CRUD(create read update delete) pattern

Author : Rhio.kim
Date : 2008.07.08
Blog : blog.ecmas4.com
Mail : Rhio.kim@gmail.com
Version : 1.0.0 / Update : 2008.08.18

CRUD(create read update delete) Pattern 정의

모티브 :

웹 페이지에서 일어나는 사용자의 단일 액션에 대응하는 일련의 프로세스를 하나의 클래스에서 구현한다. 일련의 프로세스는 사용자가 서버에 요청을 하기 위해서 클릭을 한다거나 입력을 하고 요청을 하고 그에 따른 서버 측에서 처리가 이루어지고 처리 결과를 다시 사용자의 브라우저에 통보를 하고 브라우저는 결과를 통해 사용자에게 결과를 인식 시키는 일련의 작업을 말합니다.

목적 및 장점 :

1. CRUD(Create, Read, Update, Delete) 인터랙션에 대한 처리와 시스템 장애에 대한 빠른 문제 파악과 대응

조건 :

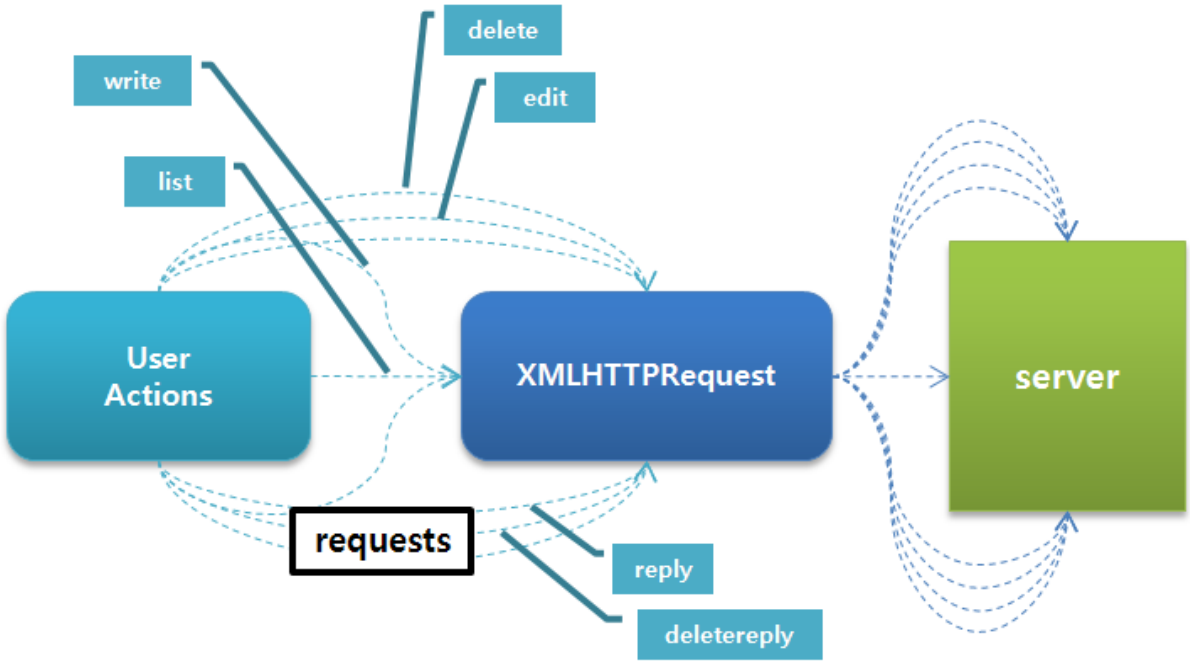
1. XHR Wrapped 클래스가 존재하여야 한다. (prototype.js, dojo, jQuery, etc)
2. XHR 오브젝트를 이용한 데이터 처리가 있어야 한다.
3. 요청을 위한 단계와 응답에 대한 처리 단계가 간단하고 명료해야 한다.

제약 :

1. 복잡한 UI 처리 및 CRUD 이외의 처리가 다소 병행되어 진다면 클래스 혹은 객체가 무거워질 수 있다.

