1.REST 소개와 @RestController 애노테이션

가. REST는 'Representational State Transfer' 의 약어로 하나의 URI는 하나의 고유한 리소스를 대표하도록 설계된다는 개념이다. 최근에는 서버에 접근하는 기기의 종류가 다양해 지면서 다양한 기기에서 공통으로 데이터를 처리할 수 있는 규칙을 만들려고 하는데 이러한 시도가 REST 방식이다.

나. REST API는 외부에서 위와 같은 방식으로 특정 URI를 통해서 사용자가 원하는 정보를 제공하는 방식이다. 최근에는 오픈 API에서 많이 사용되면서 REST 방식으로 제공되는 외부 연결 URI를 REST API라고 하고, REST 방식의 서비스 제공이 가능한 것을 'Restful'하다고 표현한다.

다. 스프링 3.0버전부터 @ResponseBody 애노테이션을 지원하면서 본격적으로 REST 방식의 처리를 지원하였고, 4.0부터는 @RestController가 본격적으로 소개되었다.

라. 스프링 4.0버전부터 지원되는 '@RestController' 애노테이션의 경우 기존의 특정한 JSP와 같은 뷰를 만들어 내는 것이 목적이 아닌 REST 방식의 데이터 처리를 위해서 사용하는 애노테이션이다.

마.스프링 3.0버전까지는 컨트롤러는 주로 @Controller 애노테이션만을 사용하였고, 화면 처리는 jsp가 아닌 데이터 자체를 서비스 하려면 해당 메서드나 리턴타입이 '@ResponseBody' 애노테이션을 추가하는 것으로 작성되었다. 기능적인 면에서는 크게 이전 3.0버전과 달라진 점은 없지만, 컨트롤러 자체의 용도를 지정한다는 점에서 변화가 있다고 볼수 있다.

2. 테스트용 @RestController 애노테이션을 사용한 컨트롤러 생성

가. @RestController는 jsp와 같은 뷰페이지를 만들지 않고, 단순 문자열, JSON, XML 같은 데이터 자체를 웹브라우저에 반환한다.

```
package org.zerock.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.zerock.domain.SampleVO;

@RestController //스프링 4.0 이후부터 이 애노테이션으로
// jsp뷰페이지를 직접 만들지 않고, REST 방식의 데이터 처리
```

```
//를 위해서 사용하는 애노테이션이다. 만들어 지는 데이터 객
//체는 문자열,xml,JSON 이다.
@RequestMapping("/sample") //컨트롤 자체에 매핑주소 등록
public class SampleController6 {

	@RequestMapping("/hello")
	// /hello라는 매핑주소를 등록, get or post 처리가능
	public String hello() {
		return "rest begin";//문자열 결과가 반환
	}

	@RequestMapping("/sendVO")
	public SampleVO sendVO() {
		//리턴 타입이 SampleVO이면 변수명이 JSON 객체의
		//키이름이 된다.
	SampleVO vo=new SampleVO();
	vo.setFirstName("홍");
	vo.setLastName("길동");
	vo.setMno(7);

	return vo;
	}

	@RequestMapping("/sendList")
	public List<SampleVO> sendList(){
		List<SampleVO> list=new ArrayList<>();

		for(int i=1;i<=10;i++) {
			SampleVO vo=new SampleVO();
			vo.setMno(i);
			vo.setFirstName("세종");
			vo.setLastName("대왕님");

			list.add(vo);//컬렉션 추가
		}//for
		return list;
	}//sendList()

	//키,값 쌍의 Map타입 JSON
	@RequestMapping("/sendMap")
```

```
	public Map<Integer,SampleVO> sendMap(){
		Map<Integer,SampleVO> map=new HashMap<>();

		for(int i=1;i<=10;i++) {
			SampleVO vo=new SampleVO();

			vo.setMno(i);
			vo.setFirstName("이");
			vo.setLastName("순신");

			map.put(i,vo);//컬렉션에 키,값 저장
		}//for
		return map;
	}//sendMap()

	@RequestMapping("/sendError")
	public ResponseEntity<Void> sendListAuth(){
	return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
/* 1.@RestController 는 별도의 jsp파일을 만들지 않고 Rest
 * 서비스를 실행하기 때문에,결과 데이터에 예외적인 상황에
 * 서 문제가 발생할 수 있다. 스프링에서 제공하는 Response
 * Entity 타입은 개발자가 문제가 되는 나쁜상태 404,500 같
 * 은 Http 나쁜 상태코드를 데이터와 함께 브라우저로 전송
 * 할수 있기 때문에 좀 더 세밀한 제어가 필요한 경우 사용
 * 해 볼수 있다.
 *  400 나쁜 상태코드 BAD_REQUEST가 브라우저로 전송된다.
 */
	}

	//정상적인 json 데이터와 404(자원을 찾지 못했을때)
	//나쁜 상태코드를 함께 브라우저로 전송
	@RequestMapping("/sendErrorNot")
	public ResponseEntity<List<SampleVO>> sendListNot(){
		List<SampleVO> list=new ArrayList<>();

		for(int i=1;i<=10;i++) {
			SampleVO vo=new SampleVO();
			vo.setMno(i);
			vo.setFirstName("홍");
			vo.setLastName("길동");
```

```
                        list.add(vo);
                }
                return new ResponseEntity<List<SampleVO>>(list,
                                HttpStatus.NOT_FOUND);
        }//sendListNot()
}
```

