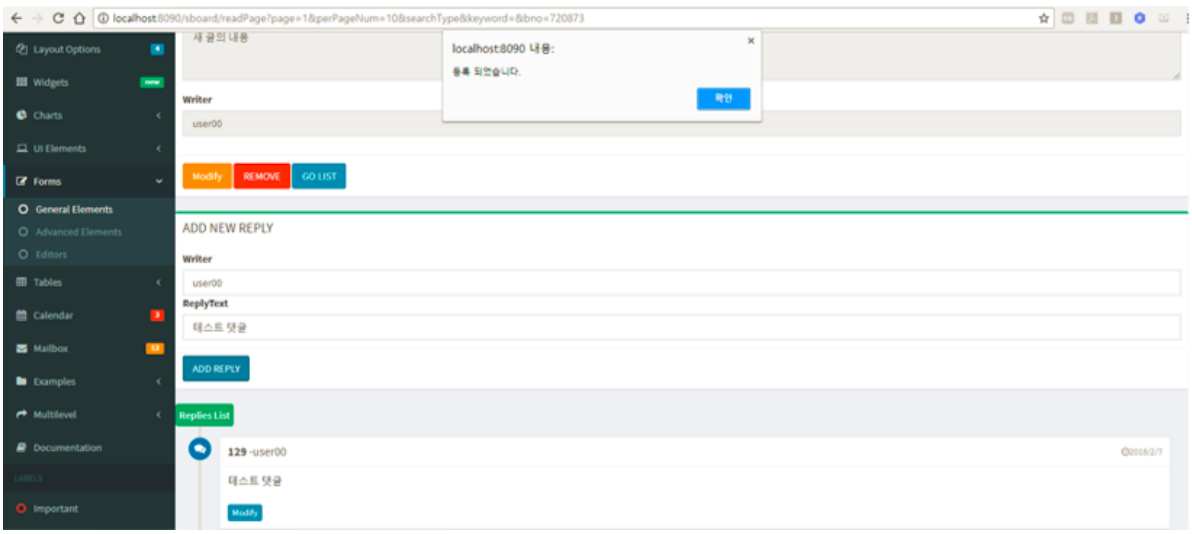
5.3.2 댓글 페이징의 이벤트 처리

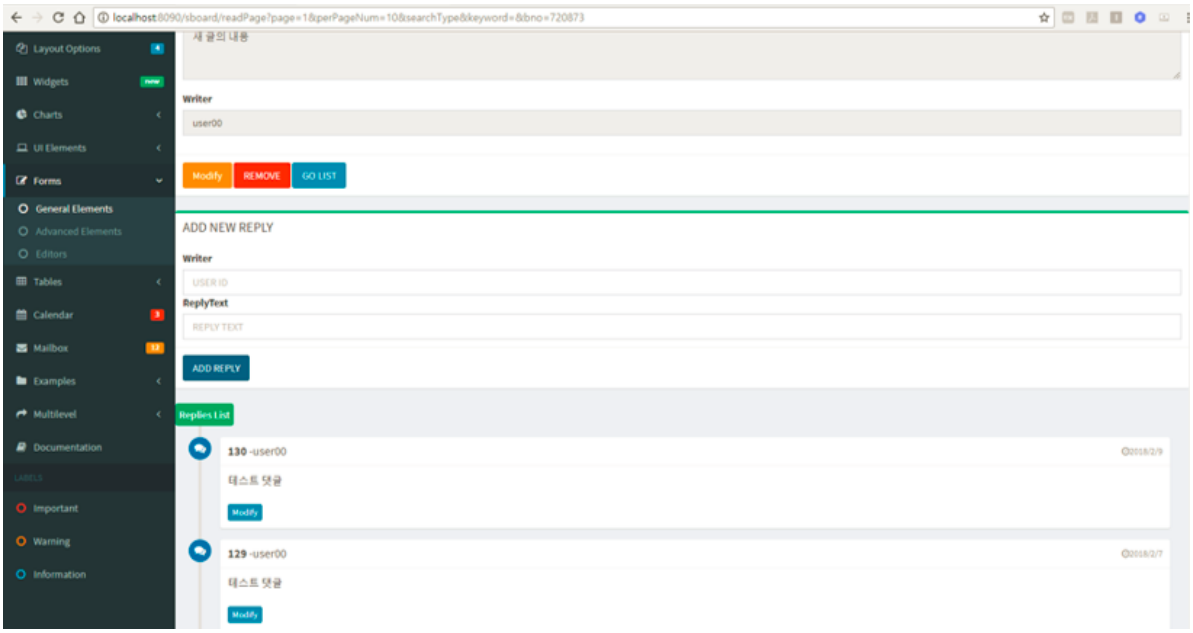
- 에서 페이징 처리가 이루어진다.
  - readPage.jsp (.pagination)

```
$(".pagination").on("cleck", "li a", function(event) {
    evnet.preventDefault();
    replyPage = $(this).attr("href");
    getPage("/replies/" + bno + "/" + replyPage);
});
```

