



[한국ICT인재개발원] 스프링 프레임워크

9. 스프링과 ajax(Oracle SQL)

前) 광고데이터 분석 1년

前) IT강의 경력 2년 6개월

前) 머신러닝을 활용한 데이터 분석 프로젝트반 운영 1년

前) 리그오브 레전드 데이터 분석 등...

現) 국비반 강의 진행중

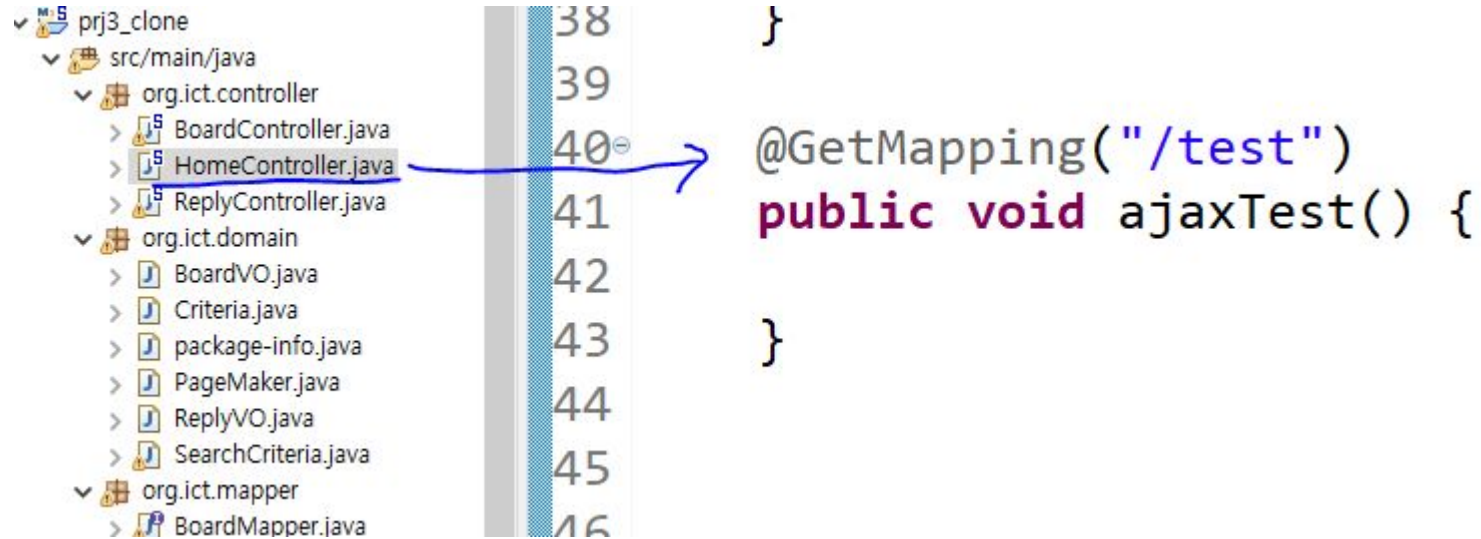
ajax는 비동기 통신을 의미합니다.

기존 방식 개발로는 일단 페이지 로딩이 완료된 다음 어떤 동작을 했을때 페이지의 이동을 전제로 개발이 이루어졌지만

비동기 방식 개발을 할 경우, 페이지의 이동이 아닌 해당 페이지 내의 html코드만 수정된다던지 하는 식으로 전체 페이지의 변경을 지양합니다.

이런 방식의 장점은 전체적인 페이지를 매번 그릴 필요가 없기 때문에

전반적인 서버 부담도 줄어 들고 사용자 입장에서 응답을 빨리 받을 수 있다는 장점을 가집니다.



먼저 개발시간의 단축을 위해 HomeController 하단에 ajaxText 메서드를 작성합니다.



```
<body>

    <h2>Ajax 테스트</h2>

    <ul id="replies">

    </ul>

    <!-- jquery는 이곳에 -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

</body>
</html>
```

다음 테스트를 할 **test.jsp**를 생성합니다.

views 바로 아래에 생성하면 되고, **body**태그는 다음과 같이 구성합니다.

```
49 • select * from ictreply;
50 • INSERT INTO ictreply(bno, replytext, replyer)
```

Result Grid | Filter Rows: | Edit: | Export/Import:

