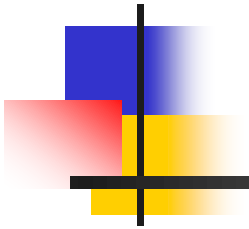
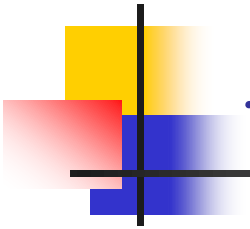


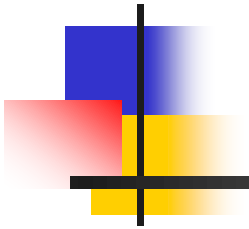
JQuery 3강- ajax

양 명 숙
[now4ever7@gmail.com]

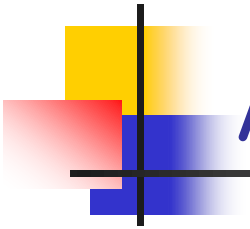


목차

- Ajax 개요
- Ajax 관련 메서드

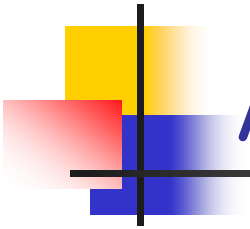


Ajax 개요



Ajax 개요

- Ajax(Asynchronous JavaScript and XML)
  - 비동기 자바스크립트와 XML의 약자로, 자바스크립트를 이용해 서버와 비동기 방식의 통신을 해서 웹 페이지를 갱신하지 않은 채로 여러 가지 작업을 수행하는 프로그래밍 모델, 구현 방식
  - 기존의 웹 페이지는 새로운 데이터를 사용자에게 보여주려면 항상 페이지를 전환해야 했다
  - Ajax를 이용하면 페이지를 전환하지 않고, 서버에서 데이터를 받아와 사용자에게 보여줄 수 있다

