

# Part 3 Ajax 댓글 처리

## Chapter 04 화면에서의 Ajax 호출

- @RestController를 이용하는 경우에는 데이터만 만들어져 전송되기 때문에, 사실상 개발의 대부분은 이를 처리하는 화면으로 집중된다.
- 앱에서는 HTTP 방식으로 데이터를 주고받을 수 있는 라이브러리 등을 활용해서 데이터를 처리
- 웹에서는 주로 Ajax를 이용해서 처리한다.

### 4.1 개발의 순서 결정

- Ajax를 이용하는 댓글 처리는 개발 대부분이 화면에서의 처리가 많으므로, 간단한 페이지를 만들어 결과 데이터의 처리를 먼저 연습한다.
- 작업 순서
  - 특정 게시물을 미리 지정, 전체 댓글의 목록을 가져와 출력하는 기능 작성
  - 댓글을 입력할 수 있는 화면을 구성, 이를 이용해서 새로운 댓글 추가
    - 전체 목록을 우선 구현하는 이유는 새로운 댓글이 추가된 후 자동으로 전체 댓글을 갱신하도록 작성해서 볼 수 있게 하기 위함
  - 댓글의 목록에서 특정 게시물을 선택해서 수정과 삭제 작업이 이루어지는 화면을 구성
  - 댓글의 삭제 작업을 진행하고, 등록과 마찬가지로 전체 목록을 갱신해서 댓글이 사라지는 결과를 확인할 수 있게 한다.
  - 댓글의 수정 작업을 진행하고, 목록이 갱신되어 결과를 볼 수 있게 한다.
  - 모든 작업이 완료되었으므로, 댓글에 대한 페이지징 처리를 진행한다.

#### 4.1.1 테스트를 위한 컨트롤러와 JSP

- JSP를 구성하기 위해서 컨트롤러에 새로운 URI를 등록해야 한다.
- HomeController를 이용해 URI를 처리할 수 있도록 메서드를 추가
- HomeController