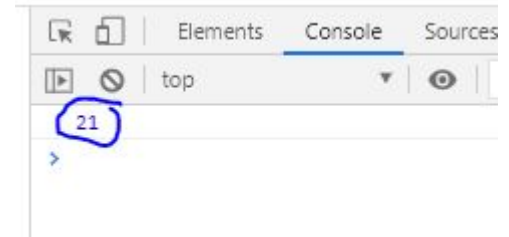
rno	bno	replytext	replyer	regdate	updatedate
1	1	댓글	쓴자	2021-03-25 22:29:56	2021-03-25 22:29:56
2	1	댓글	쓴자	2021-03-25 22:30:27	2021-03-25 22:30:27
3	1	댓글	쓴자	2021-03-25 22:30:41	2021-03-25 22:30:41
4	1	댓글	쓴자	2021-03-25 22:30:43	2021-03-25 22:30:43
5	1	댓글	쓴자	2021-03-25 22:32:24	2021-03-25 22:32:24
6	1	댓글	쓴자	2021-03-25 22:32:56	2021-03-25 22:32:56
7	1	댓글	쓴자	2021-03-25 22:33:15	2021-03-25 22:33:15
8	1	댓글	쓴자	2021-03-25 22:34:09	2021-03-25 22:34:09
9	1	댓글	쓴자	2021-03-25 22:34:14	2021-03-25 22:34:14
10	1	댓글	쓴자	2021-03-25 22:34:16	2021-03-25 22:34:16
11	1	글인텐	글쓴...	2021-03-25 22:36:12	2021-03-25 22:36:12
12	1	글인텐	글쓴...	2021-03-25 22:36:12	2021-03-25 22:36:12
13	1	글인텐	글쓴...	2021-03-25 22:36:13	2021-03-25 22:36:13

DB를 조회해 댓글이 2개 이상인 글번호(bno)를 미리 기억해두세요.

없다면 insert구문을 실행해 댓글을 2개 이상 보유한 글을 만들어줍니다.

```
<script type="text/javascript">
    var bno = 1;

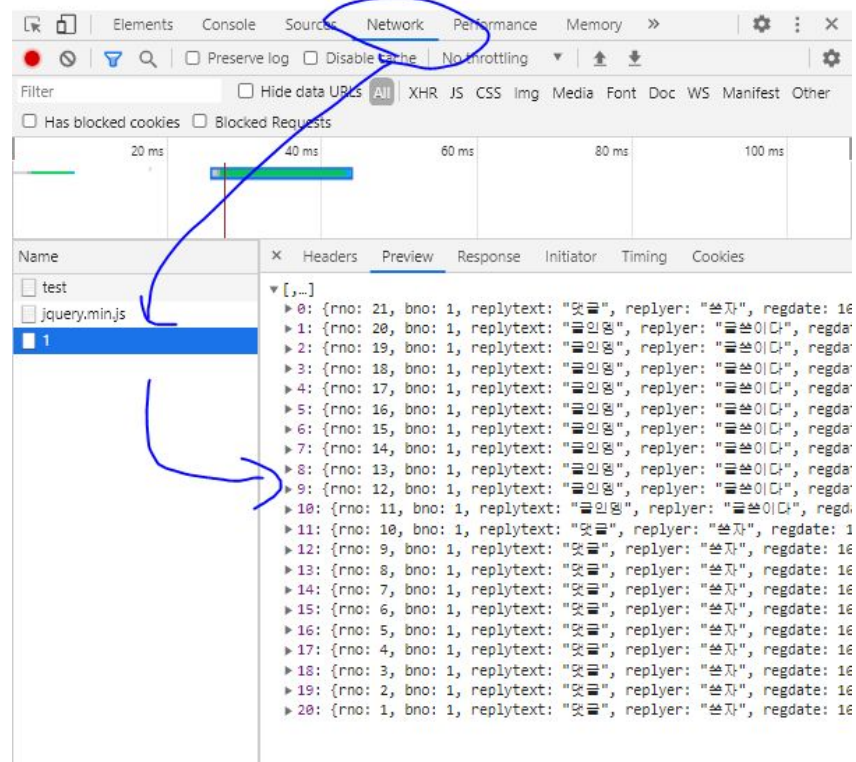
    $.getJSON("/replies/all/" + bno, function(data){
        console.log(data.length);
    });
</script>
```



첫 번째로는 페이지를 열자마자 해당 글(교안 기준 1번 bno)의 글 개수를 콘솔창에 찍어주는 자바스크립트 코드를 작성해보겠습니다.

```
$.getJSON("주소", function(data) {
    console.log(data.length);
});
```

는 rest 주소에 데이터를 요청하고 받아온 데이터를 data변수에 담아줍니다.



또한 Network 탭을 열어보면 전달받은 데이터의 내용을 추가로 볼 수 있습니다.


```
<script>
$.getJSON("/replies/all/" + bno, function(data){
    //data 변수가 바로 얻어온 json데이터의 집합
    console.log(data.length);

    // str 변수 내부에 문자 형태로 html 코드를 작성함
    var str = "";

    $(data).each(function(){
        // $(data).each()는 향상된 for문처럼 내부데이터를 하나하나 반복합니다.
        // 내부 this는 댓글 하나하나입니다.
        str += "<li data-rno='" + this.rno + "' class='replyLi'"
            + this.rno + ":" + this.reply
            + "<button>수정/삭제</button></li>";
    });

    // #replies인 ul태그 내부에 str을 끼워넣음
    $("#replies").html(str);
});
}</script>
```

str 내부에 로 구성된 태그를 전체 댓글 개수만큼 반복해서 작성해두도록 .each구문을 사용하고, 그에 다른 결과는 아까 생성해놨던 내부에 삽입하도록 처리했습니다.

```

function getAllList(){
    $.getJSON("/replies/all/" + bno, function(data){
        //data 변수가 바로 얻어온 json데이터의 집합
        console.log(data.length);

        // str 변수 내부에 문자 형태로 html 코드를 작성함
        var str = "";

        $(data).each(function(){
            // $(data).each()는 향상된 for문처럼 내부데이터를 하나하나 반복합니다.
            // 내부 this는 댓글 하나하나입니다.
            str += "<li data-rno='" + this.rno + "' class='replyLi'"
                + "this.rno + ":" + this.reply
                + "<button>수정/삭제</button></li>";
        });

        // #replies인 ul태그 내부에 str을 끼워넣음
        $("#replies").html(str);
    });
}

getAllList();
    
```

