5. REST & Ajax

Objectives

- REST방식의 데이터 교환 방식의 이해
- Ajax를 통한 JSON 데이터 통신
- jQuery를 이용하는 Ajax 처리
- JavaScript의 모듈 패턴

1. REST방식으로 전환

웹의 과거와 현재

- 과거의 웹 서비스
  - 고정된 브라우저의 주소창
  - 특정한 확장자를 이용하는 모델 2 방식(ex> *.do)
  - 특정한 파라미터에 의한 분기 구조
- 현재의 웹 서비스
  - URI + 식별데이터
  - GET/POST외에 PUT/DELETE 등의 다양한 전송 방식 사용
  - 서버에서는 순수한 데이터만을 서비스 하는 방식

REST방식

- REST는 'Representational State Transfer'의 약어로 하나의 URI는 하나의 고유한 리소스(Resource)를 대표하도록 설계된다는 개념에 전송방식을 결합해서 원하는 작업을 지정

- 스프링에서는 다양한 어노테이션과 기능을 통해서 REST방식의 서비스를 간편하게 구축할 수 있음

어노테이션	기능
@RestController	Controller가 REST 방식을 처리하기 위한 것임을 명시합니다.
@ResponseBody	일반적인 JSP와 같은 뷰로 전달되는 게 아니라 데이터 자체를 전달하기 위한 용도
@PathVariable	URL 경로에 있는 값을 파라미터로 추출하려고 할 때 사용
@CrossOrigin	Ajax의 크로스 도메인 문제를 해결해주는 어노테이션
@RequestBoby	JSON 데이터를 원하는 타입으로 바인딩 처리

@RestController

- 스프링 4에서부터는 @Controller 외에 @RestController라는 어노테이션을 추가해서 해당 Controller의 모든 메서드의 리턴 타입을 기존과 다르게 처리한다는 것을 명시
- @RestController는 메서드의 리턴 타입으로 사용자가 정의한 클래스 타입을 사용할 수 있고, 이를 JSON이나 XML로 자동으로 처리

예제프로젝트의 준비

- 데이터의 처리는 XML과 JSON을 이용할 것이므로 pom.xml을 변경
- jackson-databind와 Jackson-dataformat-xml 라이브러리 활용
- Java객체를 JSON으로 쉽게 변환할 수 있는 gson 라이브러리

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.6</version>
  </dependency>

<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.6</version>
  </dependency>

<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.2</version>
  </dependency>
```

@RestController의 반환 타입

- @RestController를 사용하는 컨트롤러에서는 다음과 같은 반환 타입들을 사용한다.
  - String 혹은 Integer 등의 타입들
  - 사용자 정의 타입
  - ResponseEntity<> 타입
- 주로 ResponseEntity 타입을 이용하는 것이 일반적

SampleVO클래스와 SampleController

- JSON 혹은 XML로 변환될 데이터

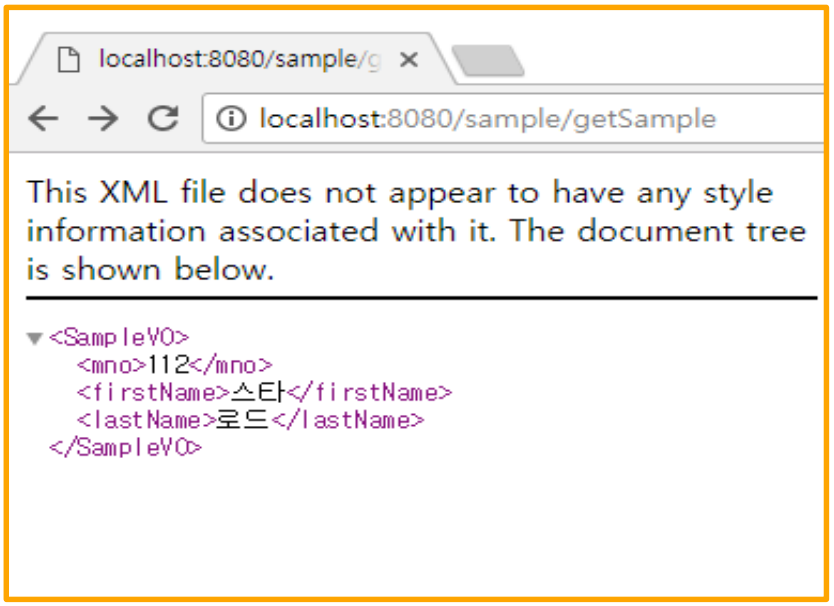
```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class SampleVO {
  private Integer mno;
  private String firstName;
  private String lastName;
}
```

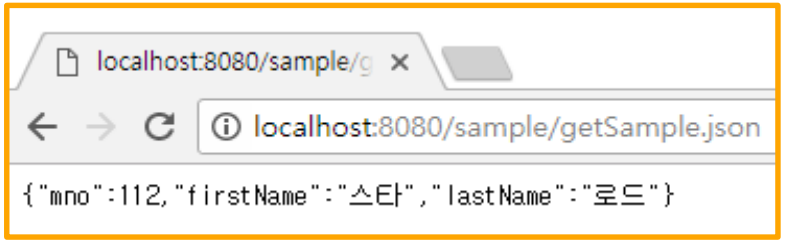
- SampleVO를 서비스하는 SampleController

```
@RestController
@RequestMapping("/sample")
@Log4j
public class SampleController {
}
```

JSON/XML의 테스트

```
 @GetMapping(value = "/getSample", produces =
{ MediaType.APPLICATION_JSON_UTF8_VALUE,MediaType.APPLICATION_XML_VALUE })
  public SampleVO getSample() {
    return new SampleVO(112, "스타", "로드");
 }
```



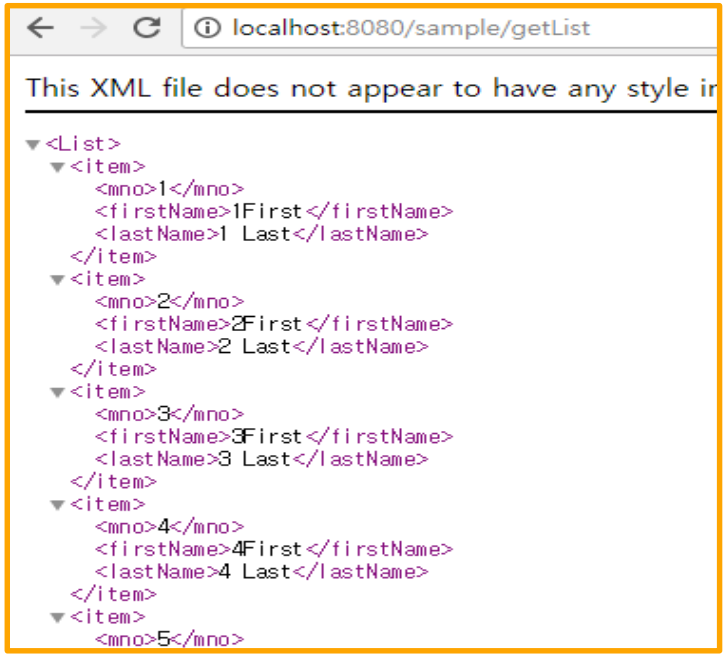


확장자에 따라 다른 타입으로 서비스

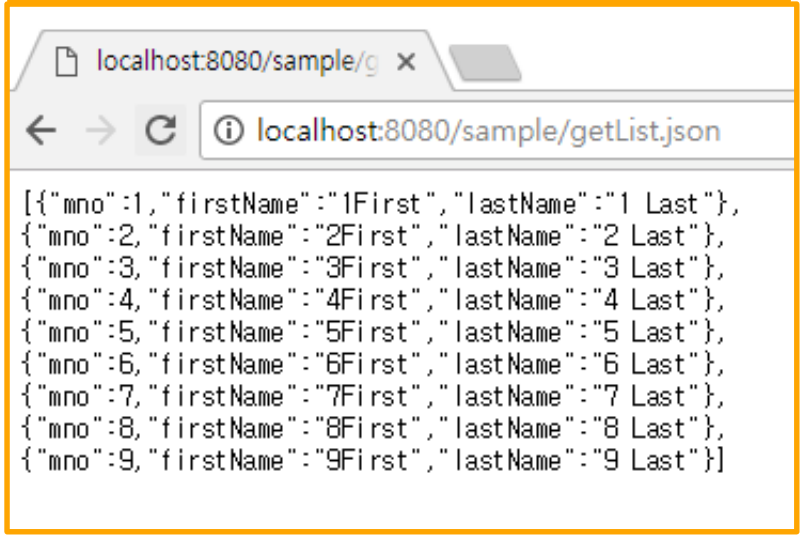
Collection타입의 객체 반환-List

```
@GetMapping(value = "/getList")
 public List<SampleVO> getList() {
   return IntStream.range(1, 10)
         .mapToObj(i -> new SampleVO(i, i + "First", i + " Last"))
        .collect(Collectors.toList());
 }
```

http://localhost:8080/sample/getList



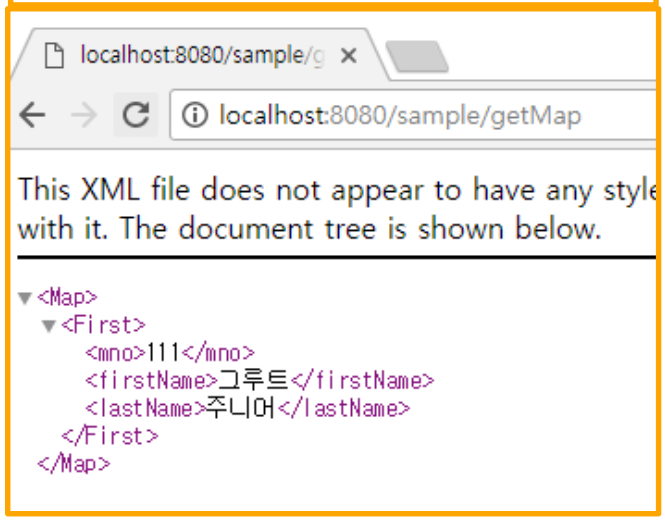
http://localhost:8080/sample/getList.json



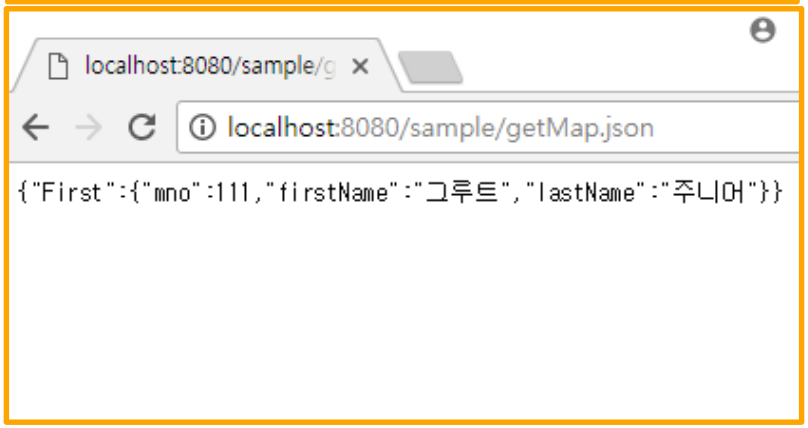
Collection타입의 객체 반환-Map