3. org.zerock.domain.SampleVO 데이터 저장빈 클래스 작성

```
package org.zerock.domain;

public class SampleVO {

        private int mno;//변수명이 JSON의 키이름이 된다.
        private String firstName;//성
        private String lastName;//이름

        public int getMno() {
                return mno;
        }
        public void setMno(int mno) {
                this.mno = mno;
        }
        public String getFirstName() {
                return firstName;
        }
        public void setFirstName(String firstName) {
                this.firstName = firstName;
        }
        public String getLastName() {
                return lastName;
        }
        public void setLastName(String lastName) {
                this.lastName = lastName;
        }
}
```

4. Advanced REST Client를 이용한 테스트

가. REST방식은 다양한 디바이스 장치로부터 서버에 데이터와 작업을 요청하고,결과를 받는다는 점에서 유용하지만, 일반적인 웹 페이지와는 달리 화면을 제작하지 않는 형태로 처리되기 때문에, 결과를 체크하면서 개발하기에는 적합하지 않다.

나. 최근에는 여러 종류의 REST 클라이언트 프로그램이 존재한다. 좀더 쉽게 결과를 테스트하기 위해서 크롬 브라우저의 앱으로 존재하는 Advanced REST Client 나 Post Man 등의 프로그램이다.여기서는 가장 많은 사용자가 존재하는 Advanced REST Client를 이용하도록 하겠다.

5. REST 와 AJAX

가. 웹을 통해서 작업할 때 REST방식이 가장 많이 쓰이는 형태가 아작스와 같이 결합된 형태이다. 아작스는 비동기식 프로그램으로 주로 브라우저에서 화면 전환이 없이 서버와 필요한 일정 영역부분만 대화형으로 데이터를 주고받는 형태의 메시지 전송 방식이다.

나. 아작스를 가장 쉽게 설명하면 화면 전환이나 깜빡임 없이 서버에서 보낸 데이터를 받는 방법이다.

다. 아작스의 가장 큰 특징은
첫째, 브라우저의 화면 전환없이 사용자 측면에서 좋다는 점.
둘째, 서버에서 화면에 필요한 모든 데이터를 만드는 대신 서버는 필요한 데이터만 전달하기 때문에, 개발에 무게 중심이 브라우저 쪽으로 많이 배분된다는 점이라고 할 수 있다. 아작스 시작 시점에서는 서버와 메시지 전달에 있어서 일반 문자나 XML을 주로 사용했지만, 최근에는 JSON 형태의 데이터를 사용하는 경우도 많이 증가하고 있다.
REST방식이 데이터를 호출하고, 사용하는 방식을 의미한다면 아작스는 실제로 그를 이용하는 수단에 가깝다고 할 수 있다.

6. 댓글을 위한 테이블 설계

```
--댓글 테이블 작성
create table tbl_reply(
  rno number(38) primary key --댓글 번호
  ,bno number(38) default 0 --게시판 번호값이 저장되는 컬
  --럼, 외래키 제약조건으로 추가 설정,tbl_board테이블의
  --게시물 번호값만 저장
  ,replyer varchar2(100) not null --댓글 작성자
  ,replytext varchar2(4000) not null --댓글내용
  ,regdate date --등록날짜
  ,updatedate date --수정날짜
);
select * from tbl_reply order by rno desc;

--외래키 설정
alter table tbl_reply add constraint fk_board
```

```
foreign key(bno) references tbl_board(bno);
--foreign key(외래키)

--댓글 시퀀스 생성
create sequence rno_seq
start with 1
increment by 1
nocache;

--dual 테이블에서 시퀀스 번호값 확인
select rno_seq.nextval from dual;
```