선언

호출

이 전체 댓글 불러오는 기능은 많이 사용하므로 함수화 해서 저장해둡니다.
마지막에 한 번 호출해서 제대로 작동하는지만 체크하고 넘어갑니다.

```
</head>
<body>

  <h2>Ajax 테스트</h2>

  <div>
    <div>
      REPLYER <input type="text" name="replyer" id="newReplyWriter">
    </div>
    <div>
      REPLY <input type="text" name="reply" id="newReply">
    </div>
    <button id="replyAddBtn">리플추가</button>
  </div>

  <!-- <ul id="replies">

  </ul> -->
```

이번엔 댓글 등록 화면을 테스트해보겠습니다.
test.jsp의 은 잠깐 주석처리하고, body태그 아래를 위와 같이 수정합니다.

Ajax 테스트

REPLYER

REPLY TEXT

```
// 비동기 코드
$("#replyAddBtn").on("click", function(){
    // 각 input태그에 들어있던 글쓴이, 본문의 value값을 변수에 저장함.
    var replyer = $("#newReplyWriter").val();
    var reply = $("#newReply").val();
    // 디버깅시는 console.log() 내부에 적어서 확인합니다.
    //console.log(replyer + "/" + reply);
    $.ajax({
        type : 'post',
        url : '/replies/',
        headers: {
            "Content-Type" : "application/json",
            "X-HTTP-Method-Override" : "POST"
        },
        dataType : 'text',
        data : JSON.stringify({
            bno:bno,
            replyer:replyer,
            reply:reply
        }),
        success : function(result){
            if(result == 'SUCCESS'){
                alert("등록 되었습니다.");
            }
        }
    });
});
```

우선, 위와 같은 Ajax 테스트 라는 창이 나오게 되는데
먼저 우측과 같은 형태로 작성하면, 데이터가 들어갑니다. 댓글이 들어가는지
확인부터 해 보고 다음 단계를 설명해드리겠습니다.

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

먼저 위와 같이 \$.ajax() 메서드를 호출해서 구성하는것이 일반적입니다.
\$.ajax() 사이에 { }로 감싼 json 형태를 전달하는것이 그 시작입니다.

```
type : 'post',  
url : '/replies',  
headers: {  
    "Content-Type" : "application/json",  
    "X-HTTP-Method-Override" : "POST"  
},  
data : { ... }
```

감싸진 부분의

`type:'post'`는 말 그대로 `post`방식 호출을 하겠다는 의미이고

`url : '/replies'`는 해당 주소로 요청을 하겠다는 것입니다.

`headers : { ... }`는 혹시 해당 `restcontroller`가 `json`만을 허용하는 경우 이로직이 `json`데이터를 보낼을 명시합니다.


```
dataType : 'text',
data : JSON.stringify({
    bno:bno,
    replyer:replyer,
    reply:reply
}),
```

```
var bno = 1;
```

```
// 각 input태그에 들어있던 글쓴이, 본문의 value값을 변수에 저장함.
var replyer = $("#newReplyWriter").val();
var reply = $("#newReply").val();
```

위의 **dataType**은 'text'이며, 비동기식 전송을 할 때는 text형식이지만 파싱하면 json으로 인식할 수 있는 문자열을 보내는것이 일반적입니다.

하단 **data**는 보낼 데이터이며, **JSON.stringify()** 내부에 {...}을 입력해서 위에서 선언되어 얻어온 **bno**, **replyer**, **replytext**들을 json으로 먼저 만든 다음 문자열로 변환해 날려줍니다.

```
success : function(result) {  
    if(result == 'SUCCESS'){  
        alert("등록 되었습니다.");  
    }  
}
```

success : 함수

를 보내는 것이 일반적이며, **result**는 **ResponseEntity**에서 리턴해준 문자열을 받게 됩니다.

success에 해당하는 함수는 요청이 성공했을때 실행합니다.


```
$.getJSON("/replies/all/" + bno, function(data){
    //data 변수가 바로 얻어온 json데이터의 집합
    console.log(data.length);

    // str 변수 내부에 문자 형태로 html 코드를 작성함
    var str = "";

    $(data).each(function(){
        // $(data).each()는 향상된 for문처럼 내부데이터를 하나하나 반복합니다
        // 내부 this는 댓글 하나하나입니다.
        str += "<li data-rno='" + this.rno + "' class='replyLi'"
            + "this.rno + ":" + this.reply
            + "<button>수정/삭제</button></li>";
    });
});
```

```
$.ajax({
    type : 'post',
    url : '/replies',
    headers: {
        "Content-Type" : "applicat
        "X-HTTP-Method-Override" :
    },
    dataType : 'text',
    data : JSON.stringify({
```

즉, \$.getJSON() 은 이 페이지에서 데이터를 서버에 요청해 가지고 올때,
\$.ajax()는 이 페이지에 입력된 어떤 자료를 서버에 보낼때 사용하는
기능들입니다.