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>

    </head>

    <body>

        <h2>Ajax Test Page</h2>
        <ul id="replies">
        </ul>
```

```

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

</body>
</html>

```

- HTML 상에는 댓글의 목록을 처리하는  
을 작성, 이를 위하여 댓글을 보여준다.

## 4.2 전체 댓글 목록의 테스트

- JavaScript의 경우 별도 파일로 처리하는 것이 좋지만, 테스트 화면에서는 직접 작성
- tbl\_reply에 있는 특정한 게시물의 번호를 결정

sdbs.tbl\_reply: 32 행 (총) (대략적)

rno	bnr	replytext	replyer	regdate	updatedate
60	720,873	댓글	user00	2018-02-04 04:10:59	2018-02-04 04:10:59
61	720,873	댓글	user00	2018-02-04 04:11:31	2018-02-04 04:11:31
62	720,873	댓글	user00	2018-02-04 04:11:32	2018-02-04 04:11:32
63	720,873	댓글	user00	2018-02-04 04:11:33	2018-02-04 04:11:33
64	720,873	댓글	user00	2018-02-04 04:12:58	2018-02-04 04:12:58
65	720,873	댓글	user00	2018-02-04 04:12:58	2018-02-04 04:12:58
66	720,873	댓글	user00	2018-02-04 04:12:58	2018-02-04 04:12:58
67	720,873	댓글	user00	2018-02-04 04:12:58	2018-02-04 04:12:58
71	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
72	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
73	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
74	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
75	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
76	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
77	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
78	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
86	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
87	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
88	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
89	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
90	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
91	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
92	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
93	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
94	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
95	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
96	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
97	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
98	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
99	720,873	댓글	user00	2018-02-04 04:13:00	2018-02-04 04:13:00
100	720,873	댓글 100	user00	2018-02-04 04:13:00	2018-02-04 04:13:56
119	72,873	댓글을 추가합니다.	user00	2018-02-04 18:12:30	2018-02-04 18:12:30

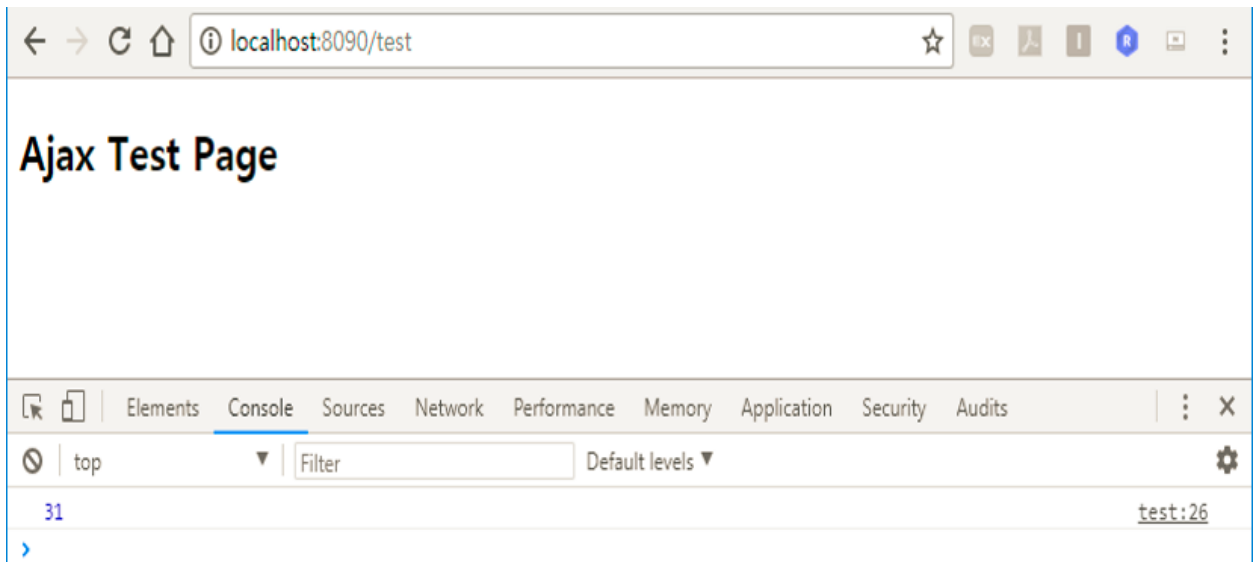
- test.jsp (jQuery ReplyController 호출)
  - @RestController의 경우 객체를 JSON 방식으로 전달하기 때문에 jQuery를 이용해서 호출할 때는 getJSON()을 이용한다.
  - Browser console에서 확인

```
<script>
  var bno = 720873;