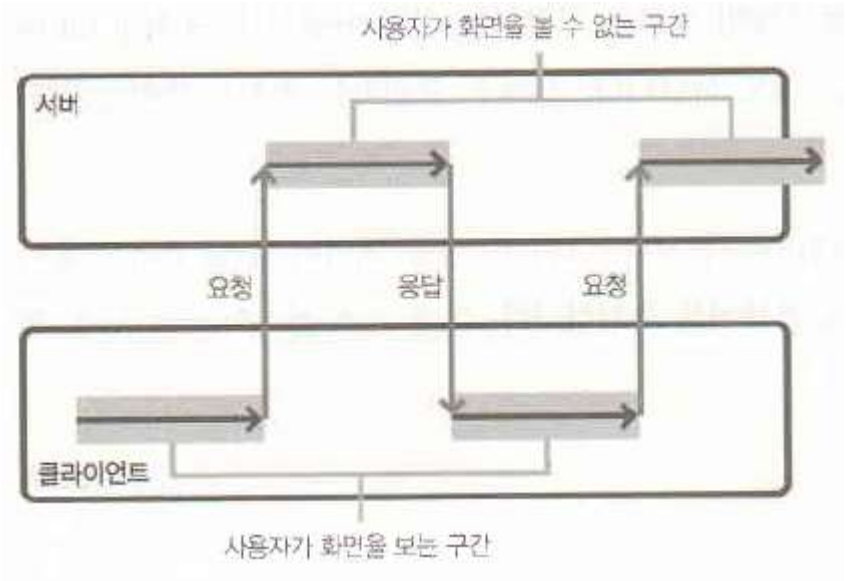


AJAX 개요

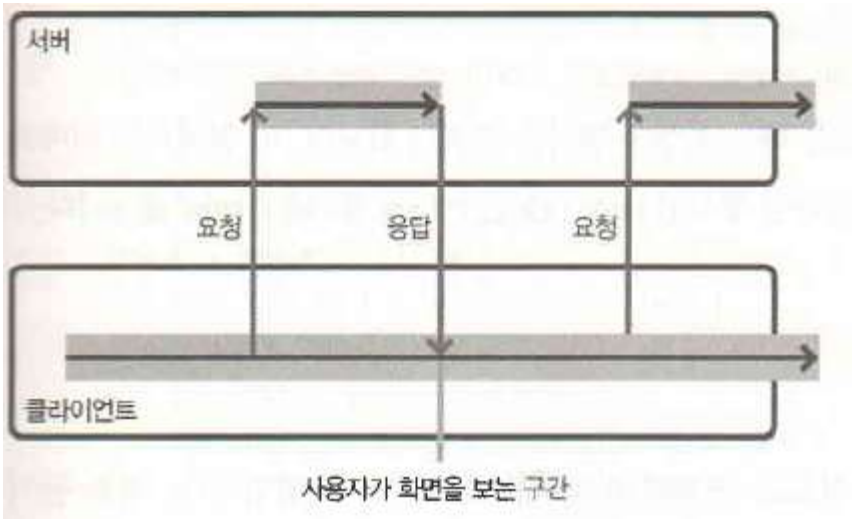
- AJAX(Asynchronous JavaScript and XML)
  - 사용자 경험을 개선하기 위한 웹 기반의 비 동기적 통신 기술로, 웹 상에서 화면의 새로 고침(refresh)없이 빠르게 화면 전환됨
  - 웹 상에서 새로운 데이터나 프로그램의 새로운 부분에 접근할 때마다 웹 페이지 전체를 새로 읽지 않고, 소량의 데이터만 전송함으로서 웹 페이지의 일부를 동적으로 구현함이 가능하게 됨
  - 웹 서버와 데이터를 주고받아 사용자가 페이지 이동 없이 즉각적으로 원하는 기능을 수행할 수 있도록 하는 것

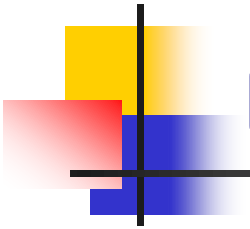
Ajax 개요

[1] 기존 방식



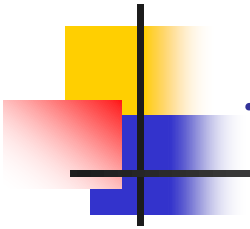
[2] Ajax 방식





대표적인 적용 예

- 검색엔진이나 쇼핑몰의 자동완성 기능
  - http://www.naver.com/
- 구글맵
  - http://maps.google.com/
  - XMLHTTP와 자바스크립트를 이용하여 순수 HTML만으로 모든 브라우저에서 정확하게 동작하는 지도 서비스를 만듬
    - 특정 브라우저에 얽매이지 않고 새로고침 없이 액티브 X를 설치하지 않고 지도서비스 사용
    - 액티브 X – 인터넷 익스플로러에서만 동작하는 비표준 기술



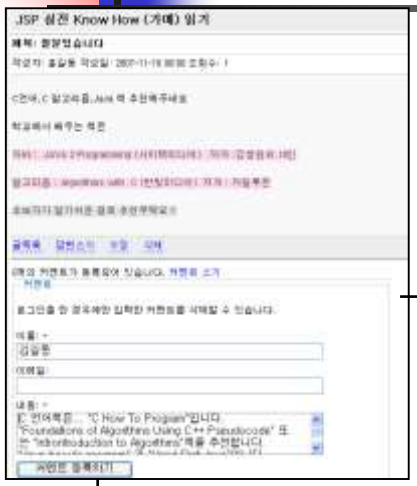
동기적 통신/비 동기적 통신

- 동기적 통신
  - 사용자가 어떤 행동을 취했을 경우(예:버튼 클릭) 그 요청이 서버로 전달되어 처리되고 결과가 반환되어 오기까지 다음 행동을 취하지 못하고 대기해야만 하는 것을 말함
  - 동기적 – 하나의 작업이 완료된 후에만 다음 작업을 진행할 수 있는 구조
  - 대부분의 웹/윈도우 응용 프로그램이 동기적으로 구현됨

동기적 통신/비 동기적 통신

- 비 동기적 통신
  - 사용자가 버튼을 클릭한 뒤, 그 행동에 의한 결과를 기다림 없이 다른 버튼을 클릭한다거나, 다른 입력 작업을 수행할 수 있다는 것
  - 사용자는 요청에 대한 응답이 돌아오지 않은 상태이더라도, 기다림 없이 다른 작업을 진행할 수가 있음
  - 비 동기적 – 하나의 작업이 완료되지 않은 상태에서도 다른 작업을 얼마든지 시작할 수 있는 구조
  - Ajax 웹 응용 프로그램 모델