이상의 로직들은 페이지 이동 없이 이루어졌습니다.

이런 식으로 rest서버에 ajax요청을 해서 비동기 작업을 하는것이 바로 rest개발
방식이며, 어플리케이션 등에서도 비슷한 수순으로 개발이 이루어집니다.

```
success : function(result){  
    if(result == 'SUCCESS'){  
        alert("등록 되었습니다.");  
        // 댓글 쓰고 나서 다시 새롭게 갱신된 목록을  
        // 넣어주도록 전체 댓글 목록 다시 조회  
        getAllList();  
    }  
}
```

추가

그리고 댓글은 쓰자마자 갱신되어야 하기 때문에, 댓글을 등록했을 때 글 등록 팝업창이 뜬 뒤에 바로 전체 댓글목록을 가져올 수 있도록 `getAllList()`를 호출하는 구문을 추가합니다.

```
$.getJSON("/replies/all/" + bno, function(data){
    //data 변수가 바로 얻어온 json데이터의 집합
    console.log(data.length);

    // str 변수 내부에 문자 형태로 html 코드를 작성함
    var str = "";

    $(data).each(function(){
        // $(data).each()는 향상된 for문처럼 내부데이터를 하나하나 반복합니다.
        // 내부 this는 댓글 하나하나입니다.
        str += "<li data-rno='" + this.rno + "' class='replyLi'"
            + "this.rno + "':'" + this.reply
            + "<button>수정/삭제</button></li>";
    });
```

추가

이제 모든 댓글은 표출시 버튼을 가질 수 있도록 위와 같이 수정합니다.

- 13:AJAX 넘모 쉬워영 수정/삭제
- 10:AJAX 개꿀잼이네 수정/삭제
- 9:원리를 정확하게 알아쏘요 수정/삭제
- 7:코사인엑스 수정/삭제
- 6:대충 본문 수정/삭제
- 5:댓글시험 수정/삭제
- 4:댓글시험 수정/삭제
- 3:댓글시험 수정/삭제

수정된 결과가 위처럼 나온다면 성공입니다.

```
// 이벤트 위임
// 내가 현재 이벤트를 걸려는 집단(button) 을 포함하면서 범위가 제일 좁은
// #replies로 시작조건을 잡습니다.
// .on("click", "목적지 태그까지 요소들", function(){실행문})
// 과 같이 위임시는 파라미터가 3개 들어갑니다.
$("#replies").on("click", ".replyLi button", function(){
    // this는 최 하위태그인 button, button의부모면 결국 .replyLi
    var replyLi = $(this).parent();

    // .attr("속성명") 을 하면 해당 속성의 값을 얻습니다.
    var rno = replyLi.attr("data-rno");// data-rno=this.rno 얻어오기
    var reply = replyLi.text(); // li태그 글씨만 얻기.

    // 클릭한 버튼에 해당하는 댓글번호 + 본문이 얻어지나 디버깅
    console.log(rno + ":" + reply);
});
```

다음으로 각 이벤트 버튼 모두가 독립적으로 클릭시 반응하도록 “위임”처리를 했습니다.

위와같이 onclick 이벤트를 처리할 때 두 번째 파라미터로 함수 대신 “부모요소 자식요소” 와 같이 적으면, 실제로는 replyLi 하위의 button 모두가 따로따로 기능합니다.

```
<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>
```

이제 수정 및 삭제 처리를 위한 창을 만들겠습니다.

이 창은 modal을 변형해서 만들며, html요소에는 존재하지만 평소에는 보이지 않다가, 추후에 버튼을 클릭했을때만 화면에 표출되는 창으로 설계합니다.


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

상단 head태그 내에 style부분도 작성합니다.

```
// 이벤트 위임
// 내가 현재 이벤트를 걸려는 집단(button) 을 포함하면서 범위가 제일 좁은
// #replies로 시작조건을 잡습니다.
// .on("click", "목적지 태그까지 요소들", function(){실행문})
// 과 같이 위임시는 파라미터가 3개 들어갑니다.
$("#replies").on("click", ".replyLi button", function(){
    // this는 최 하위태그인 button, button의부모면 결국 .replyLi
    var replyLi = $(this).parent();

    // .attr("속성명") 을 하면 해당 속성의 값을 얻습니다.
    var rno = replyLi.attr("data-rno");// data-rno=this.rno 얻어오기
    var reply = replyLi.text(); // li태그 글씨만 얻기.

    // 클릭한 버튼에 해당하는 댓글번호 + 본문이 얻어나 디버깅
    console.log(rno + ":" + reply);

    // 모달 열리도록 추가
    $(".modal-title").html(rno);// 모달 상단에 댓글번호 넣기
    $("#replytext").val(reply);// 모달 수정창에 댓글본문 넣기
    $("#modDiv").show("slow"); // 창에 애니메이션 효과 넣기
});
```

이벤트 위임은 이제 alert창 대신 모달이 직접 열리도록 아래 3줄을 우측과 같이 변경해주시면 됩니다.