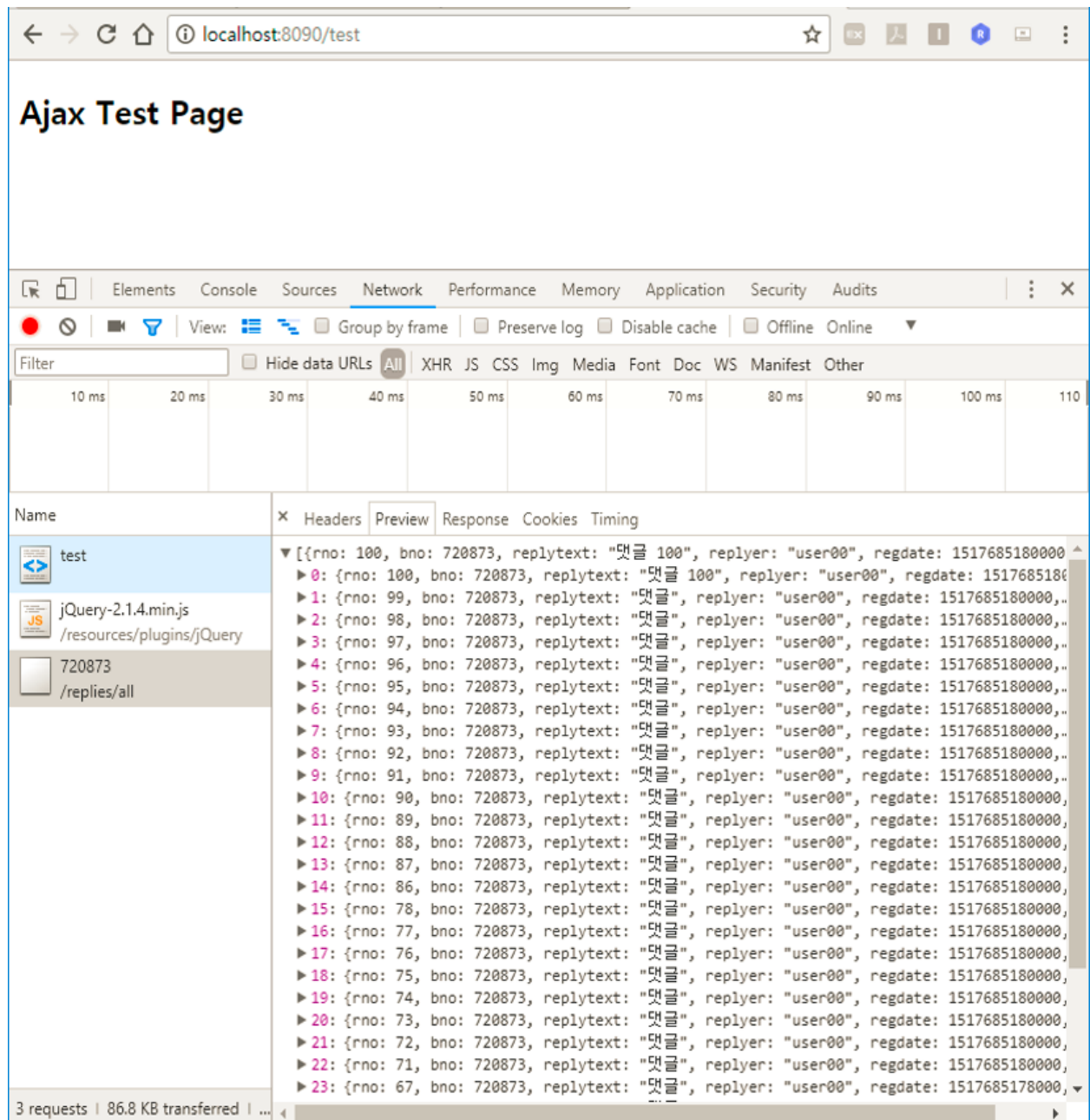
$.getJSON("/replies/all/" + bno
    , function(data) {
    console.log(data.length);
  });

</script>
```

- length



- Network



## 4.2.1 전체 댓글 목록 출력

### ○ 현재 댓글 목록의 경우

요소에 출력, 문자열을 구성해서 화면에 출력

#### ■ test.jsp (script)

- Ajax로 호출된 목록에 대해서 루프를 돌면서
- 태그를 생성해 낸다.
- 각
- 마다 댓글의 번호와 내용이 출력
- 
- 태그의 속성으로 사용된 'data-rno'의 'data-'로 시작되는 속성은 이름이나 개수에 관계없이 태그 내에서 자유롭게 사용할수 있는 속성으로 id나 name 속성을 대신해서 사용하기 편리하다.