Ajax 방식의 사이트 특징



자바 스크립트

DOM

서버와 통신 담당

웹 서버

데이터 전달

요청받음

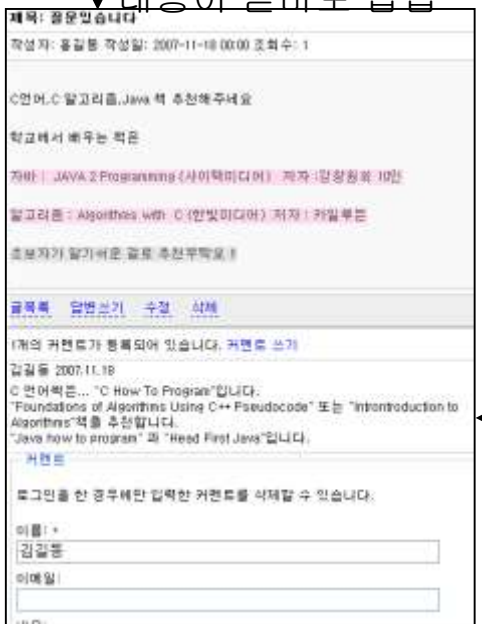
요청처리

DB

XML 생성 (또는 텍스트, JSON)

XMLHTTPRequest

페이지 이동없이 입력한 내용이 곧바로 삽입



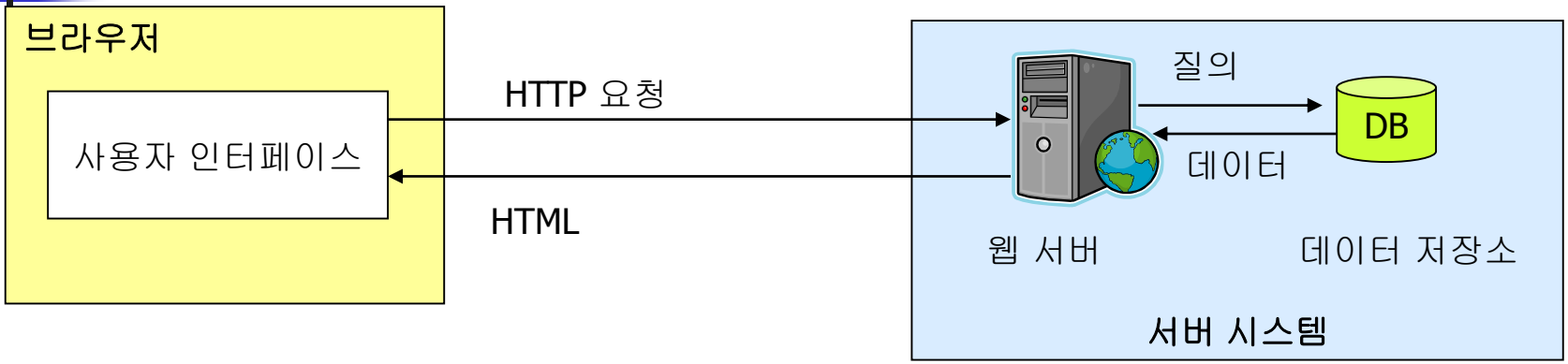
자바 스크립트

DOM

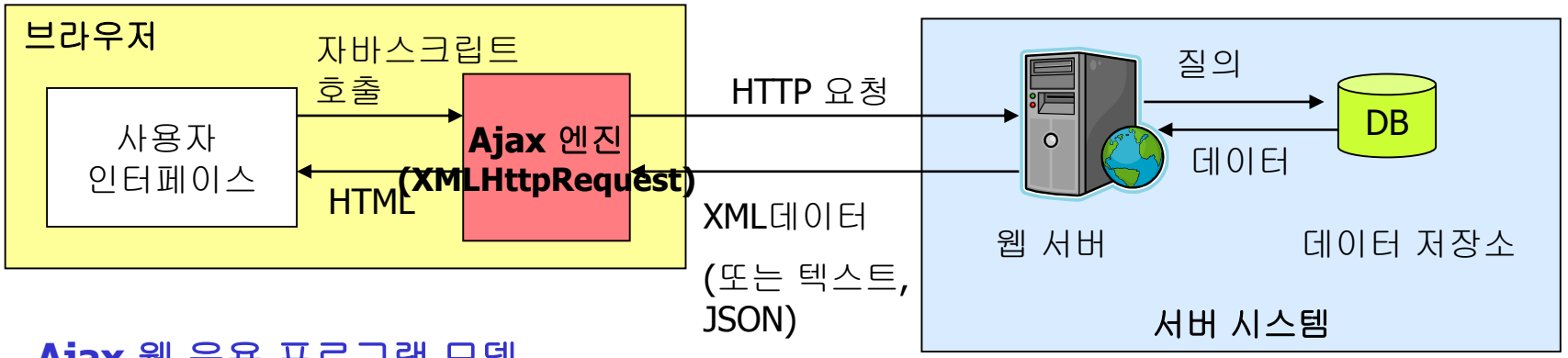
Ajax 방식의 사이트 특징

- 사용자가 이벤트(버튼 클릭 등)를 발생시키면 자바스크립트는 DOM을 사용해서 필요한 정보를 구한 뒤 XMLHttp 객체를 통해서 웹 서버에 요청을 전달하게 됨
- 웹 서버는 XMLHttp로부터의 요청을 알맞게 처리한 뒤, 그 결과를 XML이나 단순 텍스트, JSON으로 생성해서 XMLHttp에 전송함
- 서버로부터 응답이 도착하면 XMLHttp는 자바 스크립트에 도착 사실을 알리게 되고, 자바 스크립트는 응답 데이터와 DOM을 조작해서 사용자 화면에 반영함
- Ajax와 기존 방식과 차이점
  - 웹 브라우저가 아닌 **XMLHttp객체가 웹 서버와 통신**함
  - 웹 서버의 **응답 결과가** HTML이 아닌 **XML** 또는 단순 텍스트, JSON임
  - 페이지 이동 없이 결과가 화면에 반영됨

2가지 형태의 웹 응용 프로그램 모델



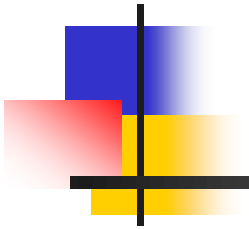
전형적인 웹 응용 프로그램 모델



Ajax 웹 응용 프로그램 모델

Ajax의 구동방식

- 서버와 클라이언트 입장
  - **전통적**인 경우 – 모든 웹 페이지가 **하나의 완전한 HTML**을 반환하는 형태로 제작됨
  - **Ajax** – 서버에서는 전체 HTML을 반환해주는 페이지가 아닌 **특정 기능을 수행한 결과만**을 텍스트나 JSON, XML로 반환하는 페이지가 필요함
    - 이 페이지는 Ajax엔진에 의해서 요청되며, 그 결과 데이터 또한 Ajax엔진이 비 동기적으로 받아오게 됨
    - Ajax엔진에 의해 가져와진 데이터는 HTML과 자바스크립트를 통해서 화면에 동적으로 반영됨
- Ajax의 동작 방식은 **딱 필요한 데이터만을 가져와** 화면에 반영하므로, 네트워크의 트래픽을 감소



데이터 전송 형식

데이터 전송 형식

- 서버와 클라이언트가 데이터를 주고 받을 때는 특정한 형식을 맞춰야 함
- 서버와 클라이언트가 통신할 때 자주 사용하는 파일 형식
  - CSV, XML, JSON 형식