모달의 경우, jquery로 모달창을 설정한 다음, .show()를 사용하면 드러나도록 설계되어있습니다.

“slow”를 붙여서 천천히 열리는 애니메이션 효과를 추가합니다.

• 13:글인덱	수정/삭제
• 12:글인덱	수정/삭제
• 11:글인덱	수정/삭제
• 10:댓글	수정/삭제
• 9:댓글	수정/삭제
• 8:댓글	수정/삭제
• 7:댓글	수정/삭제
• 6:댓글	수정/삭제
• 5:댓글	수정/삭제
• 4:댓글	수정/삭제
• 3:댓글	수정/삭제
• 2:댓글	수정/삭제
• 1:댓글	수정/삭제

이름 표시



수정/삭제 버튼을 클릭했을때 가운데에 위와 같이 수정창이 드러나면 성공입니다.

```
$("#replyDelBtn").on("click", function(){
    //삭제에 필요한 댓글번호 모달 타이틀 부분에서 얻기
    var rno = $(".modal-title").html();

    $.ajax({
        type : 'delete',
        url : '/replies/' + rno,
        // 전달 데이터가 없이 url과 호출타입만으로 삭제처리하므로
        // 이외 정보는 제공할 필요가 없음
        success : function(result){
            if(result == 'SUCCESS'){
                alert(rno + "번 댓글이 삭제되었습니다.");
                // 댓글 삭제 후 모달창 닫고 새 댓글목록 갱신
                $("#modDiv").hide("slow");
                getAllList();
            }
        }
    });
});
```

다음 삭제로직 전체입니다.

modal-title의 글 번호를 얻어오고, replytext의 수정할 글 내용을 가져온 다음 ajax통신을 이용해 delete로직을 실행한 다음 모달을 닫고 댓글목록을 갱신합니다.

```

$("#replyModBtn").on("click", function(){
    //수정에 필요한 댓글번호 모달 타이틀 부분에서 얻기
    var rno = $(".modal-title").html();
    //수정에 필요한 본문내용을 #reply의 value값으로 얻기
    var reply = $("#reply").val();

    $.ajax({
        type : 'patch',
        url : '/replies/' + rno,
        header : {
            "Content-Type" : "application/json",
            "X-HTTP-Method-Override" : "PATCH"
        },
        dataType : 'text',
        data : JSON.stringify({reply:reply}),
        success : function(result){
            if(result == 'SUCCESS'){
                alert(rno + "번 댓글이 수정되었습니다.");
                // 댓글 삭제 후 모달창 닫고 새 댓글목록 갱신
                $("#modDiv").hide("slow");
                getAllList();
            }
        }
    });
});

```

수정로직은 위와 같이 작성합니다.

전반적으로는 delete와 같고, type만 put(patch)로 바꿉니다.

단, 수정은 삭제와 달리 수정할 내용이 동반되므로 data속성도 추가해주세요.
contentType은 만약 415에러가 뜬다면 추가하면 되는 부분입니다.

```
var bno = ${board.bno};

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

        console.log(data.length);

        var str = "";

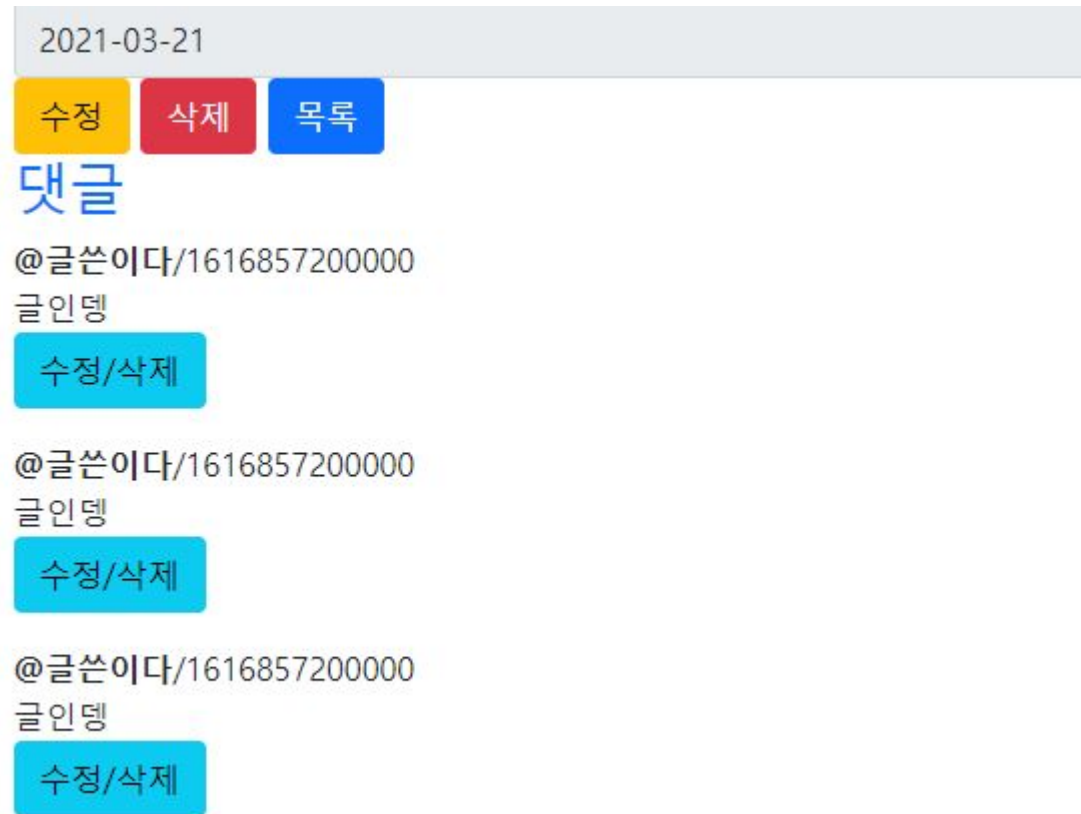
        $(data).each(function() {
```