```

<script>
var bno = 720873;

$.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
            + "</li>";
    });
    $("#replies").html(str);
});
</script>

```

#### ■ 스크립트 출력 결과

The screenshot shows a web browser at `localhost:8090/test` displaying an 'Ajax Test Page'. The page contains a list of 14 items, each with a number and the text '댓글' (comment). The list is ordered from 100 down to 86. The browser's developer tools are open, showing the HTML structure of the list and the CSS styles for the body element.

The HTML structure shown in the developer tools is:

```

<!DOCTYPE html>
<html>
<head>...</head>
<body> == $0
  <h2>Ajax Test Page</h2>
  <ul id="replies">
    <li data-rno="100" class="replyLi">100:댓글 100</li>
    <li data-rno="99" class="replyLi">99:댓글</li>
    <li data-rno="98" class="replyLi">98:댓글</li>
    <li data-rno="97" class="replyLi">97:댓글</li>
    <li data-rno="96" class="replyLi">96:댓글</li>
    <li data-rno="95" class="replyLi">95:댓글</li>
    <li data-rno="94" class="replyLi">94:댓글</li>
    <li data-rno="93" class="replyLi">93:댓글</li>
    <li data-rno="92" class="replyLi">92:댓글</li>
    <li data-rno="91" class="replyLi">91:댓글</li>
    <li data-rno="90" class="replyLi">90:댓글</li>
    <li data-rno="89" class="replyLi">89:댓글</li>
    <li data-rno="88" class="replyLi">88:댓글</li>
    <li data-rno="87" class="replyLi">87:댓글</li>
    <li data-rno="86" class="replyLi">86:댓글</li>
  </ul>

```

The CSS styles shown in the developer tools are:

```

element.style {
}
body {
  display: block;
  margin: 8px;
}

```

The browser's developer tools also show a box model diagram for the body element, indicating a margin of 8px, a border, and a padding of 8px. The dimensions of the body element are 945 x 702.906.


## 4.2.2 전체 목록에 대한 함수 처리

- 전체 목록 갱신하는 부분을 `getAllList()` 함수 처리

#### ■ test.jsp

- 댓글을 등록하는 부분을 구성
- 등록 이후 작성된 getAllList()를 이용해서 추가된 댓글을 볼 수 있게 작성

```
<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>
```



A screenshot of a web browser window. The address bar shows 'localhost:8090/test'. The page title is 'Ajax Test Page'. The page content includes a form with two input fields: 'REPLYER' and 'REPLY TEXT'. Below these fields is a button labeled 'ADD REPLY'.

#### ■ test.jsp ('ADD REPLY' 버튼 클릭 이벤트 처리)

- jQuery를 이용하여 \$.ajax()를 통해 서버를 호출
- 전송하는 데이터는 JSON으로 구성된 문자열을 사용하고, 전송받은 결과는 단순 문자열
- jQuery가 제공하는 \$.POST() 등을 사용하지 않는다.
- \$.ajax()를 이용한다.

```
$("#replyAddBtn").on("click", function() {
  var replyer = $("#newReplyWriter").val();
  var replytext = $("#newReplyText").val();