7. 댓글을 위한 도메인 데이터 저장빈 클래스 작성

```
package org.zerock.domain;

public class ReplyVO {

	/*
	 * tbl_reply 테이블의 컬럼명과 빈클래스 변수명을 같게
	 * 한다.
	 */
	private int rno;//댓글번호
	private int bno;//게시판 번호
	private String replyer;//댓글 작성자
	private String replytext;//댓글내용
	private String regdate;//등록날짜
	private String updatedate;//수정날짜

	public int getRno() {
		return rno;
	}
	public void setRno(int rno) {
		this.rno = rno;
	}
	public int getBno() {
		return bno;
	}
	public void setBno(int bno) {
		this.bno = bno;
	}
	public String getReplyer() {
```

```
                return replyer;
        }
        public void setReplyer(String replyer) {
                this.replyer = replyer;
        }
        public String getReplytext() {
                return replytext;
        }
        public void setReplytext(String replytext) {
                this.replytext = replytext;
        }
        public String getRegdate() {
                return regdate;
        }
        public void setRegdate(String regdate) {
                this.regdate = regdate;
        }
        public String getUpdatedate() {
                return updatedate;
        }
        public void setUpdatedate(String updatedate) {
                this.updatedate = updatedate;
        }
}
```

8. ReplyDAO 인터페이스 작성

```
package org.zerock.persistence;

import java.util.List;

import org.zerock.domain.ReplyVO;

public interface ReplyDAO {

        void addReply(ReplyVO vo);
        List<ReplyVO> list(int bno);
        void updateReply(ReplyVO vo);
        void delReply(int rno);
}
```

9. ReplyDAOImpl.java 작성

```
package org.zerock.persistence;

import java.util.List;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.zerock.domain.ReplyVO;

@Repository
public class ReplyDAOImpl implements ReplyDAO {

	@Autowired
	private SqlSession sqlSession;

	@Override
	public void addReply(ReplyVO vo) {
		this.sqlSession.insert("reply_in",vo);
		//reply_in은 매퍼태그의 insert 아이디명.
	}//댓글등록

	@Override
	public List<ReplyVO> list(int bno) {
		return this.sqlSession.selectList("reply_list",
					bno);
	}//댓글 목록

	@Override
	public void updateReply(ReplyVO vo) {
		this.sqlSession.update("reply_edit",vo);
	}//댓글수정

	@Override
	public void delReply(int rno) {
		this.sqlSession.delete("reply_del",rno);
	}//댓글 삭제
}
```

10. reply.xml 매퍼태그 작성

```
<?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="Reply">

   <!-- 댓글 추가 -->
   <insert id="reply_in">
    insert into tbl_reply (rno,bno,replyer,replytext,
    regdate) values(rno_seq.nextval,#{bno},#{replyer},
    #{replytext},sysdate)
   </insert>

   <!-- 댓글목록 -->
   <select id="reply_list" resultType="r">
   <!-- resultType 속성은 반환타입 -->
    select * from tbl_reply where bno=#{bno} order by
    rno desc
   </select>

   <!-- 댓글 수정 -->
   <update id="reply_edit">
    update tbl_reply set replytext=#{replytext},
    updatedate=sysdate where rno=#{rno}
   </update>

   <!-- 댓글 삭제 -->
   <delete id="reply_del">
    delete from tbl_reply where rno=#{rno}
   </delete>
 </mapper>
```

11. ReplyService.java 인터페이스 작성

```
package org.zerock.service;

import java.util.List;

import org.zerock.domain.ReplyVO;
```

```
public interface ReplyService {

	void addReply(ReplyVO vo);
	List<ReplyVO> listReply(int bno);
	void updateReply(ReplyVO vo);
	void remove(int rno);
}
```

12. ReplyServiceImpl.java 작성

```
package org.zerock.service;

import java.util.List;

import javax.inject.Inject;

import org.springframework.stereotype.Service;
import org.zerock.domain.ReplyVO;
import org.zerock.persistence.ReplyDAO;

@Service
public class ReplyServiceImpl implements ReplyService {

	@Inject
	private ReplyDAO replyDAO;//자동 의존성 주입.

	@Override
	public void addReply(ReplyVO vo) {
		this.replyDAO.addReply(vo);
	}

	@Override
	public List<ReplyVO> listReply(int bno) {
		return this.replyDAO.list(bno);
	}

	@Override
	public void updateReply(ReplyVO vo) {
		this.replyDAO.updateReply(vo);
	}

	@Override
```

```
	public void remove(int rno) {
		this.replyDAO.delReply(rno);
	}
}
```