```
@GetMapping(value = "/getMap")
 public Map<String, SampleVO> getMap() {
   Map<String, SampleVO> map = new HashMap<>();
   map.put("First", new SampleVO(111, "그루트", "주니어"));
   return map;
 }
```

http://localhost:8080/sample/getMap



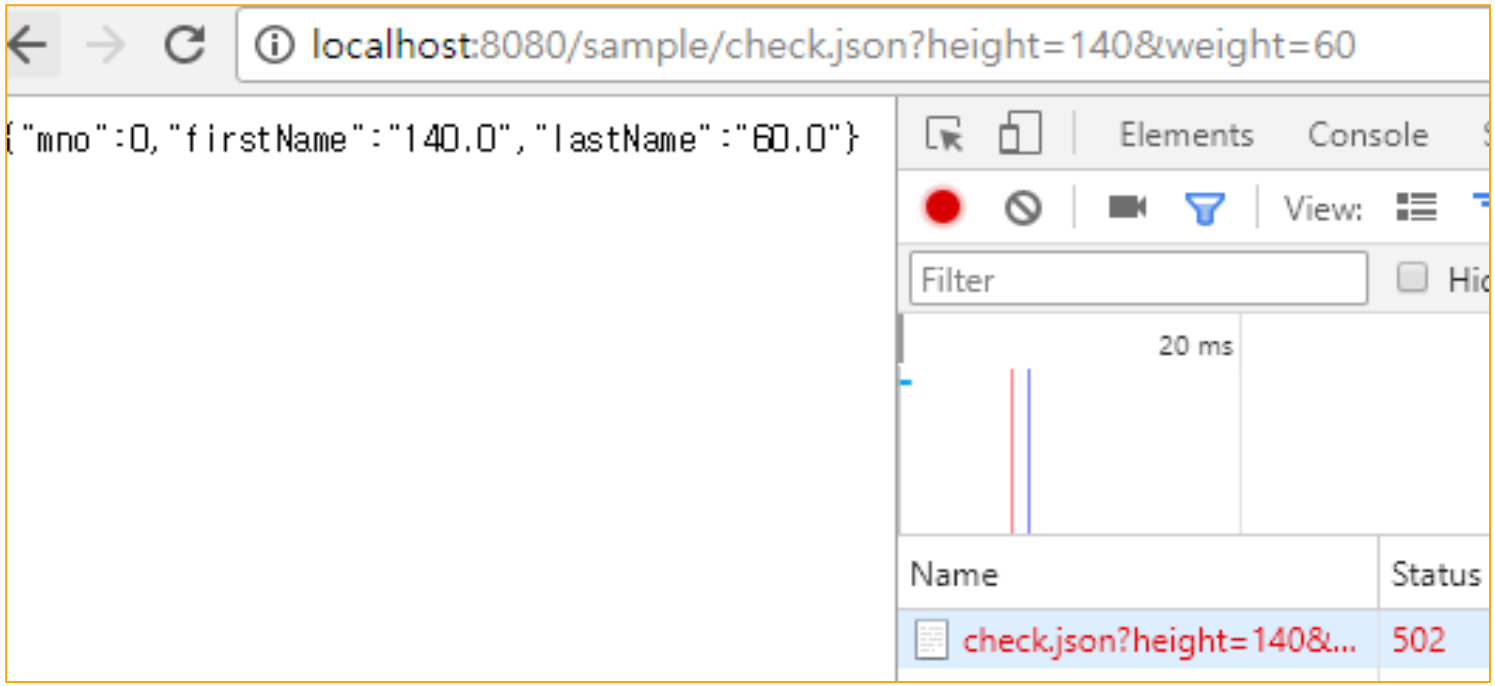
http://localhost:8080/sample/getMap.json



ResponseEnitity타입

- 단순히 데이터뿐만 아니라 브라우저에 HTTP상태 코드등 추가적인 데이터를 전달할 수 있다는 장점

```
@GetMapping(value = "/check", params = { "height", "weight" })
  public ResponseEntity<SampleVO> check(Double height, Double weight) {
    SampleVO vo = new SampleVO(000, "" + height, "" + weight);
    ResponseEntity<SampleVO> result = null;
    if (height < 150) {
      result = ResponseEntity.status(HttpStatus.BAD_GATEWAY).body(vo);
    } else {
      result = ResponseEntity.status(HttpStatus.OK).body(vo);
    }
    return result;
 }
```


localhost:8080/sample/check.json?height=140&weight=60
{"mno":0,"firstName":"140.0","lastName":"60.0"}
Elements
Console
View:
Filter
20 ms
Name
Status
check.json?height=140&...
502

@RestController의 파라미터

- @PathVariable: 일반 컨트롤러에서도 사용이 가능하지만 REST 방식에서 자주 사용됩니다. URL 경로의 일부를 파라미터로 사용할 때 이용
- @RequestBody: JSON 데이터를 원하는 타입의 객체로 변환해야 하는 경우에 주로 사용
- 일반 <form>방식으로 처리된 데이터

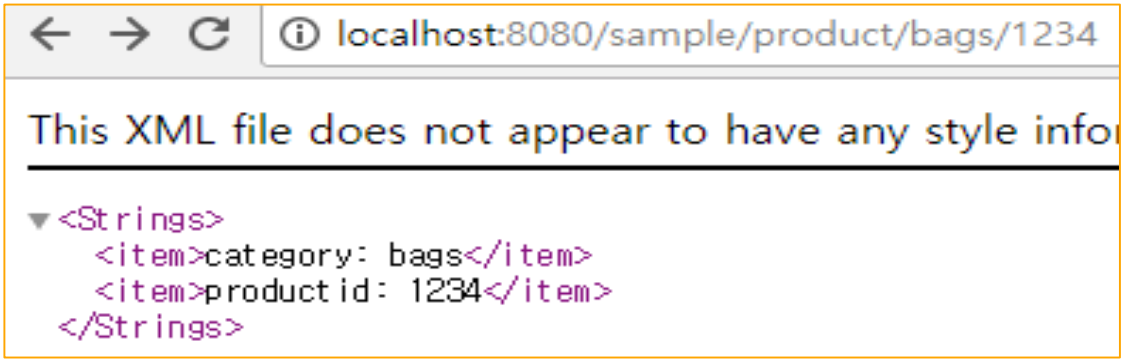
@PathVariable

- URI경로 중간에 들어간 값을 얻기 위해서 사용

http://localhost:8080/sample/{sno}

http://localhost:8080/sample/{sno}/page/{pno}

```
@GetMapping("/product/{cat}/{pid}")
  public String[] getPath(
      @PathVariable("cat") String cat,
      @PathVariable("pid") Integer pid) {
    return new String[] { "category: " + cat, "productid: " + pid };
 }
```



@RequestBody

- 전송된 데이터가 JSON이고, 이를 컨트롤러에서는 사용자 정의 타입의 객체로 변환할때 사용

```
@Data
public class Ticket {

  private int tno;
  private String owner;
  private String grade;
}
```

```
@PostMapping("/ticket")
public Ticket convert(@RequestBody Ticket ticket){
  log.info("convert.....ticket" + ticket);
  return ticket;
}
```

일반적으로 브라우저에서는 JSON형태의 데이터를 전송할 수 없으므로
별도의 REST관련 도구를 이용해서 테스트를 진행해야 함

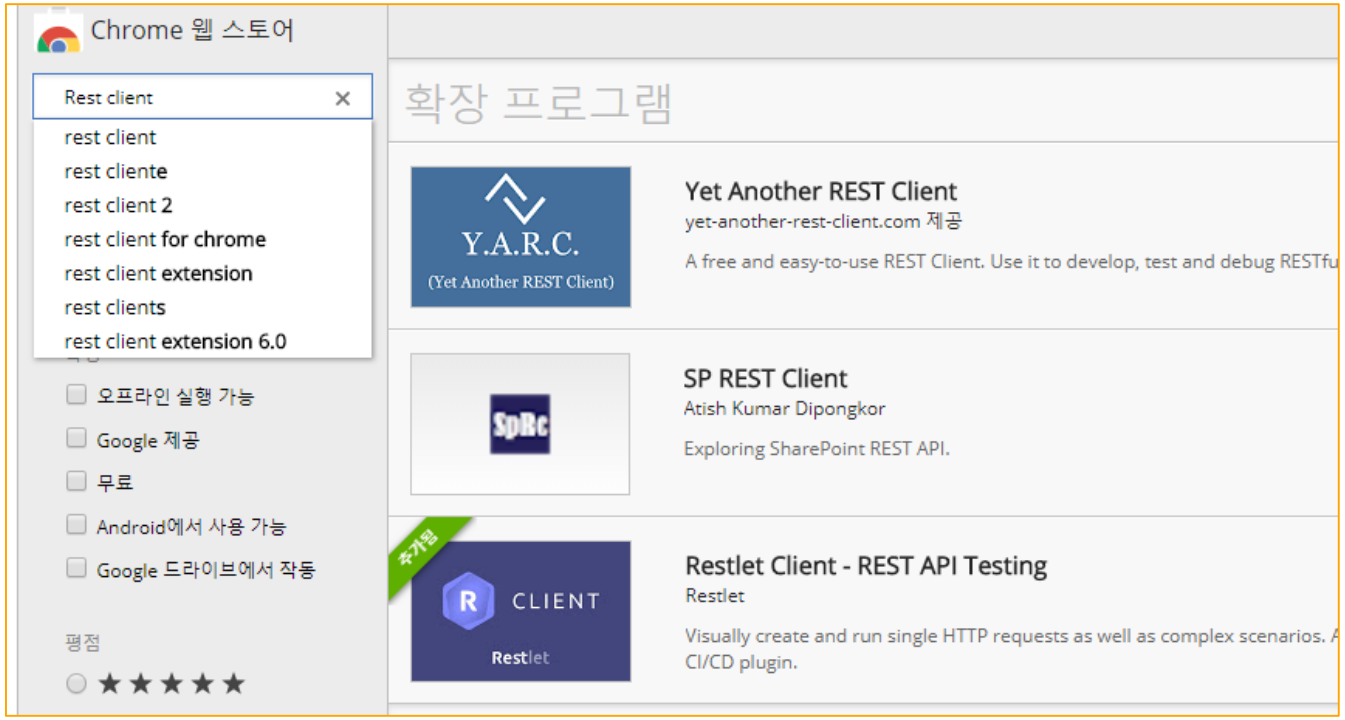
REST 방식의 테스트

- JUnit기반의 테스트
- 별도의 프로그램인나 크롬 확장 프로그램등을 이용하는 테스트

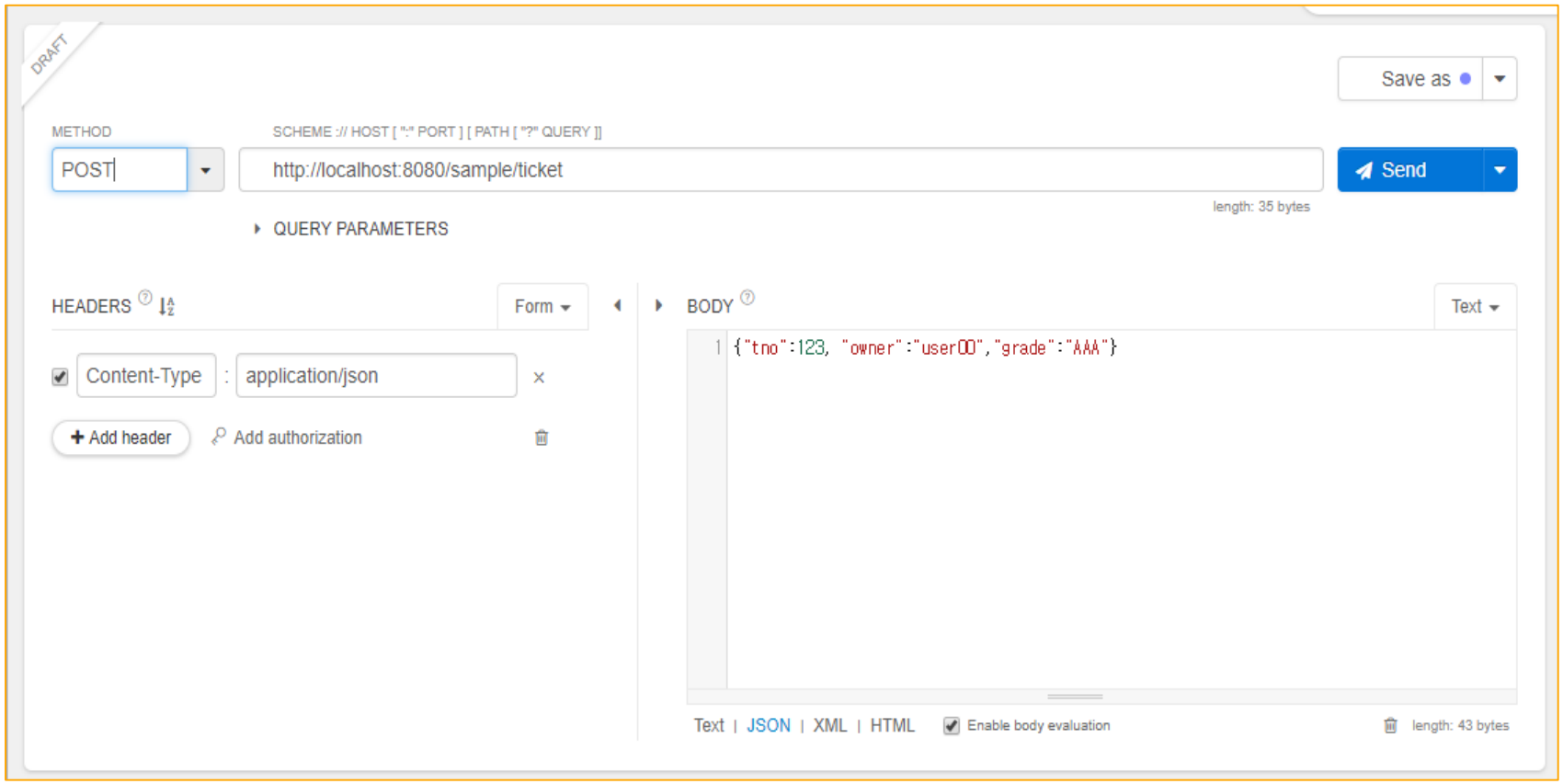
```
@Test
 public void testConvert() throws Exception {
   Ticket ticket = new Ticket();
   ticket.setTno(123);
   ticket.setOwner("Admin");
   ticket.setGrade("AAA");