이제 get.jsp로 돌아와 글 번호를 받도록 var bno에 처리해주시고,
다음 test.jsp에 설정해둔 getAllList()를 복사해옵니다.

```
$(data).each(function() {  
    var timestamp = this.updatedate;  
    var date = new Date(timestamp);  
  
    var formattedTime = "게시일 : " + date.getFullYear()  
        + "/" + (date.getMonth()+1)  
        + "/" + date.getDate()  
  
    str += "<div class='replyLi' data-rno='" + this.rno + "'><strong>@"  
        + this.replyer + "</strong> - " + formattedTime + "<br>"  
        + "<div class='replytext'>" + this.replytext + "</div>"  
        + "<button type='button' class='btn btn-info'>수정/삭제</button>"  
        + "</div>";  
});
```

```
<div class="row">  
    <h3 class="text-primary">댓글</h3>  
    <div id="replies">  
        <!-- 댓글이 들어갈 위치 -->  
    </div>  
</div><!-- row -->  
  
liv><!-- container -->
```

다음, 댓글 출력 로직은 마음대로 바꿔주시면 되지만, 우선 예시로 간단하게 댓글임을 식별할 수 있으며, 수정/삭제버튼이 존재하는 형태로 만들었습니다.



위 코드를 실행하면 댓글이 이렇게 표출됩니다.

단, 날짜를 아직은 timestamp타입으로 출력중인데, 이는 date형식으로 바꿔주는 추가 작업을 해야합니다.

```
$(data).each(function() {  
    var timestamp = this.updatedate;  
    var date = new Date(timestamp);  
  
    var formattedTime = "게시일 : "+ date.getFullYear()  
        + "/" + (date.getMonth()+1)  
        + "/" + date.getDate()  
  
    str += "<p class='replyLi'><strong>@" + this.replyer + "</strong> - " + formattedTime  
        + this.replytext + "<br>"
```

구문 내에 먼저 `updatedate`를 받아온 다음,
`Date`타입으로 변경을 해 줍니다.

그 뒤 `getFullYear()`로 연도를, `getMonth`로 월을(단, 0부터 시작하니 주의)
`getDate`로 일자를 받아와 출력위치에 둡니다.

댓글

@처리하자 - 게시일 : 2021/3/28
시간을

수정/삭제

@처리하자 - 게시일 : 2021/3/28
시간을

수정/삭제

@처리하자 - 게시일 : 2021/3/28
시간을

수정/삭제

그러면 위와 같이 게시시간이 우리가 사용하는 시간형태로 변합니다.


```
<div class="row box-box-success">
  <div class="box-header">
    <h2 class="text-primary">댓글 작성</h2>
  </div><!-- header -->
  <div class="box-body">
    <strong>Writer</strong>
    <input type="text" id="newReplyer" placeholder="Replyer" class="form-control">
    <strong>ReplyText</strong>
    <input type="text" id="newReplyText" placeholder="ReplyText" class="form-control">
  </div><!-- body -->
  <div class="box-footer">
    <button type="button" class="btn btn-success" id="replyAddBtn">Add Reply</button>
  </div><!-- footer -->
</div><!-- row -->
```

다음으로 댓글 등록창을 만들어보겠습니다.

위치는 댓글목록 아래에 두겠습니다.

역시 간단하게 input태그와 버튼으로 이벤트를 처리합니다.

@처리하자 - 게시일 : 2021/3/28

시간을

수정/삭제

댓글 작성

Writer

Replyer

ReplyText

ReplyText

Add Reply

위와 같이 댓글 작성창이 아래에 추가됩니다.

```
$.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.");
      $("#newReplyer").val("");
      $("#newReplyText").val("");
      location.href="/board/get?bno=" + bno
        + "&page=" + "${cri.page }"
        + "&searchType=" + "${cri.searchType}"
        + "&keyword=" + "${cri.keyword}";
    }
  }
})
```

이벤트 역시 test.jsp에서 가져오되, 성공시 input 태그 내부를 비우는 로직만 추가합니다. 마지막에 요청한 페이지로 돌아오는 부분도 추가합니다.