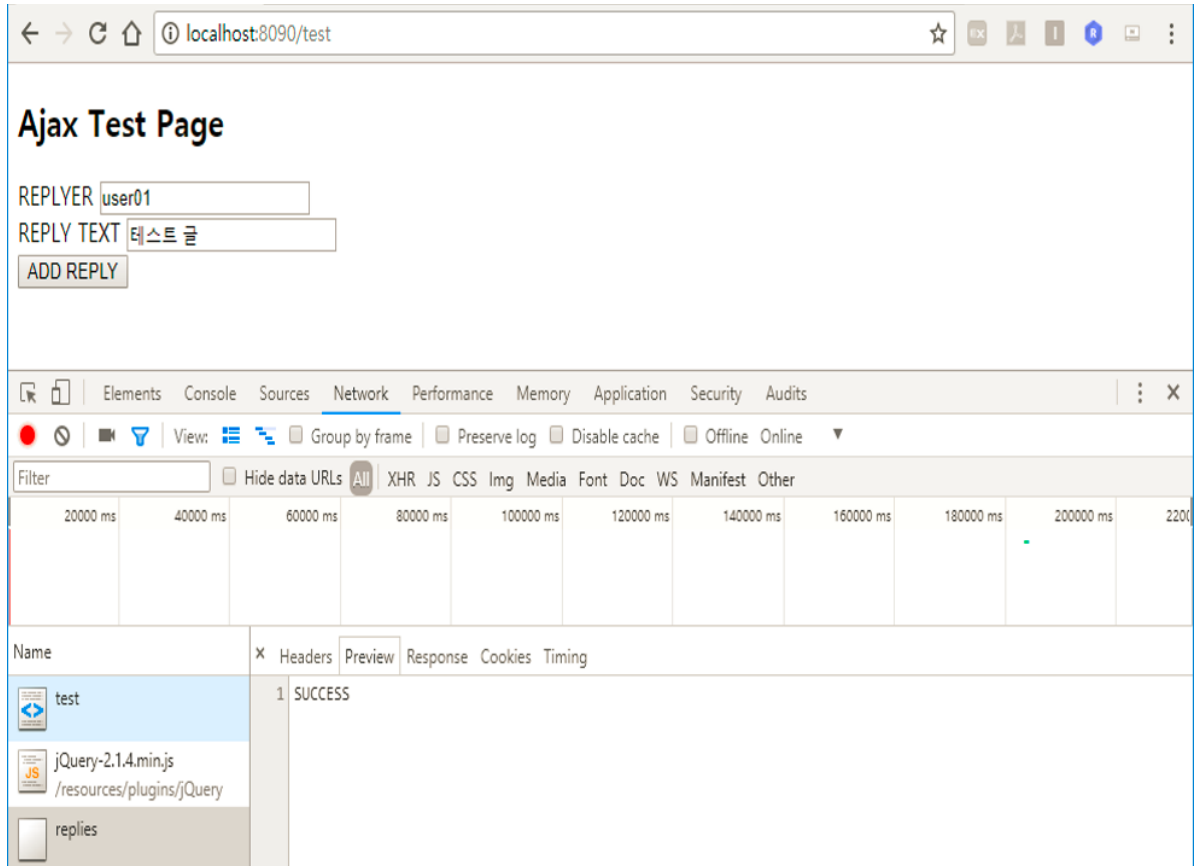
  $.ajax({
    type : 'post'
    , url : '/replies'
    , headers : {
      "Content-Type" : "application/json"
      , "X-HTTP-Method-Override" : "POST"
    }
    , dataType : 'text'
    , data : JSON.stringify({
      bno : bno
      , replyer : replyer
      , replytext : replytext
    })
    , success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

```

    }
  });
});

```

#### ■ 댓글 테스트



### 4.3.1 jQuery의 \$.post()를 사용하지 않는 이유

- \$.post()의 경우 일반적인 데이터 전송 방식에 적합하다.
  - 실행 결과는 제대로 호출되지 않고 415(지원되지 않는 타입) 상태 코드가 전송확인

```

$("#replyAddBtn").on("click", function() {
  var replyer = $("#newReplyWriter").val();
  var replytext = $("#newReplyText").val();

  $.post("/replies"
    , {
      replyer : replyer
    , replytext : replytext
    }
    , function(result) {
      alert(result);
    });
});

```

- 호출에 사용된 데이터를 살펴보면 일반적인 처리된 데이터와 동일한 것을 볼 수 있다.
- 이런 전송은 RestController에서는 @RequestBody 어노테이션이 제대로 처리되지 못하는 문제가 생긴다.
- ReplyController ( 기존에 존재하는 메서드 )

```
@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;
}
```

- @RequestBody의 경우 JSON으로 전송된 데이터를 ReplyVO 타입의 객체로 변환해주는 역할을 한다.
  - 이때의 데이터는 일반적인 데이터가 아닌 JSON으로 구성된 문자열 데이터
- \$.ajax()를 이용하는 코드에서 전송되는 데이터는 JSON.stringify()를 이용해 JSON데이터를 구성해서 전송

### 4.3.2 댓글 등록 후 전체 댓글 목록의 갱신

- 댓글 등록 후 경고창 -> 다시 목록을 갱신해서 추가된 글을 볼 수 있도록 호출 후의 처리 부분 수정
- test.jsp ( ajax )

```
$("#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) {
```