단점 :

1. 특정한 인터랙션 위한 패턴으로 확장(extend) 및 소스 재사용 면에서 용이하지 못함

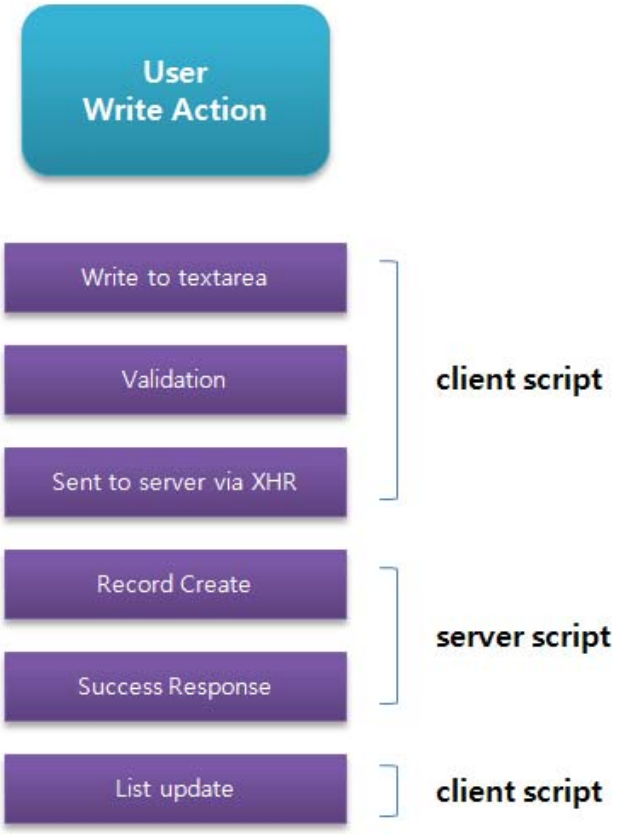


위의 그림은 간단한 게시판의 댓글을 다는 사용자 액션에 대한 흐름(서버의 응답 부분은 제외)을 그려본 것입니다. 이런 흐름은 기획자도 쉽게 알 수 있는 사용자와 시스템간의 흐름입니다.

댓글 기능에는 기본적으로 초기에 목록을 뿌리는 부분, 댓글을 쓰는 기능, 댓글을 삭제하는 기능, 댓글을 수정하는 기능, 댓글에 댓글을 다는 기능, 댓글의 댓글을 삭제하는 기능으로 나뉩니다. 즉 이런 기능들은 DBMS 의 CRUD(create, read, update, delete) 명령을 처리하는 것과도 같습니다.

위에서도 언급했듯이 이런 명령을 모두 Class 로 나누어 각각의 처리를 하게 됩니다.

간단히 Write Class 에 상세한 흐름을 살펴보면 다음과 같습니다.