   //GSON 라이브러리를 이용해서 JSON 데이터로 변환
   String jsonStr = new Gson().toJson(ticket);
   log.info(jsonStr);

   mockMvc.perform(post("/sample/ticket")
        .contentType(MediaType.APPLICATION_JSON)
        .content(jsonStr))
        .andExpect(status().is(200));
 }
```

크롬 확장 프로그램을 이용하는 테스트

- 크롬 앱 스토어에서 'REST'방식 호출이 가능한 다양한 프로그램들이 존재
  - Yet Another REST Client
  - Restlet Client
  - etc



Restlet을 이용한 테스트



다양한 전송방식과 URI설계

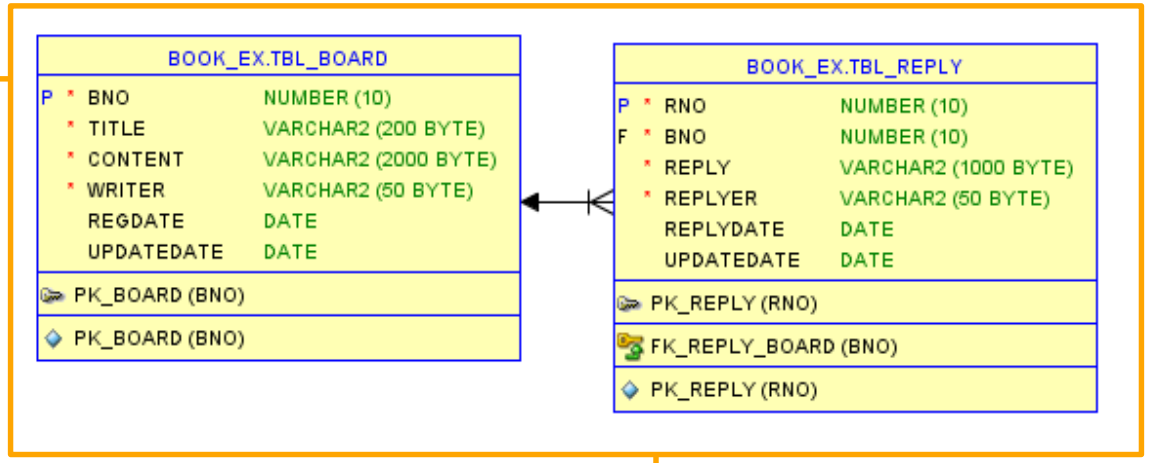
- REST 방식의 데이터 교환에서 가장 특이한 점은 기존의 GET/POST 외에 다양한 방식으로 데이터를 전달한다는 점

작업	전송방식
Create	POST
Read	GET
Update	PUT
Delete	DELETE

작업	전송방식	**URI**
등록	POST	/members/new
조회	GET	/members/{id}
수정	PUT	/members/{id} + body (json 데이터 등)
삭제	DELETE	/member/{id}

2. Ajax를 이용하는 댓글 처리

댓글 처리를 위한 테이블 설계




```
create table tbl_reply (
  rno number(10,0),
  bno number(10,0) not null,
  reply varchar2(1000) not null,
  replyer varchar2(50) not null,
  replyDate date default sysdate,
  updateDate date default sysdate
);

create sequence seq_reply;

alter table tbl_reply add constraint pk_reply primary key (rno);

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

식별키(PK) 지정

외래키(FK) 지정

ReplyVO클래스의 추가/Mapper 준비

```java
@Data
public class ReplyVO {

  private Long rno;
  private Long bno;

  private String reply;
  private String replyer;
  private Date replyDate;
  private Date updateDate;

}
```

```java
public interface ReplyMapper {


}
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.zerock.mapper.ReplyMapper">



</mapper>
```

Mapper CRUD작업-create

- 외래키가 걸려있으므로 실제 존재하는 게시물 번호를 이용해서 테스트를 진행해야 함

```
public interface ReplyMapper {
  public int insert(ReplyVO vo);
}
```

```
<mapper namespace="org.zerock.mapper.ReplyMapper">
  <insert id="insert">
    insert into tbl_reply (rno, bno, reply, replyer)
    values (seq_reply.nextval, #{bno}, #{reply}, #{replyer})
  </insert>
</mapper>
```

Mapper 특정 댓글 조회

- 댓글의 번호를 이용해서 특정 댓글 조회

```
public interface ReplyMapper {
  public int insert(ReplyVO vo);
  public ReplyVO read(Long bno);
}
```

```
<select id="read" resultType="org.zerock.domain.ReplyVO">
    select * from tbl_reply where rno = #{rno}
</select>
```

Mapper 특정 댓글 삭제/수정

- 댓글의 번호로 삭제

```
public interface ReplyMapper {
 …생략…
  public int delete(Long bno);
}
```

```
<delete id="delete">
    delete from tbl_reply
    where rno = #{rno}
</delete>
```

- 특정 댓글 수정

```
public interface ReplyMapper {
 ...
 public int update(ReplyVO reply);
}
```

```
<update id="update">
    update tbl_reply
    set reply = #{reply},
    updatedate = sysdate
    where rno = #{rno}
</update>
```

@Param 어노테이션과 댓글 목록

- MyBatis의 파라미터는 1개만 허용
- 이를 해결하기 위해 1)Map형태를 사용하거나, 2)별도의 클래스를 이용하거나, 3)@Param을 이용할 수 있음

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

```
<select id="getListWithPaging"
    resultType="org.zerock.domain.ReplyVO">
    select rno, bno, reply, replyer, replyDate, updatedate
    from tbl_reply
    where bno = #{bno}
    order by rno asc
</select>
```

서비스영역과 컨트롤러 처리

- ReplyService 인터페이스와 ReplyServiceImpl 클래스 생성

```
public interface ReplyService {
  public int register(ReplyVO vo);
  public ReplyVO get(Long rno);
  public int modify(ReplyVO vo);
  public int remove(Long rno);
  public List<ReplyVO>
   getList(Criteria cri, Long bno);
}
```

```
@Service
@Log4j
public class ReplyServiceImpl implements
ReplyService {
  @Setter(onMethod_ = @Autowired)
  private ReplyMapper mapper;

  @Override
  public int register(ReplyVO vo) {
    log.info("register....." + vo);
    return mapper.insert(vo);
  }
..생략...
```

ReplyController의 설계

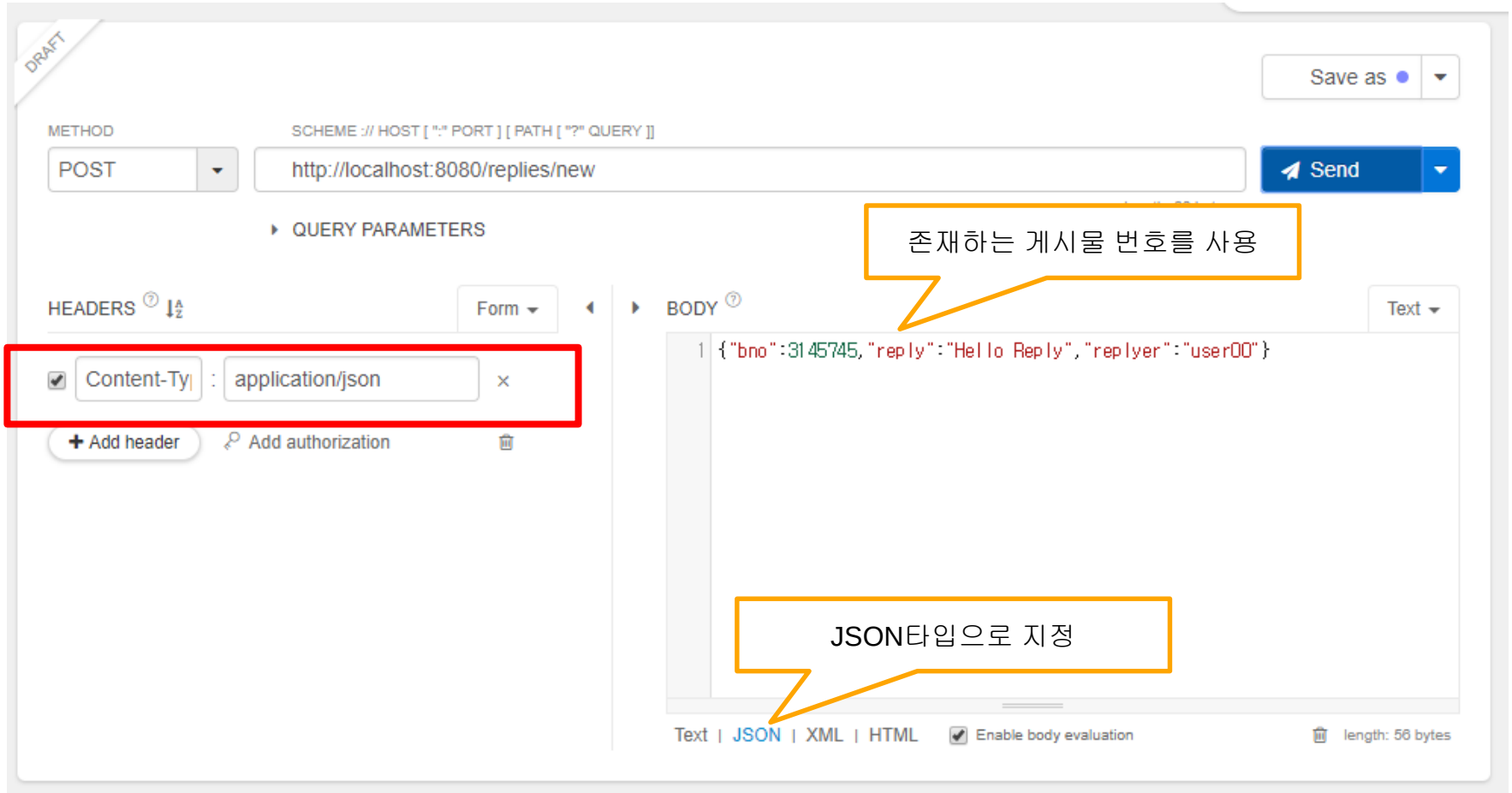
작업	URL	HTTP 전송방식
등록	/replies/new	POST
조회	/replies/:rno	GET
삭제	/replies/:rno	DELETE
수정	/replies/:rno	PUT or PATCH
페이지	/replies/pages/:bno/:page	GET

```
@RequestMapping("/replies/")
@RestController
@Log4j
@AllArgsConstructor
public class ReplyController {

   private ReplyService service;

}
```