- [1] CSV(Comma Separated Values) 형식
  - 각 항목을 쉼표로 구분해 데이터를 표현하는 방법
  - CSV 형식은 가독성이 떨어짐

```
홍길동, hong, 20
김길동, kim, 25
이길동, lee, 29
```

데이터 전송 형식

- [2] XML 형식
  - CSV 형식은 각각의 데이터가 무엇을 나타내는지 알기 힘들다
  - XML은 HTML 처럼 태그로 데이터를 표현함
  - 각각의 데이터가 어떠한 것을 의미하는 지 알 수 있다

```
<?xml version="1.0" encoding="UTF-8"?>
<members>
        <member>
                <name>홍길동</name>
                <userid>hong</userid>
                <age>20</age>
        </member>
        <member>
                <name>김길동</name>
                <userid>kim</userid>
                <age>25</age>
        </member>
        <member>
                <name>이길동</name>
                <userid>lee</userid>
                <age>29</age>
        </member>
</members>
```

데이터 전송 형식

- [3] JSON (Javascript Object Notation) 형식
  - CSV 형식과 XML 형식의 단점을 극복한 형식
  - 자바스크립트에서 사용하는 객체 형태로 데이터를 표현하는 방법
  - Ajax를 사용할 때 거의 표준으로 사용되는 데이터 표현 방식
  - CSV 형식과 달리 데이터의 가독성이 좋고, XML 형식보다 적은 용량으로 데이터를 전달할 수 있다
  - 단점 – 데이터의 양이 아주 많아지면 데이터 추출 속도가 현저하게 떨어짐

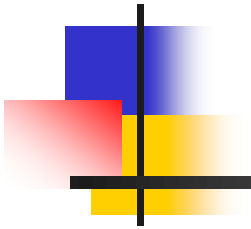
```
[{
        "name" : "홍길동",
        "userid": "hong",
        "age" : "20"
},{
        "name" : "김길동",
        "userid": "kim",
        "age" : "25"
},{
        "name" : "이길동",
        "userid": "lee",
        "age" : "29"
}]
```