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

```
<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>
```

test.jsp에서 style태그와 모달 양식을 가져와 get.jsp에 작성합니다.

```
//이벤트 위임
$("#replies").on("click", ".replyLi button", function(){
    var reply = $(this).parent();
    var rno = reply.data("rno");
    var replytext = reply.children('.replytext').html();

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

위임 양식은 잘 보셔야 합니다.

먼저 수정/삭제 버튼을 위임기능을 이용해 각각 작동하도록 처리합니다.

다음, 버튼의 부모인 replyLi를 reply변수가 가리키도록 저장하고

reply 내부의 자식인 .replytext내부의 html을 가져와서

각각 필요한 곳에 삽입한 후 모달 전체 창을 보여주도록 처리합니다.


```
//삭제버튼 작동
$("#replyDelBtn").on("click", function(){
    var rno = $(".modal-title").html();
    var replytext = $("#replytext").val();

    $.ajax({
        type : 'delete',
        url : '/replies/' + rno,
        header : {
            "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();
            }
        }
    });
});
```

삭제로직도 test.jsp에 이미 완성되어있으므로 그대로 가져옵니다.

```
//수정버튼 작동
$("#replyModBtn").on("click", function(){
    var rno = $(".modal-title").html();
    var replytext = $("#replytext").val();

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

수정로직도 test.jsp에 이미 완성되어있으므로 그대로 가져옵니다.

```
//모달 닫기는 모달의 .hide() 기능을 이용합니다.  
$("#closeBtn").on("click", function(){  
    $("#modDiv").hide("slow");  
})
```

마지막으로 모달 닫기를 처리합니다.
.hide()로 닫고, .show()로 여는 부분을 기억해주세요.

ajax에서도 페이지네이션은 **Criteria** 객체와 **PageMaker** 객체의 도움을 받습니다.

이를 위해, 먼저 댓글목록도 전체를 가져오는게 아니라 페이징 처리를 해서 가져오도록 **limit** 구문을 적용시키고

댓글 갯수를 알아야 아래쪽에 페이지네이션을 걸 수 있기 때문에

replyCount메서드를 활용해 전체 댓글 개수를 알아오는것이 먼저입니다.

```
public void delete(Long bno);
```

```
public List<ReplyVO> getListPage(@Param("bno") int bno, @Param("cri") Criteria cri);
```

```
public int count(int bno);
```

```
<select id="getListPage" resultType="org.ict.domain.ReplyVO">
    SELECT * FROM ictreply
        WHERE bno = #{bno}
        ORDER BY rno DESC
        limit #{cri.pageStart}, #{cri.number}
</select>
```

```
<select id="count" resultType="int">
    SELECT count(bno) from ictreply
        WHERE bno = #{bno}
</select>
```

그래서 ReplyMapper인터페이스와 매퍼xml에 위와 같이 메서드를 선언 및 구현해줍니다.

```
public List<ReplyVO> getListPage(int bno, Criteria cri);
```

```
public int count(int bno);
```

```
@Override
```

```
public List<ReplyVO> getListPage(int bno, Criteria cri) {
```

```
    return mapper.getListPage(bno, cri);
```

```
}
```

```
@Override
```

```
public int count(int bno) {
```

```
    return mapper.count(bno);
```

```
}
```

그리고 ReplyService쪽 역시 전부 처리합니다.

```
private int endPage;  
private boolean prev;  
private boolean next;  
private int totalReply;
```

```
// 페이지당 버튼을 몇 개씩 생성할지  
private int displayPageNum;
```