등록작업과 테스트

- 댓글 등록의 경우 브라우저에서는 JSON 타입으로 된 댓글 데이터를 전송하고, 서버에서는 댓글의 처리 결과가 정상적으로 되었는지 문자열로 결과를 알려 주는 방식으로 처리

```
@PostMapping(value = "/new", consumes = "application/json",
      produces = { MediaType.TEXT_PLAIN_VALUE })
  public ResponseEntity<String> create(@RequestBody ReplyVO vo) {
    log.info("ReplyVO: " + vo);
    int insertCount = service.register(vo);
    log.info("Reply INSERT COUNT: " + insertCount);
    return insertCount == 1 ? new ResponseEntity<>("success", HttpStatus.OK)
        : new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
  }
```


DRAFT
Save as
METHOD
SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]
POST
http://localhost:8080/replies/new
Send
QUERY PARAMETERS
존재하는 게시물 번호를 사용
HEADERS
Form
BODY
Text
Content-Ty
application/json
{"bno":3145745,"reply":"Hello Reply","replyer":"user00"}
Add header
Add authorization
JSON타입으로 지정
Text | JSON | XML | HTML
Enable body evaluation
length: 56 bytes

특정 게시물의 댓글 목록

```
@GetMapping(value = "/pages/{bno}/{page}",
      produces = {
          MediaType.APPLICATION_XML_VALUE,
          MediaType.APPLICATION_JSON_UTF8_VALUE })
  public ResponseEntity<List<ReplyVO>> getList(
      @PathVariable("page") int page, @PathVariable("bno") Long bno) {
    log.info("getList.................");
    Criteria cri = new Criteria(page,10);
    log.info(cri);
    return new ResponseEntity<>(service.getList(cri, bno), HttpStatus.OK);
  }
```

XML과 JSON타입으로 서비스


localhost:8080/replies/pages/3145745/1
This XML file does not appear to have any style information associated
<ReplyPageDTO>
<replyCnt>8</replyCnt>
<list>
<list>
<rno>5</rno>
<bno>3145745</bno>
<reply>댓글 테스트 5</reply>
<replyer>replyer5</replyer>
<replyDate>1534550815000</replyDate>
<updateDate>1534550815000</updateDate>
</list>
<list>
<rno>10</rno>
<bno>3145745</bno>
<reply>댓글 테스트 10</reply>
<replyer>replyer10</replyer>
<replyDate>1534550815000</replyDate>
<updateDate>1534550815000</updateDate>
</list>
<list>
<rno>11</rno>
<bno>3145745</bno>
<reply>Hello Reply</reply>
<replyer>user00</replyer>
<replyDate>1534552080000</replyDate>
<updateDate>1534552080000</updateDate>
</list>
<list>
<rno>12</rno>
<bno>3145745</bno>
<reply>Hello Reply</reply>
<replyer>user00</replyer>
<replyDate>1534552193000</replyDate>
<updateDate>1534552193000</updateDate>
</list>

댓글의 삭제/조회

```
@GetMapping(value = "/{rno}", produces = { MediaType.APPLICATION_XML_VALUE,
        MediaType.APPLICATION_JSON_UTF8_VALUE })
public ResponseEntity<ReplyVO> get(@PathVariable("rno") Long rno) {
  log.info("get: " + rno);
  return new ResponseEntity<>(service.get(rno), HttpStatus.OK);
}

@DeleteMapping(value= "/{rno}" ,produces = { MediaType.TEXT_PLAIN_VALUE })
public ResponseEntity<String> remove(@PathVariable("rno") Long rno) {
  log.info("remove: " + rno);
  return service.remove(rno) == 1 ? new ResponseEntity<>("success", HttpStatus.OK)
      : new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
}
```

댓글의 수정

```
@RequestMapping(method = { RequestMethod.PUT, RequestMethod.PATCH },
      value = "/{rno}", consumes = "application/json",
      produces = { MediaType.TEXT_PLAIN_VALUE })
 public ResponseEntity<String> modify(@RequestBody ReplyVO vo,
      @PathVariable("rno") Long rno) {
   vo.setRno(rno);

   log.info("rno: " + rno);
   log.info("modify: " + vo);

   return service.modify(vo) == 1 ? new ResponseEntity<>("success", HttpStatus.OK)
       : new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);

 }
```

JavaScript의 준비 – 모듈화

- JS의 모듈 패턴
  - 여러 기능들을 모아서 하나의 모듈화
  - 클로저를 이용해서 상태 유지
  - 여러 함수들이 메서드화 되므로 객체지향 구조에 적합

reply.js

```
console.log("Reply Module.....");

var replyService = (function(){

  function add(reply, callback){
    console.log("reply.............");
  }

  return {add:add};
})();
```

reply.js 댓글 등록

Ajax처리후 동작해야 하는 함수

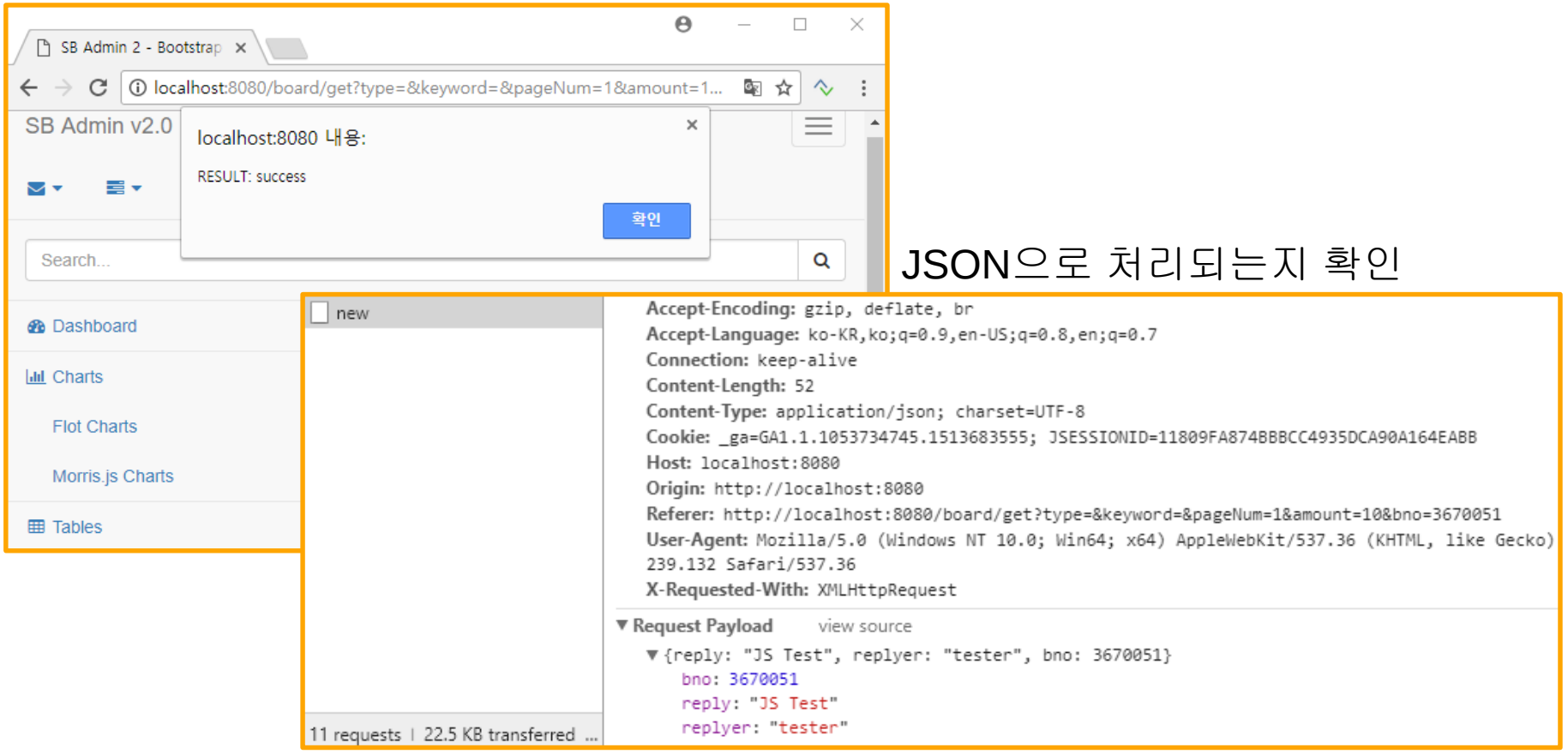
```
var replyService = (function() {
  function add(reply, callback, error) {
    console.log("add reply..............");
    $.ajax({
      type : 'post',
      url : '/replies/new',
      data : JSON.stringify(reply),
      contentType : "application/json; charset=utf-8",
      success : function(result, status, xhr) {
        if (callback) { callback(result); }
      },
      error : function(xhr, status, er) {
        if (error) { error(er); }
      }
    })
  }
   return {
    add : add
  };
})();
```

Ajax로 ReplyController호출

모듈 패턴으로 외부에 노출하는 정보

조회 화면에서 호출

```
script type="text/javascript"  src="/resources/js/reply.js"></script>
<script>
console.log("===============");
console.log("JS TEST");
var bnoValue = '<c:out value="${board.bno}"/>';

//for replyService add test
replyService.add(
    {reply:"JS Test", replyer:"tester", bno:bnoValue} ,
    function(result){
      alert("RESULT: " + result);
    }
);

</script>
```

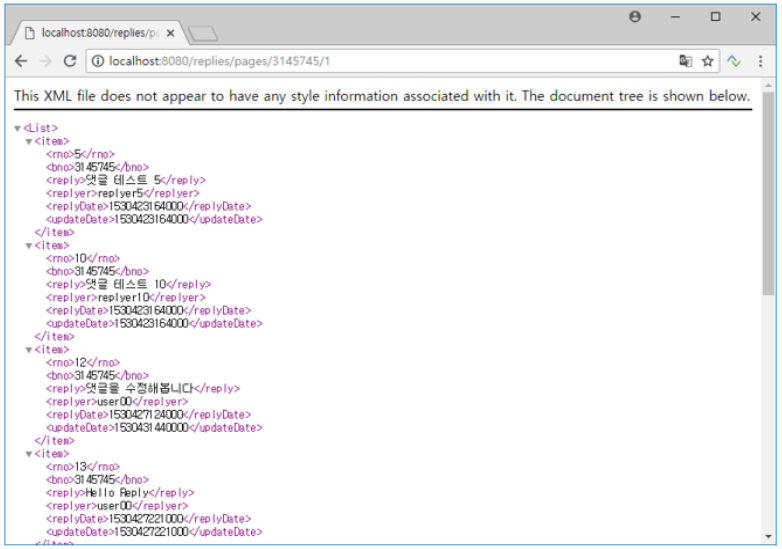


JSON으로 처리되는지 확인

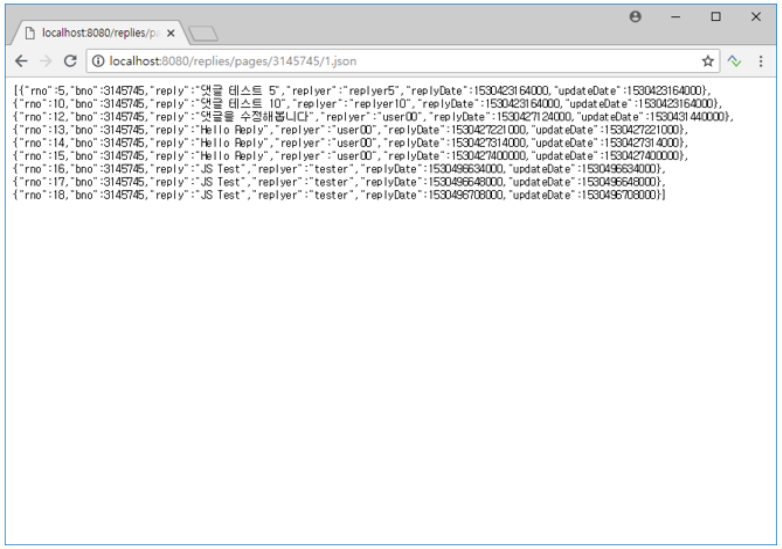
댓글의 목록 처리

댓글의 처리 전에 우선적으로 확인한 후에 진행

http://localhost:8080/replies/pages/3145745/1



http://localhost:8080/replies/pages/3145745/1.json



getJSON( )처리 –reply.js

```
function getList(param, callback, error) {
    var bno = param.bno;
    var page = param.page || 1;

    $.getJSON("/replies/pages/" + bno + "/" + page + ".json",
        function(data) {
          if (callback) {
            callback(data);
          }
        }).fail(function(xhr, status, err) {
      if (error) {
        error();
      }
    });
  }
```