자바스크립트에서 사용할 수 있는 객체이나 제약사항이 있다
- JSON 에는 객체, 배열, 문자열, 숫자, boolean, null 만 들어갈 수 있다
- 문자열은 무조건 큰 따옴표를 사용해야 함

JSON(Javascript Object Notation)

- key와 value로 구성
- [ ]를 사용하여 배열형 value 사용 가능

```
[
   { "key1" : "value",   "key":["item1", "item2"]},
   { "key1" : "value",   "key":["item1", "item2"]}
]
```



JQuery Ajax

JQuery Ajax

- jQuery는 Ajax와 관련된 다양한 종류의 메서드를 제공
- 가장 기본적인 jQuery의 Ajax 관련 메서드
  - $.ajax() 메서드 – 가장 많이 사용함

```
$.ajax(options) ;

$.ajax(url, options) ;
```

- options : 옵션 객체

```
$.ajax('/search.do', {
                success: function (res) { }
});
```

- success 속성 : 이벤트
- $.ajax() 메서드는 Ajax가 성공했을 때 자동으로 success 이벤트를 실행함
- success 이벤트 리스너의 첫 번째 매개변수는 Ajax가 성공했을 때 받은 데이터
- $.ajax() 메서드는 내부에서 XMLHttpRequest 객체를 만들어 Ajax를 수행함

JQuery Ajax

```
$.ajax({
            url: '/search.do',
            success: function (res) {
                $('#div1').append(res);
            }
});
```

- 매개변수 url은 옵션 속성으로 지정할 수도 있다

$.ajax() 메서드의 옵션

옵션 속성 이름	설명	자료형
async	동기, 비동기를 지정합니다.	Boolean
complete(xhr, status)	Ajax 완료 이벤트 리스너를 지정합니다.	Function
data	요청 매개변수를 지정합니다.	Object, String
error(xhr, status, error)	Ajax 실패 이벤트 리스너를 지정합니다.	Function
jsonp	JSONP 매개변수 이름을 지정합니다.	String
jsonpCallback	JSONP 콜백 함수 이름을 지정합니다.	String, Function
success(data, status, xhr)	Ajax 성공 이벤트 리스너를 지정합니다.	Function, Array
timeout	만료 시간을 지정합니다.	Number
type	'GET' 또는 'POST' 등을 지정합니다.	String
url	대상 URL을 지정합니다.	String

$.ajax() 메서드의 옵션

url	요청을 보낼 서버 URL(JSP, ASP, PHP 등)
type	HTTP 메서드, 기본값은 GET, POST와 GET 중 선택
data	서버로 전송되는 데이터
dataType	응답 결과의 데이터 형식 가용한 형식 : xml, html, json, jsonp, script, text
timeout	요청에 대한 응답 제한 시간 (millisecond)
contentType	서버 전송시의 content-type, application/x-wwwform-urlencoded 기본
success	응답이 성공하는 경우 호출되는 콜백함수
error	에러 발생시 실행되는 콜백함수
complete	요청이 완료되었을 때 호출되는 콜백함수
asyn	false로 지정하면 비동기 호출 전송. true 설정시 동기 호출

JQuery Ajax

```
$.ajax({
        url: '/search.do',
        type: 'GET',
        data: {
            keyword: '김',
            id: 'abc'
        },
        success: function (res) {
            $('#div1').append(res);
        }
});
```

- 요청 매개변수와 함께 Ajax 요청 수행

예제

sk,sbs,sm

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<script src="<c:url value='/jquery/jquery-1.11.1.js' />" type="text/javascript"></script>
<script>
    $(document).ready(function() {
        $.ajax({
            url :"<c:url value='/search.do' />",
            success : function(res) {
                $('#div1').append(res);
            },
            error: function(xhr, status, error){
                alert(error);
            }
        });
    });
</script>

<div id="div1"></div>
```

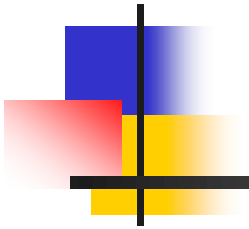
AjaxTestController

```
@Controller
public class AjaxTestController {
	private static final Logger logger = LoggerFactory.getLogger(AjaxTestController.class);

	@RequestMapping("/jqTest/ajaxTest1.do")
	public void ajaxTest1() {
	}

	//ajax 이용
	@ResponseBody
	@RequestMapping("/search.do")
	public String search(@RequestParam(required=false) String keyword,
				@RequestParam(required=false) String id){
		logger.debug("ajax 이용1, keyword={},id={}", keyword, id);