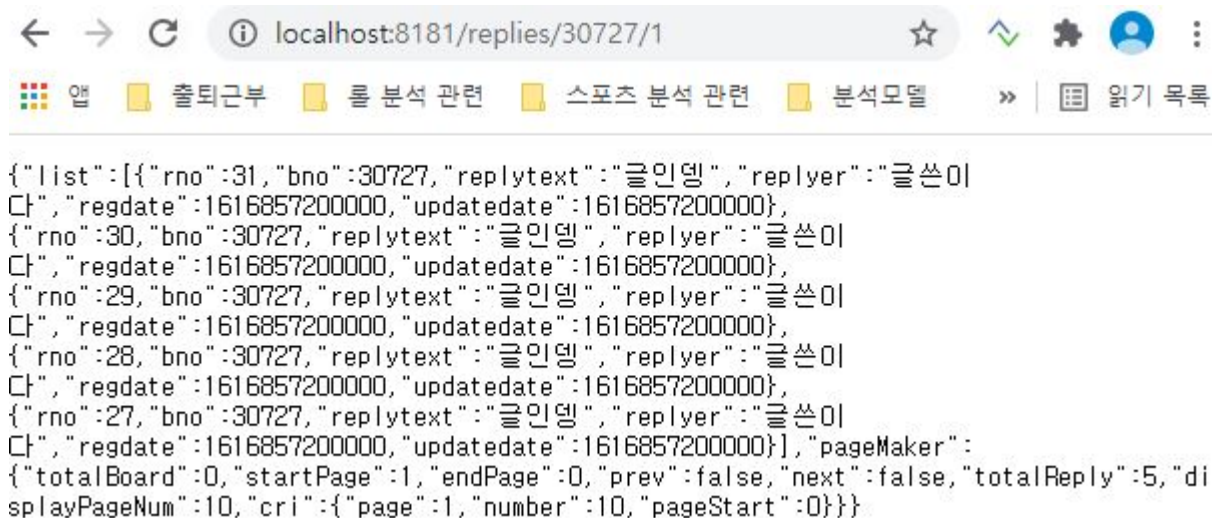
```
// 페이지가 바뀔 때마다 버튼 갯수 및 범위를 산출하는 메서드
```

```
public void calcData2() {  
    this.displayPageNum = 10;  
  
    this.endPage = (int)(Math.ceil(cri.getPage() /  
        (double) displayPageNum) * displayPageNum);  
  
    this.startPage = (endPage - displayPageNum) + 1;  
  
    int tempEndPage = (int)(Math.ceil(totalReply /  
        (double)cri.getNumber()));  
    if(endPage > tempEndPage) {  
        endPage = tempEndPage;  
    }  
  
    prev = startPage == 1 ? false : true;  
  
    next = endPage * cri.getNumber() >=  
        totalReply ? false : true;  
}
```



```
public void setTotalReply(int totalReply) {  
    this.totalReply = totalReply;  
  
    calcData2();  
}
```

PageMaker내부에는 setTotalReply메서드를 만듭니다.
setTotalBoard메서드와 동일하게 만들어주시면 됩니다.
추가로 calcData2를 생성해서 totalReply를 책임지도록 해 주세요.



```
{
  "list": [
    {
      "rno": 31,
      "bno": 30727,
      "replytext": "글인영",
      "replyer": "글쓴이",
      "regdate": 1616857200000,
      "updatedate": 1616857200000
    },
    {
      "rno": 30,
      "bno": 30727,
      "replytext": "글인영",
      "replyer": "글쓴이",
      "regdate": 1616857200000,
      "updatedate": 1616857200000
    },
    {
      "rno": 29,
      "bno": 30727,
      "replytext": "글인영",
      "replyer": "글쓴이",
      "regdate": 1616857200000,
      "updatedate": 1616857200000
    },
    {
      "rno": 28,
      "bno": 30727,
      "replytext": "글인영",
      "replyer": "글쓴이",
      "regdate": 1616857200000,
      "updatedate": 1616857200000
    },
    {
      "rno": 27,
      "bno": 30727,
      "replytext": "글인영",
      "replyer": "글쓴이",
      "regdate": 1616857200000,
      "updatedate": 1616857200000
    }
  ],
  "pageMaker": {
    "totalBoard": 0,
    "startPage": 1,
    "endPage": 0,
    "prev": false,
    "next": false,
    "totalReply": 5,
    "displayPageNum": 10,
    "cri": {
      "page": 1,
      "number": 10,
      "pageStart": 0
    }
  }
}
```

이제 패턴대로 접속을 해 보겠습니다.

해당 글의 번호와 페이지 번호를 입력했을때 위와같이 계산되어 나오면 됩니다.

Ajax 테스트

REPLYER

REPLY TEXT

- 25:비동기 너무좋아
- 21:댓글
- 20:글인덱
- 19:글인덱
- 18:글인덱
- 17:글인덱
- 16:글인덱

다음은 댓글 출력입니다. 예전에 사용했던 ajax 테스트를 그대로 쓰겠습니다.

```
        <button id="replyAddBtn">ADD Reply</button>
    </div>
    <ul id="replies">
    </ul>
    <ul class='pagination'>
    </ul>
    <button id="addReplyList">리플  표출</button>
```

test.jsp의 리플 표출 ul 태그 하단에 pagination ul을 하나 더 만듭니다.


```
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);  
    });  
} //getPageList
```

다음으로 페이지네이션 정보까지 같이 가져오는 `getPageList` 함수를 정의합니다. 이 함수는 특정 페이지 댓글 10개만 가져올 뿐 하단에 페이징 버튼을 만들지는 않습니다.

```
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>";
    }
    $('<div>.pagination').html(str);
} //printPaging
```

따라서 하단에페이징버튼을 만들어주는 printPaging 함수를 추가로 선언합니다.

```
        $("#replies").html(str);  
        printPaging(data.pageMaker);  
    });  
} //getPageList
```

이 함수는 10개의 댓글 아래부분에 호출되어야 하기 때문에
getPageList 마지막에 호출하도록 코드를 수정합니다.

```
$(".pagination").on("click", "li a", function(e){  
    e.preventDefault();  
  
    replyPage = $(this).attr("href");  
  
    getPageList(replyPage);  
});
```

마지막으로, 현재 페이지에서 페이지징 버튼을 눌러도 a태그가 발동해 페이지가 넘어가지 않는 문제가 있기 때문에, 이를 해결하기 위해 페이지징 번호만 얻어와서 재호출하도록 수정해주세요.