Ajax로 ReplyController호출

화면에서의 호출 테스트

```
replyService.getList({bno:bnoValue, page:1}, function(list){


    for(var i = 0,  len = list.length||0; i < len; i++ ){
      console.log(list[i]);
    }
});
```

댓글의 삭제와 갱신

reply.js의 일부

```
function remove(rno, callback, error) {
   $.ajax({
     type : 'delete',
     url : '/replies/' + rno,
     success : function(deleteResult, status, xhr) {
       if (callback) {
         callback(deleteResult);
       }
     },
     error : function(xhr, status, er) {
       if (error) {
         error(er);
       }
     }
   });
 }
```

전송방식은 DELETE 방식 사용

댓글 삭제 테스트

- 존재하는 댓글의 번호를 이용해서 처리

```
//23번 댓글 삭제 테스트
replyService.remove(23, function(count) {
  console.log(count);
  if (count === "success") {
    alert("REMOVED");
  }
}, function(err) {
  alert('ERROR...');
});
```

삭제전

RN	RNO	REPLY	REPLYER	REPLYDATE	UPDATEDATE
1	5	댓글 테스트 5	replyer5	18/04/14	18/04/14
2	10	Update Reply	replyer10	18/04/14	18/04/14
3	21	JS Test	tester	18/04/14	18/04/14
4	22	JS Test	tester	18/04/14	18/04/14
5	23	JS Test	tester	18/04/14	18/04/14

삭제후

RN	RNO	REPLY	REPLYER	REPLYDATE	UPDATEDATE
1	5	댓글 테스트 5	replyer5	18/04/14	18/04/14
2	10	Update Reply	replyer10	18/04/14	18/04/14
3	21	JS Test	tester	18/04/14	18/04/14
4	22	JS Test	tester	18/04/14	18/04/14

댓글 수정/테스트

```
function update(reply, callback, error) {
     $.ajax({
      type : 'put',
      url : '/replies/' + reply.rno,
      data : JSON.stringify(reply),
      contentType : "application/json; charset=utf-8",
      success : function(result, status, xhr) {
        if (callback) {
          callback(result);
        }
      },
      error : function(xhr, status, er) {
        if (error) {
          error(er);
        }
      }
    });
  }
```

PUT방식으로 호출
전달하는 데이터는 JSON데이터

```
//22번 댓글 수정
replyService.update({
  rno : 22,
  bno : bnoValue,
  reply : "Modified Reply...."
}, function(result) {
  alert("수정 완료...");
});
```

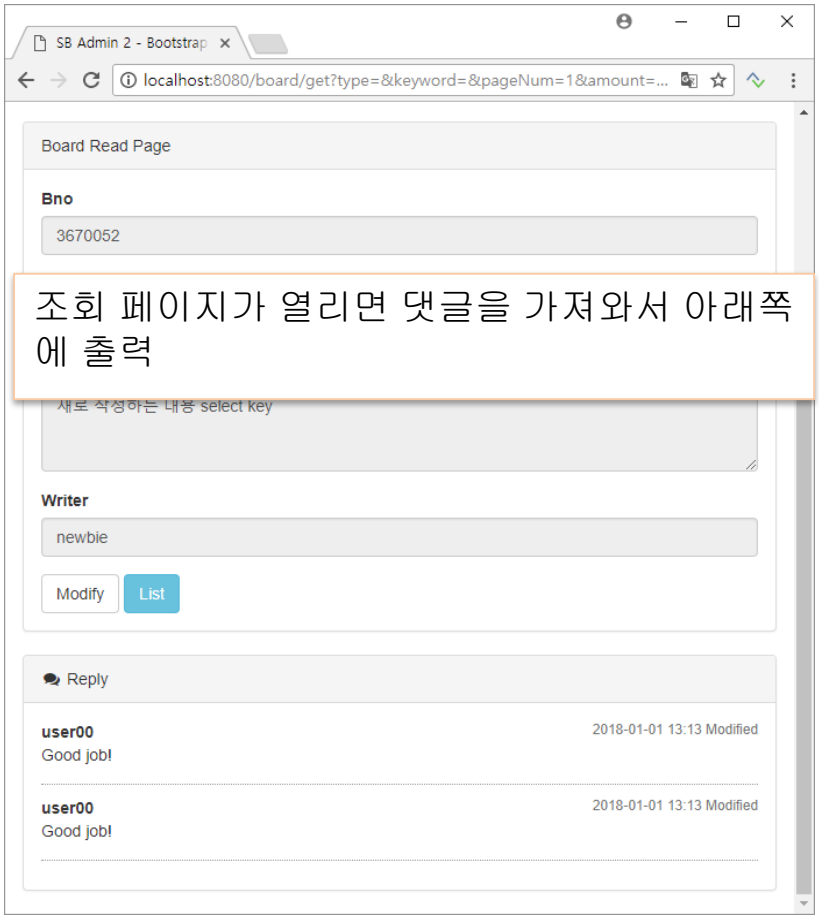
특정댓글조회/테스트

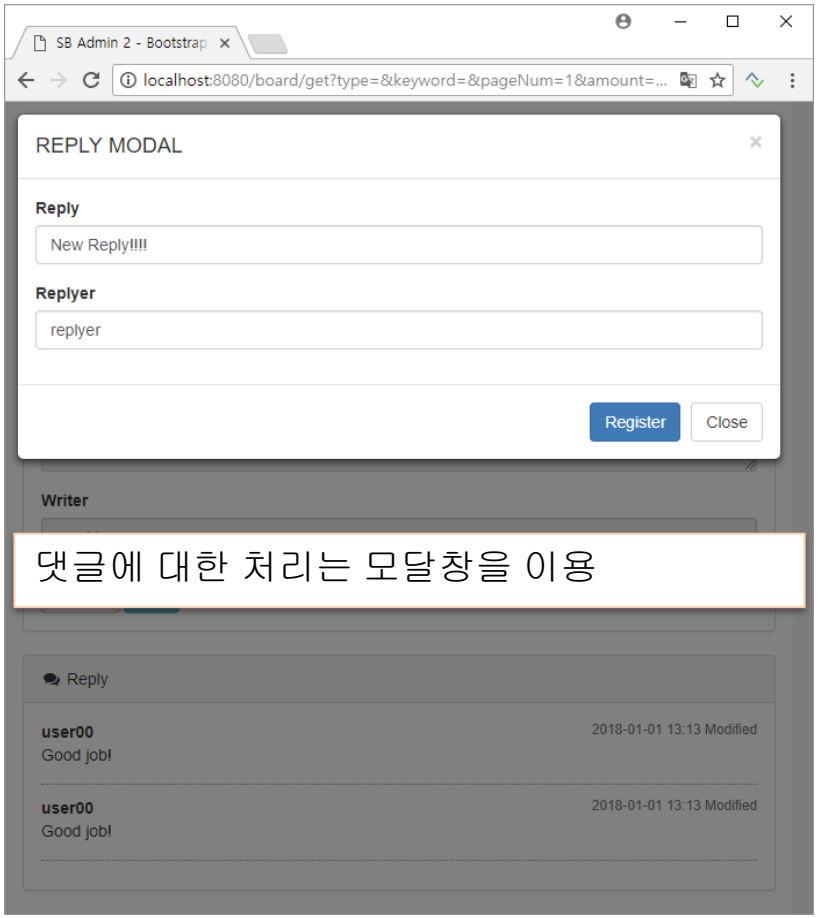
```
function get(rno, callback, error) {
    $.get("/replies/" + rno + ".json", function(result) {
      if (callback) {
        callback(result);
      }
    }).fail(function(xhr, status, err) {
      if (error) {
        error();
      }
    });
  }
```

```
replyService.get(10, function(data){
   console.log(data);
 });
```

이벤트 처리와 HTML처리

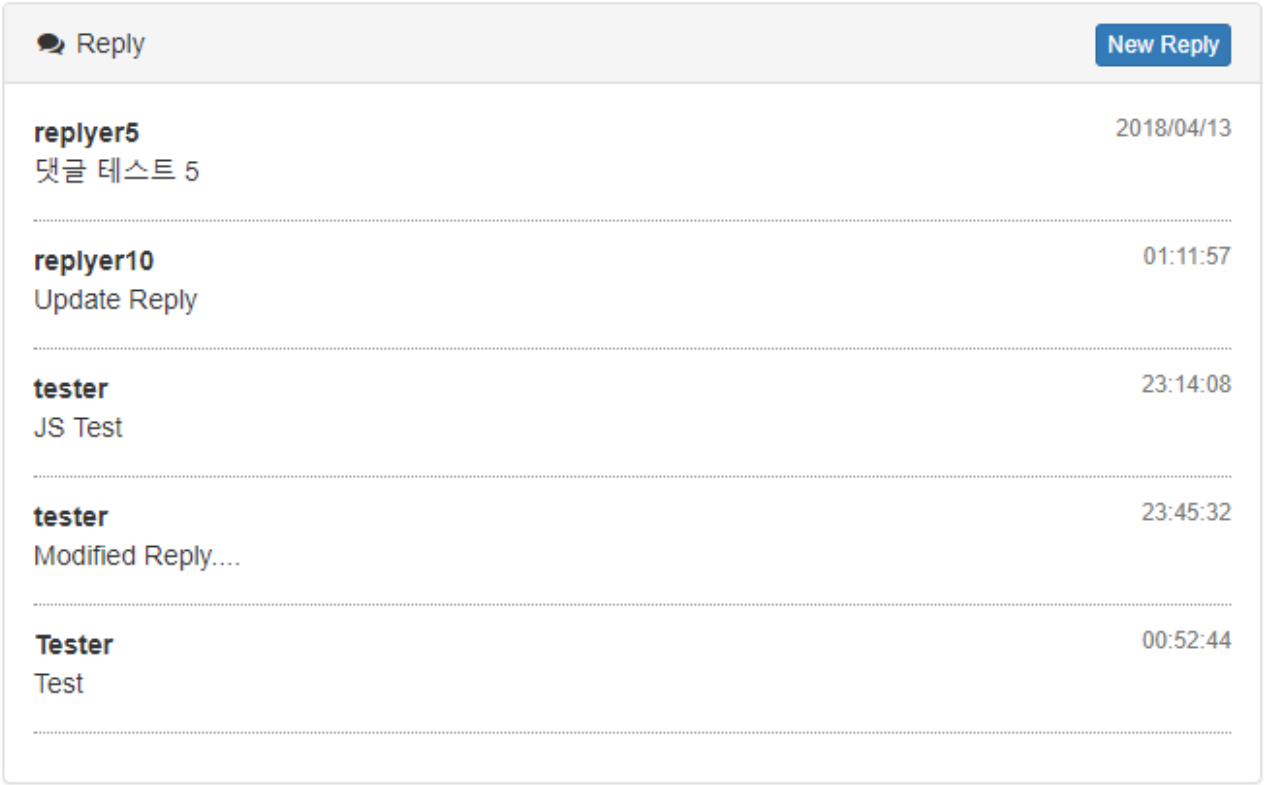
- Ajax호출 이벤트 처리와 후 처리





댓글 목록의 처리

- 댓글의 목록은 <ul> 태그 내에 <li> 태그를 이용해서 처리
- 태그는 하나의 댓글을 의미
- 수정이나 삭제 시에는 반드시 댓글의 번호(rno)가 필요하므로 'data-rno' 속성을 이용