		String result=keyword +","+id+","+"sk,sbs,sm";

		return result;
	}
```

s,abc,sk,sbs,sm

예제2

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<script src="<c:url value='/jquery/jquery-1.11.1.js' />" type="text/javascript"></script>
<script>
	$(document).ready(function() {
		$.ajax({
			url : "<c:url value='/search.do' />",
			type : 'GET',
			data : {
				keyword : 's',
				id : 'abc'
			},
			success : function(res) {
				$('#div1').append(res);
			},
			error : function(xhr, status, error) {
				alert(error);
			}
		});
	});
</script>
<div id="div1"></div>
```



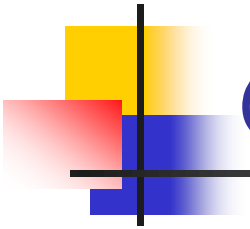
Jquery Ajax – 서버 페이지

@ResponseBody

- @ResponseBody 가 메서드 레벨에 부여되면 메소드가 리턴하는 오브젝트는 뷰를 통해 결과를 만들어내는 모델로 사용되는 대신, 메시지 컨버터를 통해 바로 HTTP 응답의 메시지 본문으로 전환됨

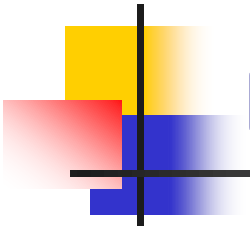
```
@RequestMapping("/test.do")
@ResponseBody
public String test(){
        String result="<html><body>Hello spring</body></html>";

        return result;
}
```

- @ResponseBody 가 없다면, 스트링 타입의 리턴값은 뷰 이름으로 인식될 것임
-@ResponseBody가 붙었으므로 스트링 타입을 지원하는 메시지 컨버터가 이를 변환해서 **HttpServletResponse 의 출력 스트림에 넣어** 버린다
- @RequestBody, @ResponseBody는 XML 이나 JSON 과 같은 메시지 기반의 커뮤니케이션을 위해 사용됨



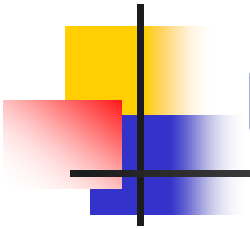
@RequestBody

- @RequestBody 가 붙은 파라미터에는 HTTP 요청의 본문(Body) 부분이 그대로 전달된다
- 일반적인 GET/POST 의 요청 파라미터라면 @RequestBody를 사용할 일이 없으나, XML이나 JSON 기반의 메시지를 사용하는 요청의 경우에는 이 방법이 매우 유용함
- @RequestBody가 붙은 파라미터가 있으면 HTTP 요청의 본문 부분을 통째로 변환해서 지정된 메서드 파라미터로 전달해 줌
- XML 본문을 가지고 들어오는 요청은 MarshallingHttpMessageConverter 등을 이용해서 XML이 변환된 오브젝트로 전달받을 수 있다
- JSON 타입의 메시지라면 MappingJacksonHttpMessageConverter를 사용할 수 있다



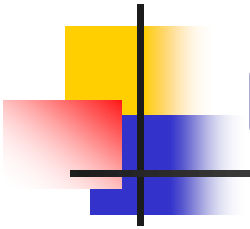
메시지 컨버터와 AJAX

- 메시지 컨버터는 XML이나 JSON을 이용한 AJAX 기능이나 웹 서비스를 개발할 때 사용할 수 있다
- HTTP 요청 프로퍼티를 모델 오브젝트의 프로퍼티에 개별적으로 바인딩하고 모델 오브젝트를 다시 뷰를 이용해 클라이언트로 보낼 콘텐트로 만드는 대신, HTTP 요청 메시지 본문과 HTTP 응답 메시지 본문을 통째로 메시지로 다루는 방식이다
- 메시지 컨버터는 파라미터의 @RequestBody와 메서드에 부여한 @ResponseBody를 이용해서 쓸 수 있다



메시지 컨버터와 AJAX

- MappingJacksonHttpMessageConverter
  - Jackon ObjectMapper를 이용해서 **자바 오브젝트와 JSON 문서를 자동변환**해주는 메시지 컨버터
- JSON을 이용한 AJAX 컨트롤러
  - AJAX – 자바스크립트를 이용해 서버와 비동기 방식의 통신을 해서 웹 페이지를 갱신하지 않은 채로 여러 가지 작업을 수행하는 프로그래밍 모델, 구현방식
  - AJAX에서 서버와 주고받는 메시지는 XML로 제한된 것이 아님, JSON이 AJAX의 인기 메시지 포맷으로 자리잡았다
  - JSON 기반의 AJAX를 **지원하려면 컨트롤러는 결과를 JSON 포맷으로 만들어서 돌려줘야 함**