13. REST 방식의 ReplyController 작성

```
package org.zerock.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.zerock.domain.ReplyVO;
import org.zerock.service.ReplyService;

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

	@Autowired
	private ReplyService replyService;

	//댓글등록
	@RequestMapping(value="",method=RequestMethod.POST)
	//post방식으로 접근하는 매핑주소를 처리하는 애노테이션
	public ResponseEntity<String> register(
				@RequestBody ReplyVO vo){
//@RequestBody 애노테이션은 전송된 JSON 데이터를 객체로
//변환한다. 데이터 전송 방식을 json을 이용한다.json으로 보
//내진 데이터를 ReplyVO 타입으로 바꿔준다.
		ResponseEntity<String> entity=null;
		try {
			this.replyService.addReply(vo);//댓글등록
entity=new ResponseEntity<String>("SUCCESS",
		HttpStatus.OK);//댓글저장 성공시 정상코드 200반환
```

```
//HTTP상태코드가 반환되면서 SUCCESS문자열을 반환.
                }catch(Exception e) {
                        e.printStackTrace();
                        entity=new ResponseEntity<String>(
                                e.getMessage(),HttpStatus.BAD_REQUEST);
//예외 에러가 발생했을때 나쁜 상태코드 문자열이 반환.
                }
                return entity;
        }//register()

        //게시물 번호에 해당하는 댓글목록
        @RequestMapping(value="/all/{bno}",
                        method=RequestMethod.GET)
        //get방식으로 접근하는 매핑주소를 처리
        public ResponseEntity<List<ReplyVO>> list(
                        @PathVariable("bno") int bno){
//@PathVariable 은 매핑주소의 게시물번호값을 추출하는 용도
//로 활용
                ResponseEntity<List<ReplyVO>> entity=null;
                try {
                entity=new ResponseEntity<>(
                                this.replyService.listReply(bno),
                                HttpStatus.OK);
//게시물 번호에 해당하는 댓글목록을 반환
                }catch(Exception e) {
                        e.printStackTrace();
   entity=new ResponseEntity<>(HttpStatus.BAD_REQUEST);
                }
                return entity;
        }//list()

        //댓글수정
        @RequestMapping(value="/{rno}",
        method= {RequestMethod.PUT,RequestMethod.PATCH})
        //PUT은 전체자료를 수정,PATCH는 일부 자료만 수정
        public ResponseEntity<String> update(
                        @PathVariable("rno") int rno,
                        @RequestBody ReplyVO vo){
                ResponseEntity<String> entity=null;
```

```
                try {
                        vo.setRno(rno);//댓글번호 저장
                        this.replyService.updateReply(vo);//댓글수정
entity=new ResponseEntity<String>("SUCCESS",
                HttpStatus.OK);
                }catch(Exception e) {
                        e.printStackTrace();
entity=new ResponseEntity<String>(e.getMessage(),
                HttpStatus.BAD_REQUEST);
                }
                return entity;
        }//update()

        //댓글 삭제
        @RequestMapping(value="{rno}",
                        method=RequestMethod.DELETE)
        public ResponseEntity<String> remove(
                        @PathVariable("rno") int rno){
                ResponseEntity<String> entity=null;

                try {
                        this.replyService.remove(rno);//댓글 삭제
entity=new ResponseEntity<String>("SUCCESS",
                HttpStatus.OK);
                }catch(Exception e) {
                        e.printStackTrace();
entity=new ResponseEntity<>(e.getMessage(),
                HttpStatus.BAD_REQUEST);
                }
                return entity;
        }//remove()
}
```

14. HomeController를 이용한 test.jsp

```
@Controller
public class HomeController {//아작스 댓글목록
        @RequestMapping("/test") // /test 매핑주소 등록
        public void test() {
                //메서드 리턴타입이 없어면 매핑주소가 /test가
                //jsp파일명이 된다. 즉 test.jsp
        }
```

```
}
```

test.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>아작스 댓글연습 </title>
<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;/* position 속성이 absolute 나 fixed 로 설정된 곳에서 사용한다. 이 속성은 요소가 겹쳐지는
          순서를 제어할 수 있다. 값이 큰것이 앞에 나온다. */
 }
</style>
</head>
<body>
<%-- 댓글 수정 폼화면 --%>
 <div id="modDiv" style="display:none;">
    <div class="modal-title"></div>
    <div>
     <textarea rows="3" cols="30"
     id="replytext"></textarea>
    </div>
    <div>
 <button type="button" id="replyModBtn">댓글수정</button>
 <button type="button" id="replyDelBtn">댓글삭제</button>
 <button type="button" id="closeBtn"
 onclick="modDivClose();">닫기</button>
    </div>
 </div>
```