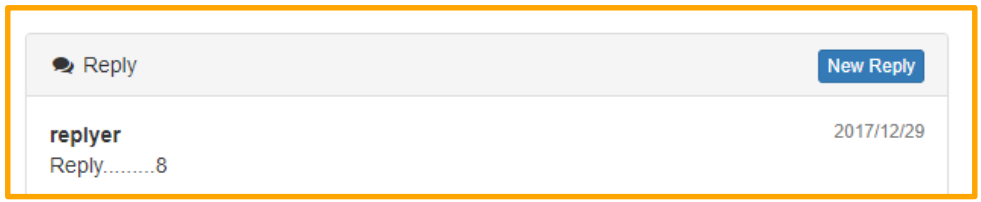


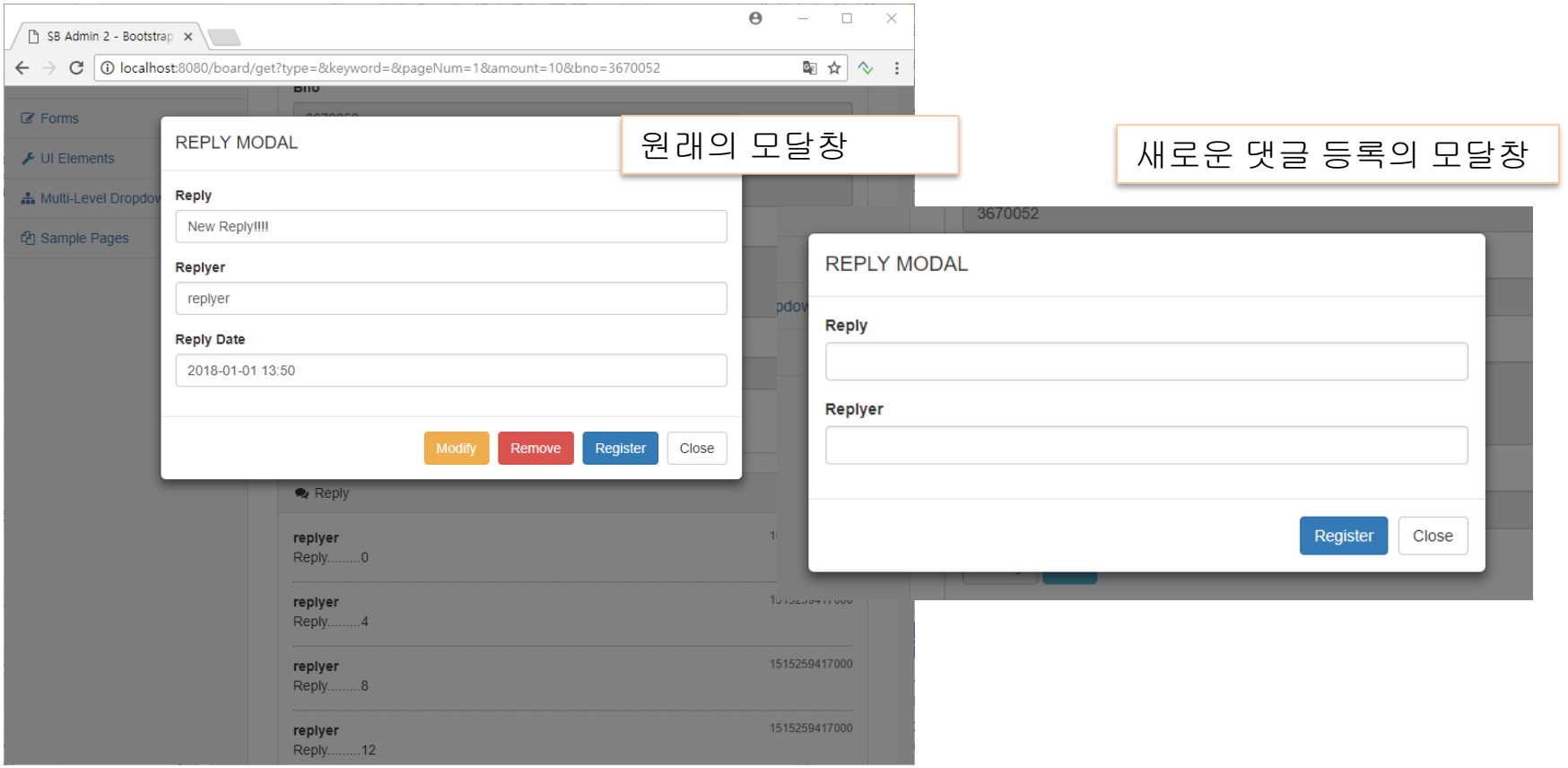
```
function showList(page){
      replyService.getList({bno:bnoValue,page: page|| 1 }, function(list) {
        var str="";
       if(list == null || list.length == 0){
        replyUL.html("");
        return;
      }
      for (var i = 0, len = list.length || 0; i < len; i++) {
          str +="<li class='left clearfix' data-rno='"+list[i].rno+"'>";
          str +="  <div><div class='header'><strong class='primary-
font'>"+list[i].replyer+"</strong>";
          str +="    <small class='pull-right text-
muted'>"+list[i].replyDate+"</small></div>";
          str +="    <p>"+list[i].reply+"</p></div></li>";
        }

    replyUL.html(str);
       });//end function
   }//end showList
```

showList(페이지번호)는 해당 게시글의 댓글을 가져온 후 태그를 만들어서 화면에 보여준다.

새로운 댓글의 처리

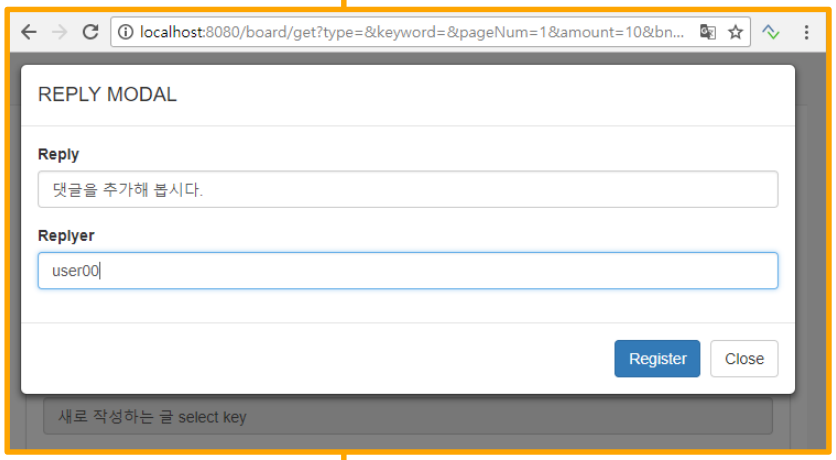
- 모달창을 이용해서 댓글 추가

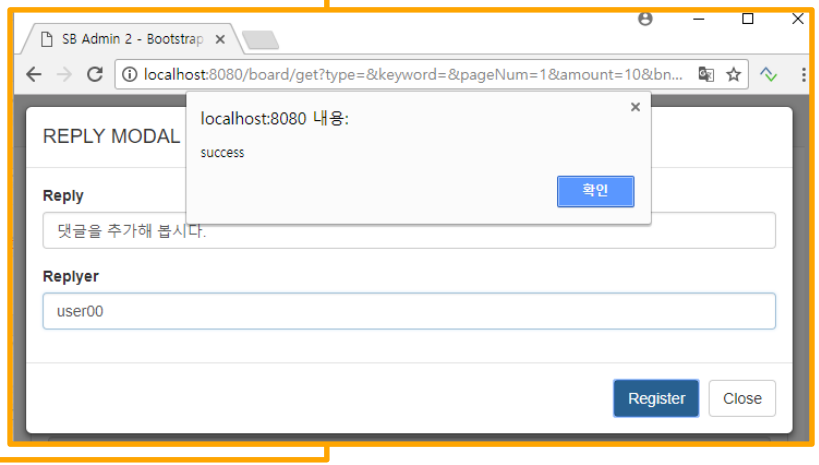




댓글 추가후 처리

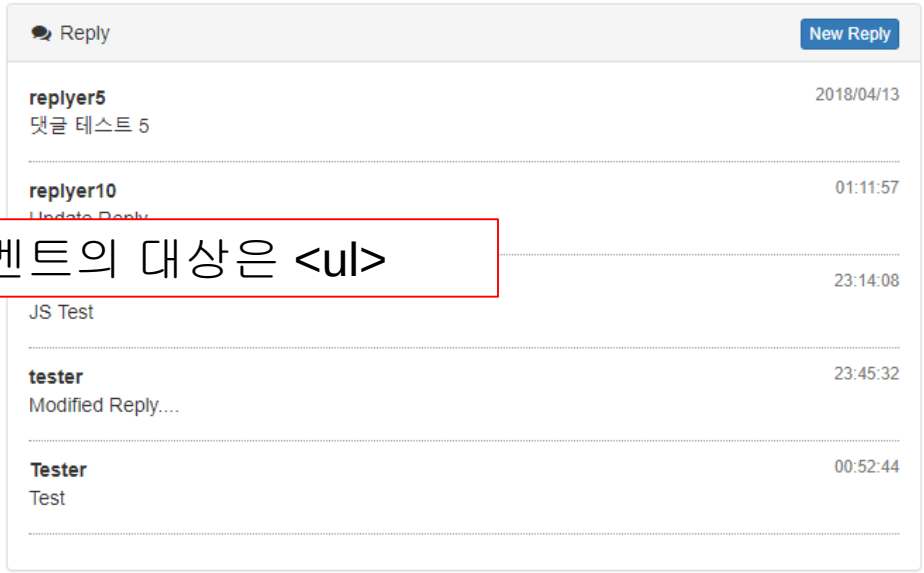
```
modalRegisterBtn.on("click",function(e){
      var reply = {
            reply: modalInputReply.val(),
            replyer:modalInputReplyer.val(),
            bno:bnoValue
          };
      replyService.add(reply, function(result){

        alert(result);

        modal.find("input").val("");
        modal.modal("hide");

      });
  });
```





특정 댓글의 클릭 이벤트

- 댓글은 Ajax로 가져온 결과를 DOM에 추가하는 형태이므로 이벤트 위임(delegation) 방식을 이용해서 처리

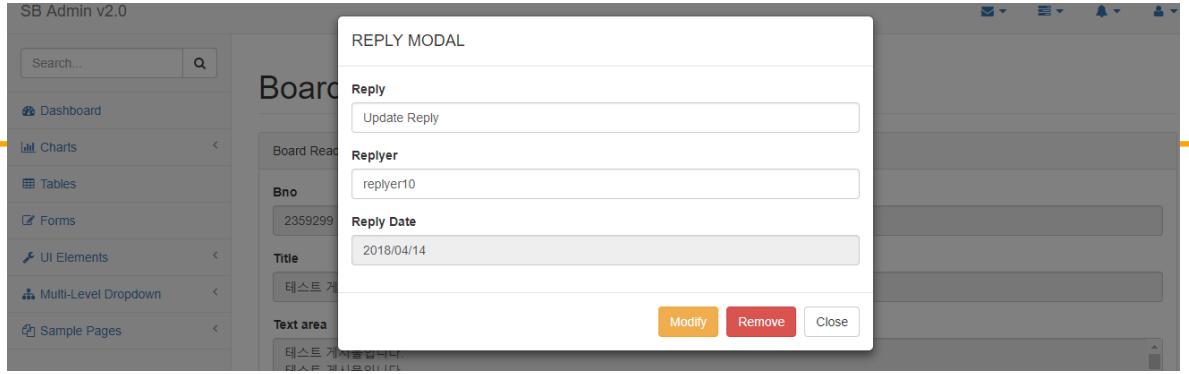


이벤트의 대상은

이벤트 위임으로 가 이벤트의 주인공

```
JS TEST
5
10
21
22
24
```

```
//댓글 조회 클릭 이벤트 처리
    $(".chat").on("click", "li", function(e){
      var rno = $(this).data("rno");
      replyService.get(rno, function(reply){
        modalInputReply.val(reply.reply);
        modalInputReplyer.val(reply.replyer);
        modalInputReplyDate.val(replyService.displayTime( reply.replyDate))
        .attr("readonly","readonly");
        modal.data("rno", reply.rno);
        modal.find("button[id !='modalCloseBtn']").hide();
        modalModBtn.show();
        modalRemoveBtn.show();
        $(".modal").modal("show");
      });
});
```