메시지 컨버터와 AJAX

- 스프링 MVC에서 결과를 JSON 포맷으로 만드는 두 가지 방법
  - [1] JSON 지원 뷰를 사용하는 방법
    - 모델에 JSON으로 변환할 오브젝트를 넣고, MappingJacksonJsonView 뷰를 선택하면 됨
  - [2] @ResponseBody 를 이용하는 것
    - 콘텐트교섭을 통해 JSON 뷰를 결정할 것이 아니고, 항상 JSON으로 고정됐다면 @ResponseBody 를 이용하는 편이 훨씬 간결함
- AJAX 요청을 보내는 방법
  - [1] 단순 GET방식
  - [2] POST방식
    - 1) 일반 폼을 보내는 것
    - 2) JSON 메시지를 보내는 것

JSON을 이용한 AJAX 컨트롤러:GET+JSON

- 1. GET 방식의 단순 요청을 받아서 JSON으로 결과 보내기
  - 결과 오브젝트에 결과를 담은 뒤 이를 다시 JSON으로 변환해서 클라이언트로 보내야 함 – 메시지 컨버터를 이용
  - [1] JSON 변환을 지원하는 MappingJacksonHttpMessageConverter를 AnnotationMethodHandlerAdapter 빈의 messageConverters 프로퍼티에 추가해준다

```
<beans:bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
        <beans:property name="messageConverters">
                <beans:list>
                        <beans:bean
        class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"/>
                </beans:list>
        </beans:property>
</beans:bean>
```

<mvc:annotation-driven>

- 애노테이션 방식의 컨트롤러를 사용할 때 필요한 DispatcherServlet 전략빈을 자동으로 등록해줌
- 또, 최신 @MVC 지원 기능을 제공하는 빈도 함께 등록하고 전략 빈의 프로퍼티로 설정해줌
- 라이브러리의 존재 여부를 파악해서 자동으로 관련 빈을 추가해주는 기능도 제공됨
- <mvc:annotation-driven>에 의해 자동으로 등록되는 빈 정보
- (1) DefaultAnnotationHandlerMapping
- (2) AnnotationMethodHandlerAdapter
- (3) ConfigurableWebBindingInitializer
- (4) 메시지 컨버터
  - AnnotationMethodHandlerAdapter 의 messageConverters 프로퍼티로 메시지 컨버터들이 등록됨
  - 4 개의 디폴트 메시지 컨버터와 함께 Jaxb2RootElementHttpMessageConverter와 MappingJacksonHttpMessageConverter가 추가로 등록됨
    - 단, 각각 JAXB2와 Jackson 라이브러리가 클래스패스에 존재하는 경우에만 등록됨

JSON을 이용한 AJAX 컨트롤러:GET+JSON

- [2] 컨트롤러 메서드에 @ResponseBody 애노테이션을 붙여준다

```
@RequestMapping("/memo/ajaxDetail.do")
@ResponseBody
public MemoVO memoDetail(@RequestParam int no) {
        logger.debug("ajax 상세 조회, no={}", no);

        //db에서 no에 해당하는 정보를 조회한 결과를 리턴
        MemoVO memo=new MemoVO(no, "홍길동","안녕");

        return memo;
}
```

⇒ 이 컨트롤러가 리턴하는 MemoVO 오브젝트는 JSON 포맷의 메시지로 변환됨
⇒ MemoVO 오브젝트는 @ResponseBody 설정에 따라 MappingJacksonHttpMessageConverter에 넘겨지고, JSON 메시지로 만들어져 HTTP 응답 메시지 본문으로 설정되어 클라이언트로 보내짐

JSON을 이용한 AJAX 컨트롤러:GET+JSON

- [3] 클라이언트에서 자바스크립트를 이용해 해당 URL(/memo/ajaxDetail.do)을 호출해주고 그 결과를 받아서 사용자에게 알려주는 코드를 작성

```
var data='no='+$('#no').val();
$.ajax({
    url: '<c:url value="/memo/ajaxDetail.do"/>',
    type: 'get',
    dataType: 'json',
    data:data ,
    success: function (res) {
            var output="번호 : "+ res.no+"<br>"
                    +"이름 : "+res.name +"<br>"
                    +"메모 : " + res.content;
        $('#result').html(output);
    }});
```