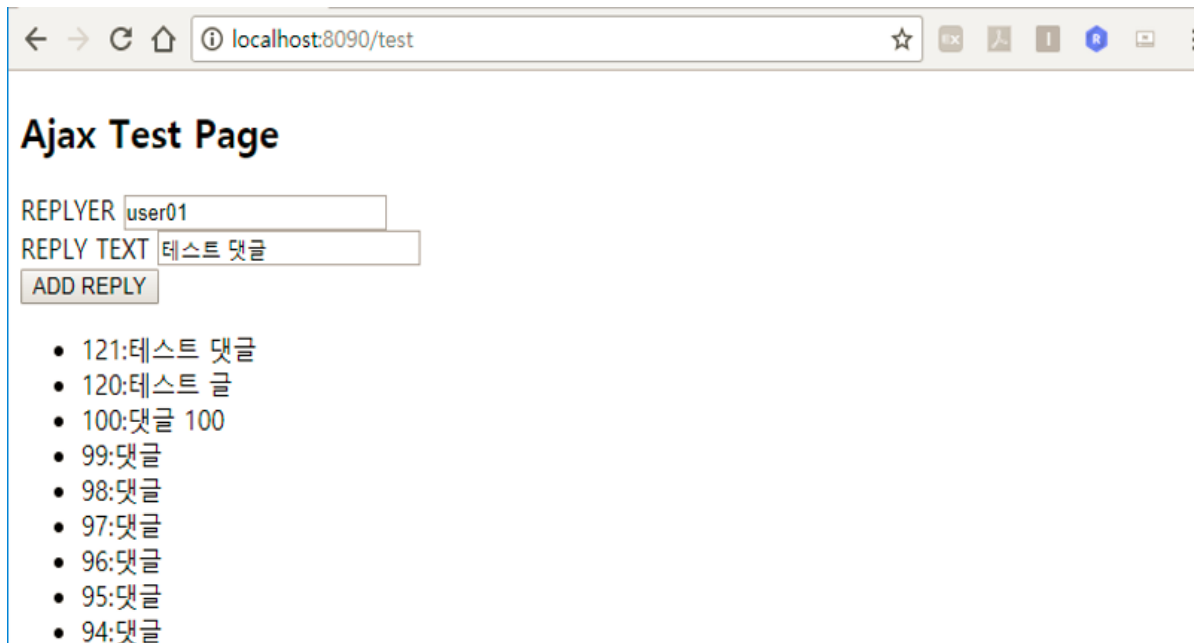


```

        if(result == 'SUCCESS'){
            alert("등록 되었습니다.");
            getAllList();
        }
    }
});
});

```

- 수정된 코드를 실행하면 댓글이 추가된 후 메시지가 보여지고, 목록이 다시 갱신되는 것을 볼 수 있다.



## 4.4 댓글 조회 및 수정/삭제

- 버튼 script 추가

```

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);
        });
}

```

- 댓글의 각 항목을 의미하는

- 의 경우 Ajax의 통신 후에 생기는 요소들이기 때문에 이벤트 처리를 할 때 기존에 존재하는  
을 이용해서 이벤트를 등록합니다.
- 이벤트는 위임(delegation) 방식으로 전달하는데, class의 속성값이 'replyLi'로 된 요소 밑의  
을 찾아서 이벤트를 전달하게 한다.
  - jQuery의 이벤트는 위처럼 아직은 존재하지 않는 요소에 이벤트를 위임해주는 편리한  
기능이 있으므로, 한번에 모든 목록에 대한 클릭 이벤트를 처리할 수 있다.