댓글의 수정/삭제 처리 이벤트

- 수정/삭제 처리는 Ajax로 하고, 모달창 close
- 수정/삭제 후에는 다시 최신 댓글 목록 갱신

```
modalModBtn.on("click", function(e){
      var reply = {rno:modal.data("rno"), reply: modalInputReply.val()};
      replyService.update(reply, function(result){
        alert(result);
        modal.modal("hide");
        showList(1);
      });
  });
```

댓글의 페이징처리

- 데이터베이스의 인덱스 설계
  - 모든 댓글의 조회는 게시물 번호를 이용해서 처리하므로, 게시물 번호의 정렬된 구조가 필요

IDX_REPLY

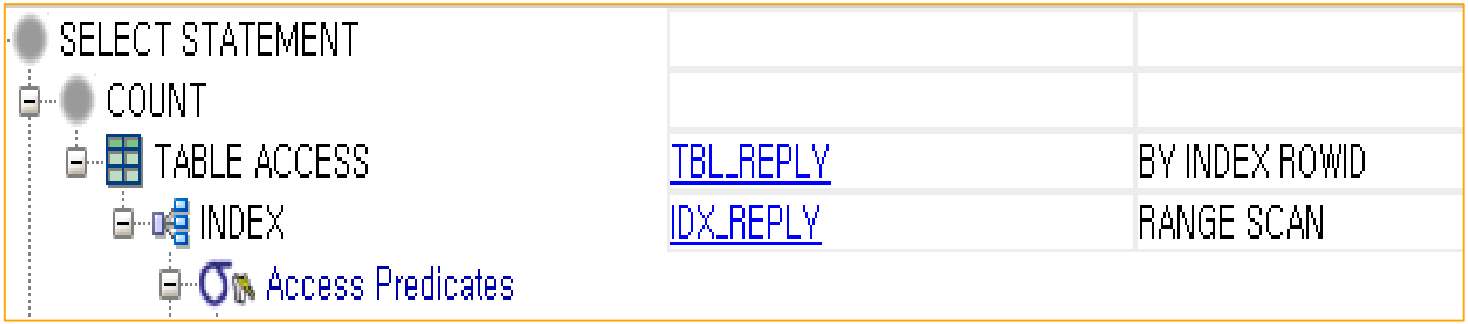
bno	rno	ROWID
300	200	AAAXXK
200	3	AAAXXB
200	13	AAAXXD
200	22	AAAXXE
200	43	AAAXXF
200	89	AAAXXG
200	100	AAAXXH
100	1	AAAXXA
100	5	AAAXXC
100	123	AAAXXC

TBL_REPLY

ROWID	rno	bno	reply
AAAXXA	1	100	xxx
AAAXXE	22	200	xxx
AAAXXD	13	200	xxx
AAAXXB	3	200	xxx
AAAXXF	43	200	xxx
AAAXXC	5	100	xxx
AAAXXG	89	200	xxx
AAAXXH	100	200	xxx
AAAXXK	200	300	xxx
AAAXXI	123	100	xxx

인덱스를 이용하는 댓글 페이징 처리

```
select /*+INDEX(tbl_reply idx_reply) */
    rownum rn, bno, rno, reply, replyer, replyDate, updatedate
    from tbl_reply
    where bno = 3145745(게시물 번호)
    and rno > 0;
```



댓글의 수(카운트)

- 댓글 페이징을 위한 숫자 파악

```
<select id="getCountByBno" resultType="int">
<![CDATA[
select count(rno) from tbl_reply where bno = #{bno}
]]>
</select>
```

ReplyServiceImpl처리

- 댓글의 숫자와 댓글 목록을 처리하는 ReplyPageDTO 클래스

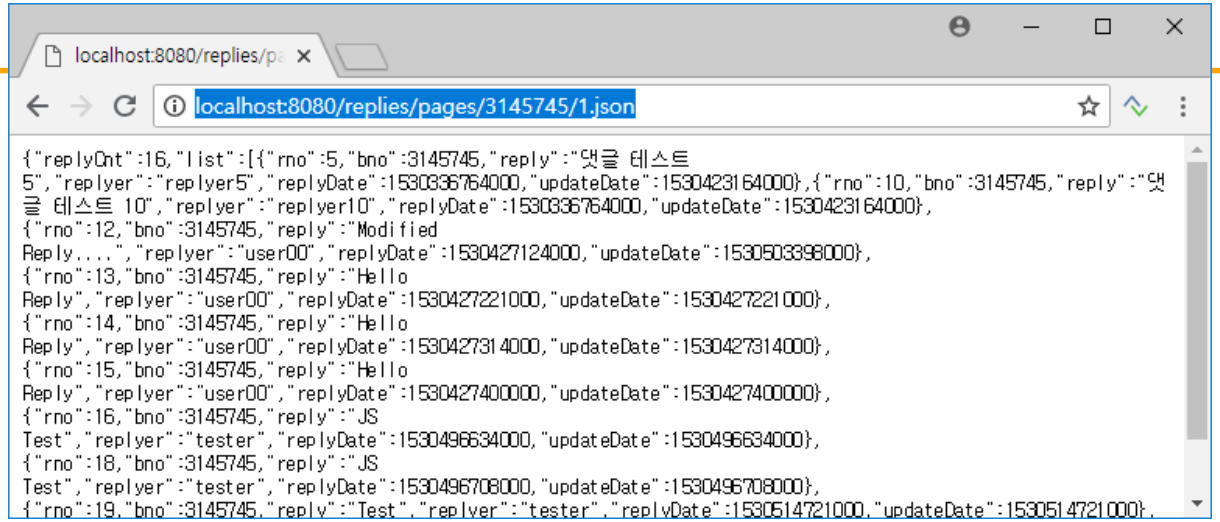
```
@Data
@AllArgsConstructor
@Getter
public class ReplyPageDTO {
  private int replyCnt;
  private List<ReplyVO> list;
}
```

```
public interface ReplyService {
...생략...
    public ReplyPageDTO getListPage(Criteria cri, Long bno);
}
```

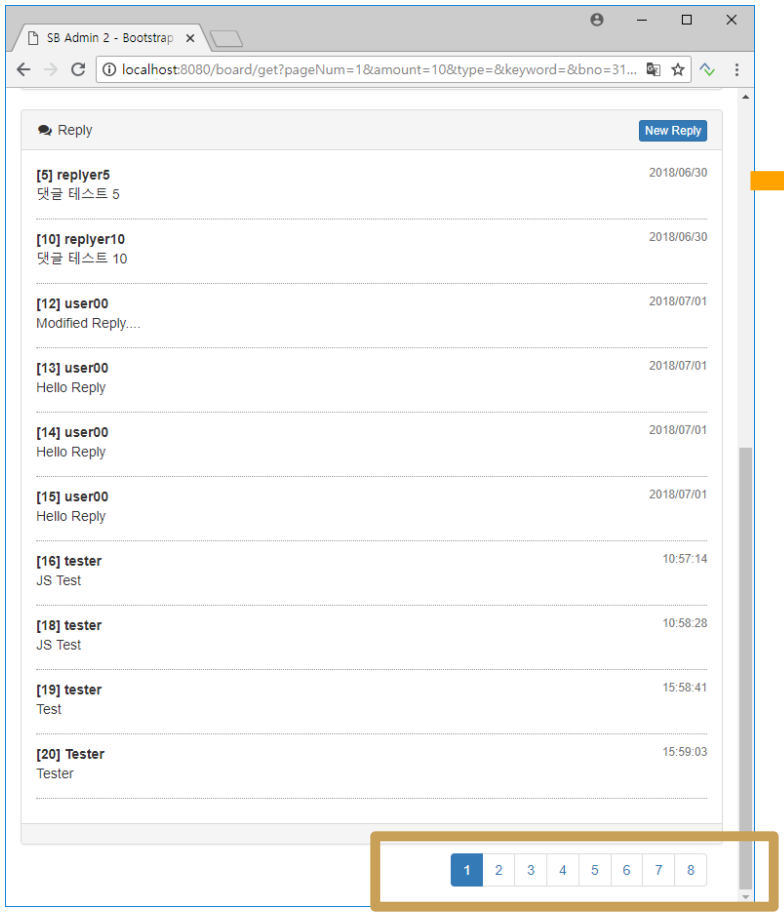
```
@Service
@Log4j
public class ReplyServiceImpl implements ReplyService {
  @Setter(onMethod_ = @Autowired)
  private ReplyMapper mapper;
  ...생략...
  @Override
  public ReplyPageDTO getListPage(Criteria cri, Long bno) {
    return new ReplyPageDTO(
        mapper.getCountByBno(bno),
        mapper.getListWithPaging(cri, bno));
  }
}
```

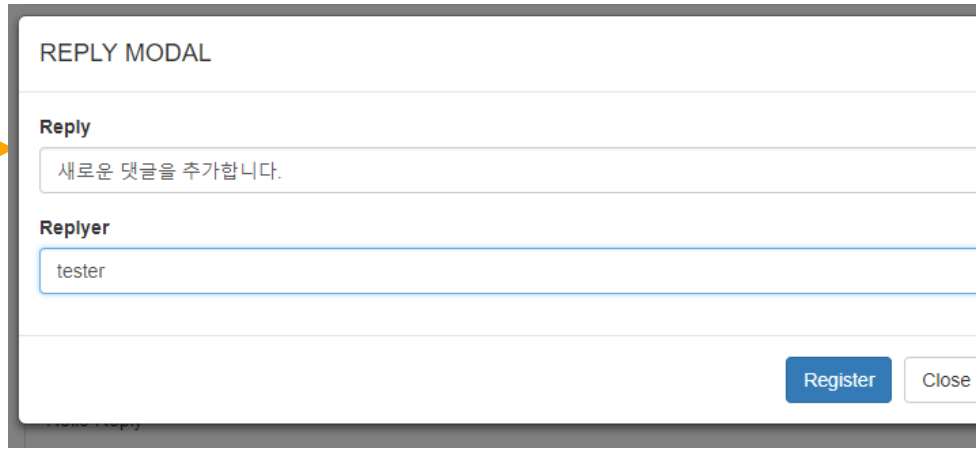
ReplyController의 수정

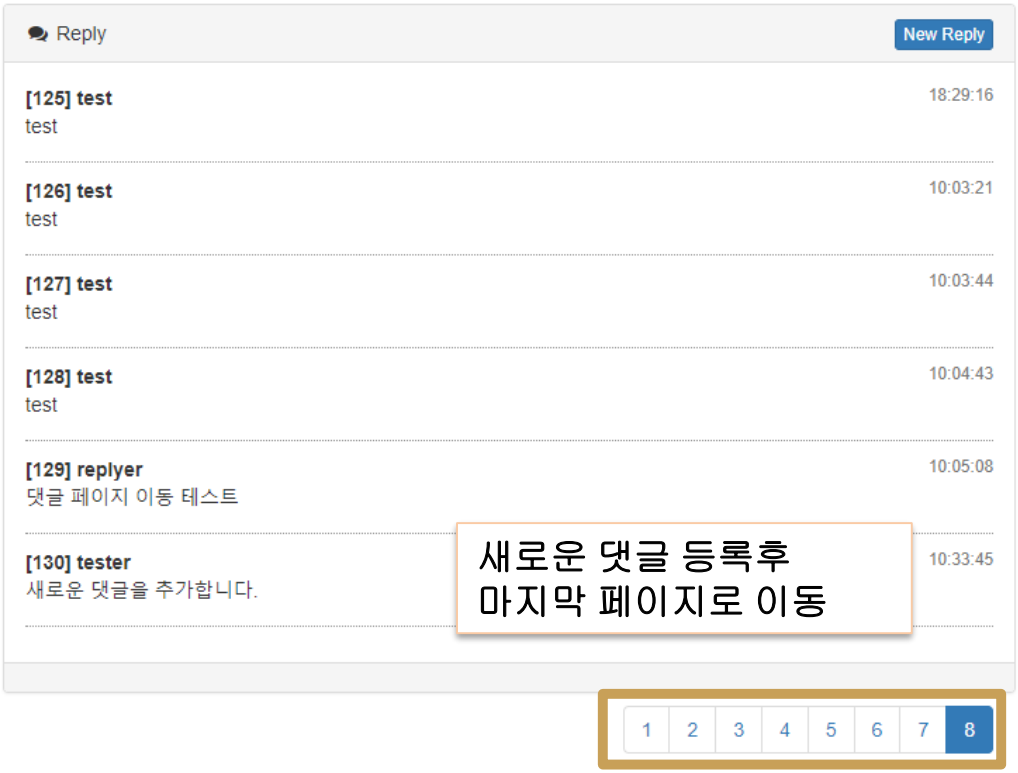
```
 @GetMapping(value = "/pages/{bno}/{page}",
      produces = { MediaType.APPLICATION_XML_VALUE,
      MediaType.APPLICATION_JSON_UTF8_VALUE })
  public ResponseEntity<ReplyPageDTO> getList(@PathVariable("page") int page,
@PathVariable("bno") Long bno) {
    Criteria cri = new Criteria(page, 10);
    log.info("get Reply List bno: " + bno);
    log.info("cri:" + cri);
    return new ResponseEntity<>(service.getListPage(cri, bno), HttpStatus.OK);
  }
```