```
<h2>아작스 연습 페이지</h2>
<div>
 <div>
  댓글 작성자:<input type="text" name="replyer"
  id="newReplyWriter" />
 </div>
 <br/>
 <div>
  댓글 내용:<textarea rows="5" cols="30"
  name="replytext" id="newReplyText"></textarea>
 </div>
 <br/>
 <button id="replyAddBtn">댓글등록</button>
</div>
<br/>
<hr/>
<br/>
<%--댓글목록 --%>
<ul id="replies"></ul>

<%-- jQuery 라이브러리 CDN 방식 --%>
<script src="https://code.jquery.com/jquery-latest.min.js"></script>
<script>
  var bno=14;//게시판 번호

  getAllList();//댓글 목록함수를 호출
  function getAllList(){
          $.getJSON("/controller/replies/all/"+bno,
                  function(data){
                  var str="";
                  $(data).each(function(){//each() 함수는 댓글목록
                          //을 반복
                          str += "<li data-rno='"+this.rno+"' class='replyLi'>"
                                  + this.rno+ " : <span class='com'
style='color:blue;font-weight:bold;'>"+ this.replytext + "</span>"
                                  +" <button>댓글수정</button></li><br/>";

                  });
                  $("#replies").html(str);
```

```
        });
}//getAllList()

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

                $.ajax({
                        type : 'post',
                        url : '/controller/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();//댓글목록 함수
                                }
                                }
                        });
});

//댓글 수정 화면
$("#replies").on("click",".replyLi button",function(){
          var reply = $(this).parent();
              var rno = reply.attr("data-rno");//댓글번호
              var replytext = reply.text();//댓글 내용

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

```
//div 닫기
function modDivClose(){
	$("#modDiv").hide("slow");
}

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

	$.ajax({
			type:'put',
			url:'/controller/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);
					}
			}});
});

//댓글 삭제
$("#replyDelBtn").on("click",function(){
	var rno = $(".modal-title").html();

		$.ajax({
			type : 'delete',
			url : '/controller/replies/' + rno,//삭제 매핑 주소
			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.");
                                        $("#modDiv").hide("slow");
                                        getAllList();
                                }
                        }
                });
  });
 </script>
</body>
</html>
```

15. 게시물 내용 밑에 댓글 기능 추가

게시물 내용 소스 생략

```
...
<br/>
 <hr/>
 <br/>

  <%-- 댓글기능추가--%>
  <div id='modDiv' style="display: none">
<div class='modal-title'></div>
<div>
<textarea rows="3" cols="30" id='replytext'></textarea>
</div>
<div>
<button type="button" id="replyModBtn">댓글수정</button>
<button type="button" id="replyDelBtn">댓글삭제</button>
<button type="button" id='closeBtn' onclick="modDivClose();">닫기</button>
</div>
</div>

<h2>아작스연습페이지</h2>

<div>
<div>
댓글작성자: <input type="text" name="replyer" id="newReplyWriter">
```

```
</div>
<br/>
<div>
댓글내용: <textarea rows="5" cols="30"
name="replytext" id="newReplyText"></textarea>
</div>
<br/>
<button id="replyAddBtn">댓글등록</button>
</div>
    <br/>
    <hr/>
    <br/>
    <%--댓글목록--%>
<ul id="replies"></ul>

<script>
var bno = ${b.bno};
//var bno=14;

getAllList();

//댓글목록
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+ " : <span class='com' style='color:blue;font-weight:bold;'>"+
this.replytext+ "</span>"
+" <button>댓글수정</button></li><br/>"
});
$("#replies").html(str);
});
}//getAllList()

//댓글추가
$("#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("등록되었습니다.");
getAllList();
}
}
});
});//#replyAddBtn

//댓글수정화면
$("#replies").on("click", ".replyLi button", function() {
var reply = $(this).parent();
var rno = reply.attr("data-rno");//댓글번호
var replytext = reply.text();//댓글내용

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

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

```
$.ajax({
type : 'delete',
url : '/replies/' + rno,//삭제매핑주소
headers : {
"Content-Type" : "application/json",
"X-HTTP-Method-Override" : "DELETE"
},
dataType : 'text',
success : function(result) {
console.log("result: " + result);
if (result == 'SUCCESS') {
alert("삭제되었습니다.");
$("#modDiv").hide("slow");
getAllList();
}
}
});
});

//댓글수정완료
$("#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("수정되었습니다.");
$("#modDiv").hide("slow");
getAllList();
// getPageList(replyPage);
```

```
}
}});
});//#replyModBtn


//div 닫기
function modDivClose(){
$("#modDiv").hide("slow");
}
</script>
</body>
</html>
```