#### 4.4.1 수정과 삭제를 위한

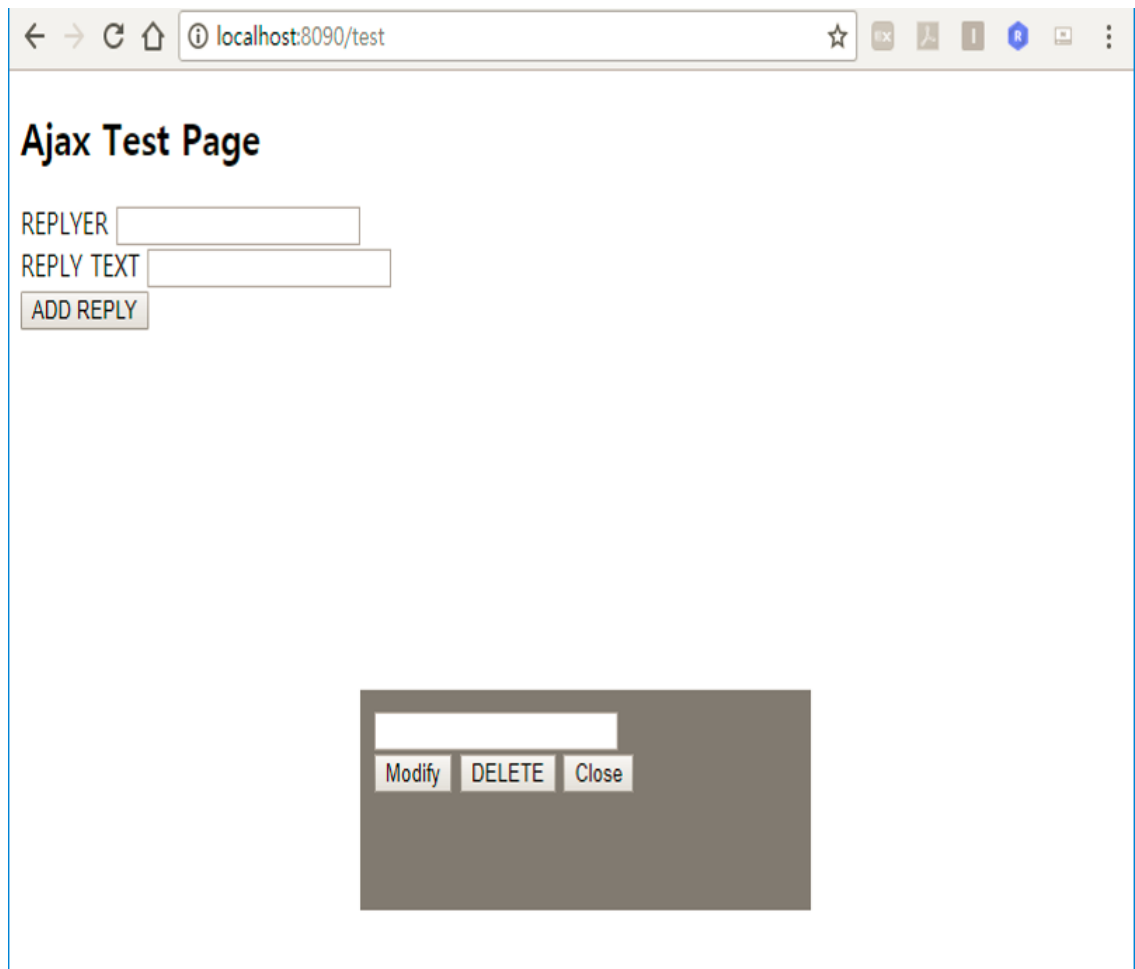
- 수정과 삭제 기능을 수행하기 위한 폼
- test.jsp (div)

```
<div id = "modDiv">
  <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.jsp (style)

```
<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>
```

- 결과



■

#### 4.4.2 에 댓글 보이기

■ test.jsp

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

#### 4.4.3 삭제 호출하기

■ test.jsp

- 삭제 작업 이후에는 보여지고 있는  
를 안보이게 처리(hide()) 한 후에 다시 전체 목록을 가져오는 getAllList()를 호출하여 처리

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

#### 4.4.4 수정 작업 처리하기

- test.jsp (replyModBtn)
  - 댓글 수정의 처리에는 PUT 방식이 사용되었고, 수정되는 게시물의 번호는 URI에 추가해서 전송한다.
  - 수정되어야 하는 데이터는 JSON으로 구성해서 전송한다.