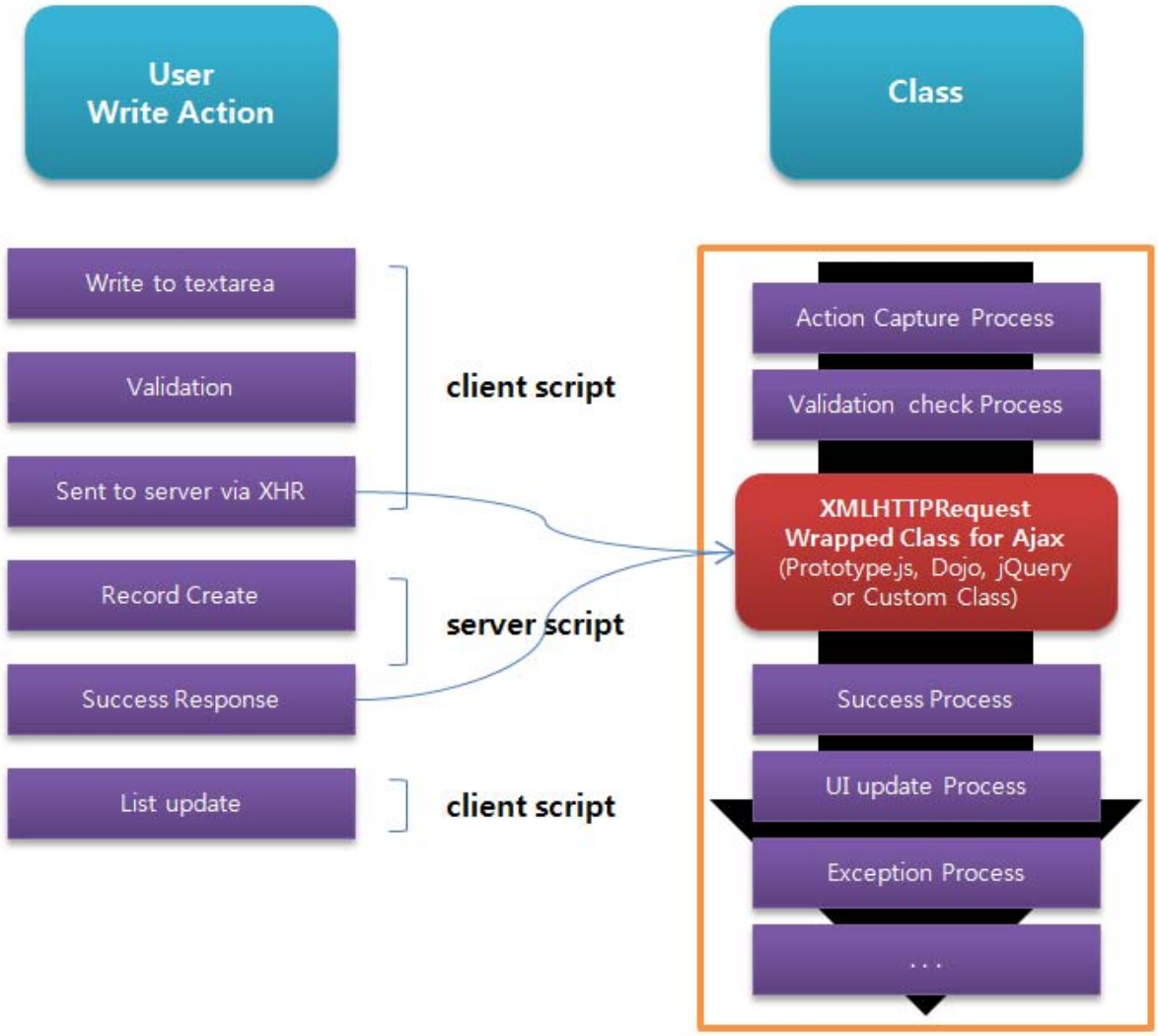
위의 그림은 사용자가 댓글을 쓰는 인터랙션에 대한 흐름을 나타냅니다. 좀더 상세한 단계가 있겠지만 현재는 기본적인 CRUD 패턴을 이해하기 위한 기본 흐름만 나타냈습니다.

위에서 분리해 놓은 client script 부분과 server script 에서 XHR 을 이용하는 부분은 client script 입니다. 즉 client script 에 해당되는 모든 액션에 대한 처리를 XHR 객체를 래핑한 클래스에서 구현되어야 할 흐름 즉 기능이라는 것입니다.

이렇게 보니 뭐 별거 없어 보입니다. 댓글과 같은 작은 시스템에서는 단순히 그림과 같은 흐름만 Ajax/Rich UI 개발자가 구현하면 되겠지만 은행권, 관공서와 같이 입력하는 양식이 많거나 복잡한 양식이 많다면 Validation 부분만 쪼개어도 매우 많은 Class 로 분리될 수 있습니다.

그리고 작성이 완료되고 처리되는 List update 단계에서도 채팅과 같이 RealTime update 가 이뤄져야 하거나 주기적인 서버와의 동기화가 필요한 경우가 발생하면 List update 부분 역시 많은 Class 로 분리되어야 할 수도 있습니다.

이처럼 복잡해질 수 있는 시스템에서 이와 같은 패턴을 적용하게 되면 경우에 따라서는 매우 많은 유지보수가 발생할 수도 있겠죠.



정리해 보면 오른쪽의 Class 와 같은 클래스가 사용자의 액션 개수만큼 생성되겠네요.

그래서 XHR Pattern 은 웹에서 간단한 인터랙션 처리(CRUD)를 위한 패턴입니다. 이와 같은 예제를 Prototype.js 를 이용하여 간단하게 구현해 보았습니다.