5.4 새로운 댓글의 등록







5.4.1 댓글 등록의 이벤트 처리

- readPage.jsp (replyAddBtn)
  - JSON으로 전송하기 위해서 별도의 HTTP의 헤더 정보를 추가하고 전송한다.

```
$("#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("");
            }
        }
    });
});
```

5.5 수정과 삭제를 위한 Modal창

- 게시물의 수정과 삭제 작업은 별도의 Modal창을 이용해 처리
- readPage.jsp (modifyModal)

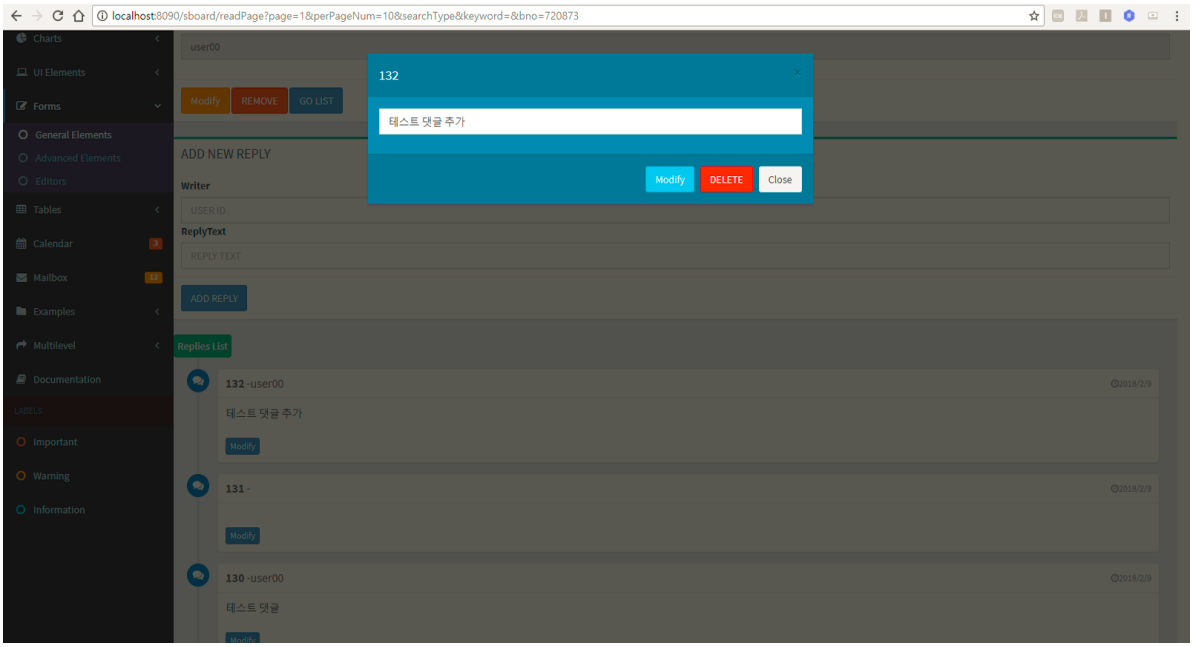
```
<!-- Modal -->
<div id="modifyModal" class="modal modal-primary fade" role="dialog">
    <div class="modal-dialog">
        <!-- Modal content -->
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal">&times;
</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>
        <!-- /.modal-content -->
    </div>
    <!-- /.modal-dialog -->
</div>
<!-- /.modal -->
```

5.5.1 각 댓글의 버튼 이벤트 처리

- readPage.jsp
    - Modal창
      의 내용물만을 처리하는 부분만 존재
    - 화면에 나타나게 하는 처리는 BootStrap이 제공하는 부분을 그대로 사용

```
$(".timeline").on("click", ".replyLi", function(event) {
    var reply = $(this);

    $("#replytext").val(reply.find(".timeline-body").text());
    $(".modal-title").html(reply.attr("data-rno"));
});
```

- readPage.jsp
  - 'data-'로 시작하는 커스텀 속성을 활용해서 modifyModal' 아이디에 속하는
    를 화면에 보이게 처리



5.6 수정과 삭제의 처리

- 수정 작업의 처리는 HTTP의 PUT 방식을 통해서 처리한다.
- readPage.jsp (replyModBtn)

```
$("#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);
        }
    });
});
```

- readPage.jsp (replyDelBtn)

```
$("#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/" + rno  + "/" + replyPage);
            }
        }
    });
});
```

5.7 정리

- Ajax의 처리는 서버와 화면의 작업을 철저히 구분
  - @RestController를 이용하는 컨트롤러의 작성과 테스트
  - Ajax를 테스트할 수 있는 환경을 만들고, 서버와의 통신을 확인
  - 화면에서 처리되어야 하는 DOM 구조와 이벤트에 대한 연습을 진행
  - 최종적으로 실제 결과물에 Ajax 처리 결과를 반영한다.