```
$("#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'){
```

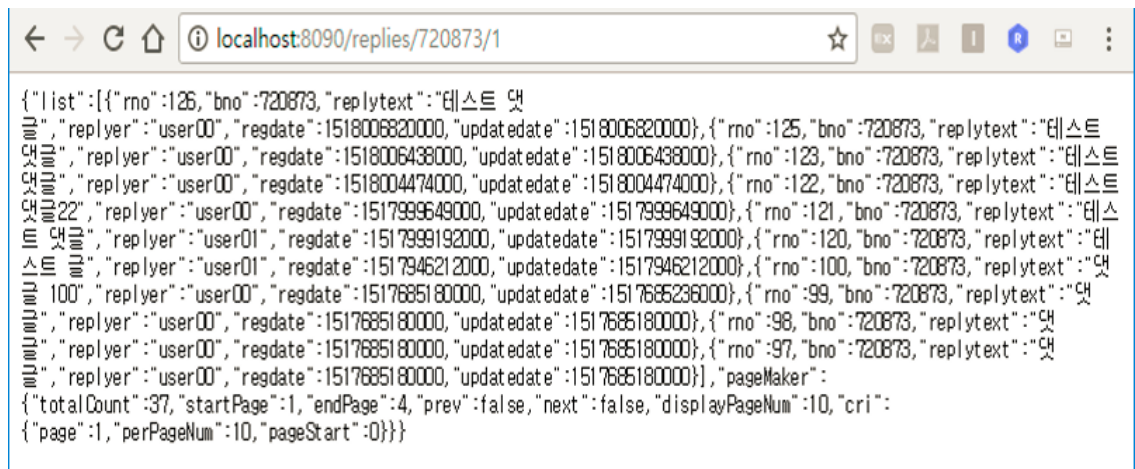
```

        alert("수정 되었습니다.");
        $("#modDiv").hide("slow");
        getAllList();
        getPageList(replyPage); // 419페이지 참조
    }
}
});
});

```

## 4.5 댓글의 페이징 처리

- 댓글의 목록에 해당하는 'list' 데이터와 페이지 구성에 필요한 'pageMaker' 데이터로 구성된다.
- pageMaker에는 페이징 처리에 필요한 데이터가 이미 계산된 상태이므로 이를 이용해서 페이지를 처리할 수 있다.



### 4.5.1 댓글 페이지를 위한 처리

- test.jsp (pagination)

```

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

```

```

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);
  });
}

```

#### ■ test.jsp (printPaging)

- pageMaker를 이용해서 화면에 페이지 번호를 출력

```

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);
}

```

```

getPageList(1);

```

#### ■ 페이징 처리 CSS

```

<style>
.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>

```

#### ■ 페이징 처리 결과

← → ↺ 📍 localhost:8090/test ☆ 🔍 📄 📌 🔒 📺

## Ajax Test Page

REPLYER   
 REPLY TEXT   
 ADD REPLY

- 129:테스트 댓글 MOD
- 128:123 MOD
- 126:테스트 댓글 MOD
- 125:테스트 댓글 MOD
- 123:123:테스트 댓글MOD MOD
- 122:테스트 댓글22 MOD
- 121:테스트 댓글 MOD
- 120:테스트 글 MOD
- 100:댓글 100 MOD
- 99:댓글 MOD

1 2 3 4

### 4.5.2 페이지 번호 이벤트 처리

- 화면에 페이지 번호가 출력되었다면 클릭 이벤트 처리를 한다.
- 태그의 내용 중 페이지 번호를 추출해서 Ajax 호출을 처리한다.
- test.jsp (.pagination)

```

var replyPage = 1;
$(".pagination").on("click", "li a", function(data) {
  event.preventDefault();
  replyPage = $(this).attr("href");
  getPageList(replyPage);
});

```