예제

```
/**
 * @projectDescription
 *
 * @author rhio.kim@gmail.com, tizie80@nate.com
 * @author rhio.kim blog.ecmas4.com
 * @version 0.0.02
 * @usage
 *
 * @sdoc none;
 * @namespace
 * @import
 */
var abstractComment = Class.create({
	initialize: function(options) {
		this.options		= Object.extend({
			path: '/rhioWorld/comment.php',
			defaultParam: '{ "rhio" : "is Cool" }',
			target: 'commentElementId'
		}, options || {})
	},

	/* public
	 * XHR request to server
	 */
	getServerData: function(param) {
		/* XMLHTTPRequest is wrapped in Ajax	*/
		new Ajax.Request(this.options['path'], {
			parameters: param || this.options['defaultParam'],
			onSuccess: function(XHR) {
				this._recive(XHR.responseText);
			}.bind(this),
			onFailure: function() {
				this._error();
			}.bind(this)
		});
	},

	render: function(json) {
```

```
                /* render proccess */
        },

        /* private
         * XHR responseText
         */
        _recive: function(str) {
                /* If str is string type formatted JSON, eval to JSON */
                var json = str || str.evalJSON();
                if(typeof json == 'object') this.render(json);
        },

        /* private
         * XHR error
         */
        _onerror: function() {

        }
});

/* listing Class */
var TList = Object.extend(abstractComment, {
        /* merge options for only TList Class */
        setOptions: function(options) {
                this.options    = Object.extend(this.options, options || {});
        },

        render: function(json) {
                /* listing */
        },

        evaluate: function() {
                /* make list html using template engine */
        }
});

/* write Class */
var TWrite      = Object.extend(abstractComment, {
        /* merge options for only TList Class */
```

```
        setOptions: function(options) {
                this.options        = Object.extend(this.options, options || {});
        },

        send: function() {
                /* get sever data, maybe call to getServerData */
        },

        render: function(json) {
                /* listing */
        },

        evaluate: function() {
                /* make list html using template engine */
        },

        validate: function() {
                /* user actions check validation */
        },

        update: function() {
                /* When writing process on success, List update on UI */
        }
});


var TEdit        = Object.extend(abstractComment, {});
var TDelete      = Object.extend(abstractComment, {});
var TReply       = Object.extend(abstractComment, {});
var TDeleteReply = Object.extend(abstractComment, {});
```

요약

Ajax 을 통한 개발에 있어서 사용자의 경험(user experience)은 매우 중요해졌습니다. 그만큼 사용자의 액션도 다양해졌고 매우 고급스럽게 바뀌어 가고 있습니다. 간단하게 인디케이터만 봐도 그렇습니다. 예전 Web1.0 방식에서는 웹 서버의 부하로 페이지가 열리지 않을 때 사용자는 단지 White Background 즉 아무것도 없는 하얀 백지 상태의 화면만 주시하고 있었어야 했습니다.

하지만 현재에는 인디케이터를 통해 사용자의 요청에 대한 진행 상황을 유연하게 표현함으로써 사용자의 경험을 그대로 충족 시켜줍니다.

이렇게 복잡해진 부분도 있지만 사용자 요청에 대한 대부분은 서버로부터 데이터를 요청하거나 입력, 갱신, 삭제에 대한 경우입니다. **요청에 대한 응답에 있어서 그 처리가 간단하고 명료한 시스템**이라면 CRUD 패턴을 이용해서 하나의 요청과 하나의 응답에 대한 처리를 하나의 클래스에서 처리한다면 시스템의 장애가 발생 시 어떤 부분 즉 어떤 사용자 액션에서 발생했는지 쉽게 파악하고 대응할 수 있게 됩니다.

하지만 설계 면에 있어서 향후 시스템의 확장이 있을 경우나 사용자의 액션에 대한 처리가 복잡해지게 되면 대응하더라도 설계 구조 자체를 뒤바꿔 할 여지가 다분해 보입니다.

실제로 저는 BBS 나 콘텐트의 댓글 시스템에 이전에 포스팅 한 Design by Design Pattern 과 함께 CRUD Pattern 을 이용하여 개발을 했습니다. 유지 보수가 발생하지 않아서 효율 및 실무에 적용할 만한 패턴인지 증명되지는 않았습니다.

이 내용은 Ajax/Rich UI 의 특정한 개발을 위한 방법론에 가깝다고 봐도 무방합니다. 또한 개인적인 경험에 의한 정리이며 일반적인 시스템 설계에서 말하는 패턴의 정의와는 상이하거나 다른 의미를 갖을 수 있습니다.