댓글의 화면 처리


SB Admin 2 - Bootstrap
localhost:8080/board/get?pageNum=1&amount=10&type=&keyword=&bno=31...
Reply
New Reply
[5] replyer5
댓글 테스트 5
2018/06/30
[10] replyer10
댓글 테스트 10
2018/06/30
[12] user00
Modified Reply....
2018/07/01
[13] user00
Hello Reply
2018/07/01
[14] user00
Hello Reply
2018/07/01
[15] user00
Hello Reply
2018/07/01
[16] tester
JS Test
10:57:14
[18] tester
JS Test
10:58:28
[19] tester
Test
15:58:41
[20] Tester
Tester
15:59:03
1 2 3 4 5 6 7 8


REPLY MODAL
Reply
새로운 댓글을 추가합니다.
Replyer
tester
Register
Close


Reply
New Reply
[125] test
test
18:29:16
[126] test
test
10:03:21
[127] test
test
10:03:44
[128] test
test
10:04:43
[129] replyer
댓글 페이지 이동 테스트
10:05:08
[130] tester
새로운 댓글을 추가합니다.
10:33:45
새로운 댓글 등록후
마지막 페이지로 이동
1 2 3 4 5 6 7 8

댓글의 페이지 계산과 출력

- reply.js 는 게시물의 번호와 페이지를 사용하도록 수정

```
function getList(param, callback, error) {
   var bno = param.bno;
   var page = param.page || 1;
   $.getJSON("/replies/pages/" + bno + "/" + page + ".json",
       function(data) {
         if (callback) {
           //callback(data); // 댓글 목록만 가져오는 경우
           callback(data.replyCnt, data.list); //댓글 숫자와 목록을 가져오는 경우
         }
       }).fail(function(xhr, status, err) {
     if (error) {
       error();
     }
   });
 }
```

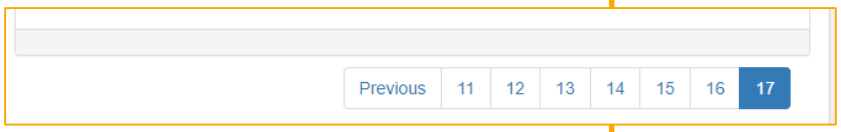
새로운 댓글 추가

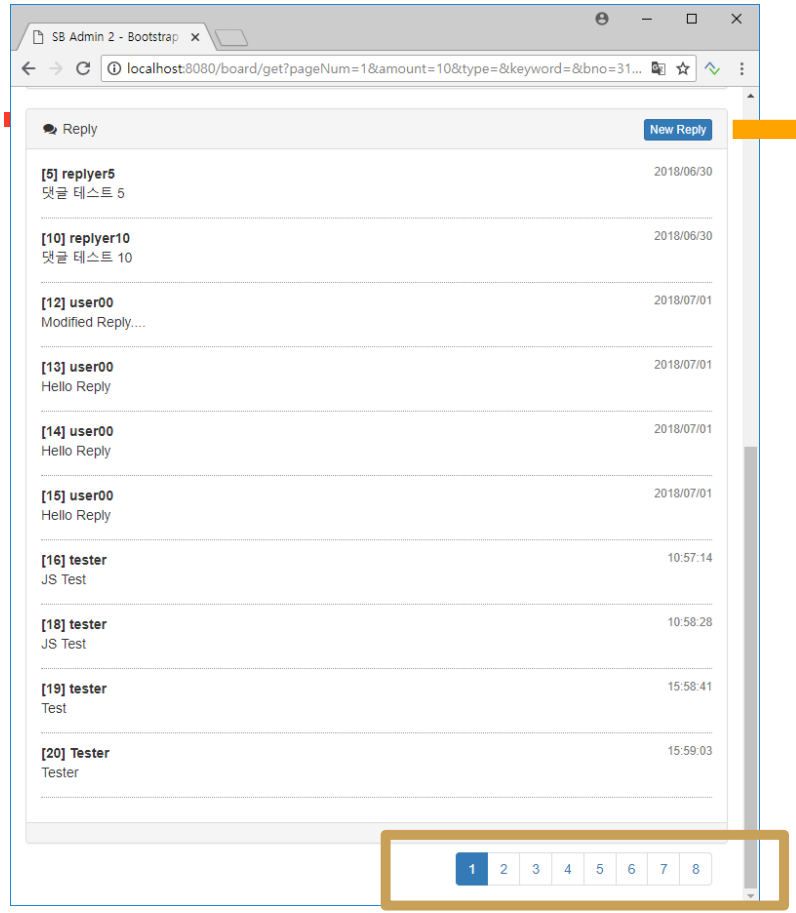
- 새로운 댓글을 추가하면 page값을 -1로 전송하고, 댓글의 전체 숫자를 파악한 후에 페이지 이동

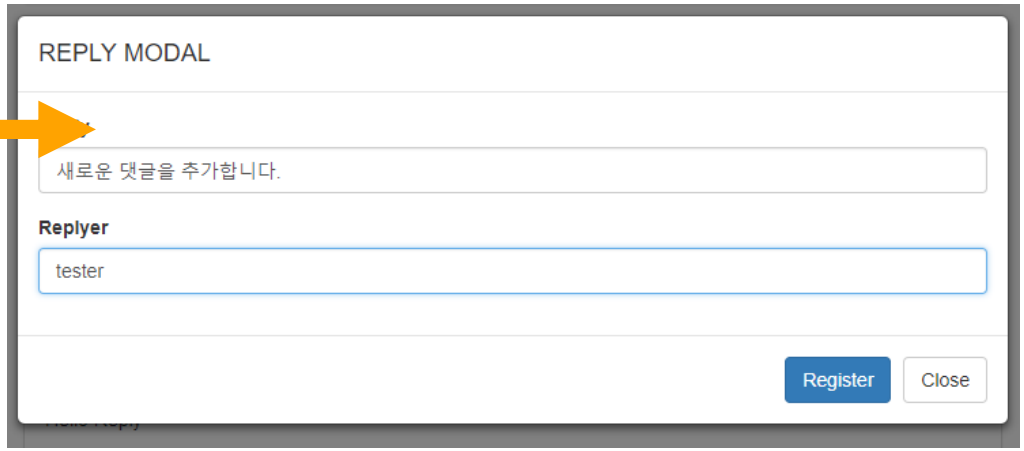
```
modalRegisterBtn.on("click",function(e){
      var reply = {
            reply: modalInputReply.val(),
            replyer:modalInputReplyer.val(),
            bno:bnoValue
          };
      replyService.add(reply, function(result){
        alert(result);
        modal.find("input").val("");
        modal.modal("hide");
        //showList(1);
        showList(-1);
     });
 });
```

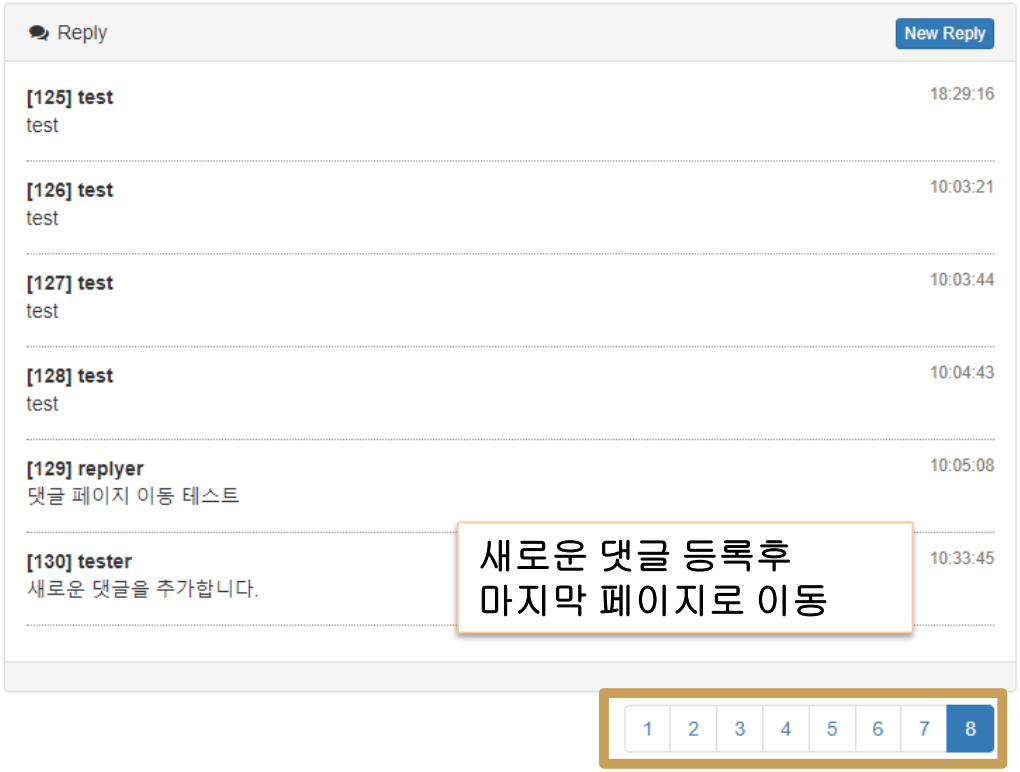
댓글의 페이지 번호 처리

```
<!-- /.panel-heading  기존에 존재하는 부분 -->
<div class="panel-body">
      <ul class="chat">
      </ul>

</div>
<!-- /.panel .chat-panel 추가-->

<div class="panel-footer">

</div>
```




SB Admin 2 - Bootstrap
localhost:8080/board/get?pageNum=1&amount=10&type=&keyword=&bno=31...
Reply
New Reply
[5] replyer5
댓글 테스트 5
2018/06/30
[10] replyer10
댓글 테스트 10
2018/06/30
[12] user00
Modified Reply....
2018/07/01
[13] user00
Hello Reply
2018/07/01
[14] user00
Hello Reply
2018/07/01
[15] user00
Hello Reply
2018/07/01
[16] tester
JS Test
10:57:14
[18] tester
JS Test
10:58:28
[19] tester
Test
15:58:41
[20] Tester
Tester
15:59:03
1 2 3 4 5 6 7 8


REPLY MODAL
새로운 댓글을 추가합니다.
Replyer
tester
Register
Close


Reply
New Reply
[125] test
test
18:29:16
[126] test
test
10:03:21
[127] test
test
10:03:44
[128] test
test
10:04:43
[129] replyer
댓글 페이지 이동 테스트
10:05:08
[130] tester
새로운 댓글을 추가합니다.
10:33:45
새로운 댓글 등록후
마지막 페이지로 이동
1 2 3 4 5 6 7 8