- JSON의 장점 : 자바의 MemoVO 오브젝트의 프로퍼티에 접근하는 것과 비슷한 표기 방법을 사용할 수 있다는 점
- MemoVO 타입을 JSON으로 변환해서 받은 결과는 res.name, res.content 와 같은 식으로 사용할 수 있다

JSON을 이용한 AJAX 컨트롤러:POST+JSON

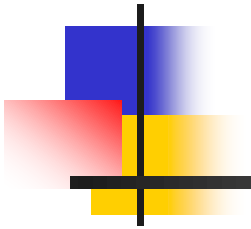
- 2. POST 메소드를 사용하고 본문에 JSON으로 된 정보를 보내는 방법
  - JSON 으로 전달되는 요청은 MappingJacksonHttpMessageConverter를 통해 @RequestBody가 붙은 파라미터로 변환되어 받을 수 있다
  - [1] 사용자 정보를 입력 받는 HTML 폼 작성
    - 등록 버튼을 누르면 폼의 모든 필드 정보를 JSON 메시지로 만들어서 POST 로 전송하도록 만든다
    - AJAX 폼 전송용 자바스크립트 코드 작성
    - 이 자바스크립트 코드에 의해 서버로 전송되는 HTTP 요청의 메시지 본문은 JSON 포맷으로 만들어짐

```
$.ajax({
        url: '<c:url value="/memo/ajaxWrite.do" />',
        type: 'post',
        dataType: 'json',
        data:$(this).serializeArray(),
        success: function (res) {
                var output=res.message+"<br>"
                +"번호 : "+ res.data.no+"<br>"
                +"이름 : "+res.data.name +"<br>"
                +"메모 : " + res.data.content;
                $('#result').html(output);
        }});
```

JSON을 이용한 AJAX 컨트롤러:POST+JSON

- [2] 서버 코드
- POST 메서드를 처리하는 컨트롤러 메소드 추가

```
@RequestMapping("/memo/ajaxWrite.do")
@ResponseBody
public ResultVO memoWrite(@ModelAttribute MemoVO memo) {
        logger.debug("ajax 등록, memo={} db에 insert함", memo);

        //db에 등록 후, 등록한 정보를 다시 리턴
        ....
        return result;
}
```



JQuery Ajax

추가적인 jQuery Ajax 메서드

- 추가적인 jQuery Ajax 메서드를 사용하면 $.ajax() 메서드를 더 간단하게 사용할 수 있다

메서드 이름	설명
$.get(url, data, callback, type)	get 방식으로 Ajax를 수행함
$.post(url, data, callback, type)	post 방식으로 Ajax를 수행함
$.getJSON(url, data, callback)	get 방식으로 Ajax를 수행해 JSON 데이터를 가져옴
$.getScript(url, callback)	get 방식으로 Ajax를 수행해 Script 데이터를 가져옴
$(selector).load(url, data, callback)	Ajax를 수행하고 선택자로 선택한 문서 객체 안에 집어 넣는다

- 이 메서드는 $.ajax() 메서드의 옵션 속성을 미리 설정함
- $.get() , $.post() 메서드
  - 사용방법이 같지만, $.ajax() 메서드에서 type 옵션 속성이 미리 지정돼 있음
  - $.post() 메서드는 type 속성이 "POST"

추가적인 jQuery Ajax 메서드

```
$.get(url, function(data, textStatus, jqXHR){}) ;
$.post(url, function(data, textStatus, jqXHR){}) ;

$.get(url, data, function(textStatus, jqXHR){}) ;
$.post(url, data, function(textStatus, jqXHR){}) ;
```

```
 $(document).ready(function () {
        $.get('/search.do', function (data) {
            $('#div').html(data);
        });
});
```

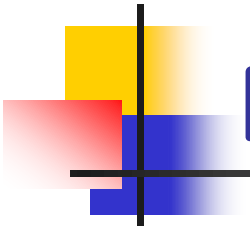
```
$(document).ready(function () {
        $('#div').load('/search.do');
});
```

결과는 동일

추가적인 jQuery Ajax 메서드

```
$(document).ready(function () {
        $.getJSON('/cg.do', function (data) {
            $.each(data, function (idx, item) {
                $('div').append('<p>' + item.cgName + ' - ' + item.cgNo +
                    '</p>');
            });
        });
});
```

/cg.do 에서 JSON을 가져와 화면에 출력함



MemoVO

```
public class MemoVO {
    private int no;
    private String name;
    private String content;
.....
}
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<script src="<c:url value='/jquery/jquery-1.11.3.js' />" type="text/javascript"></script>
<script>
    $(document).ready(function() {
        $.ajax({
            type:"GET",
            url :"<c:url value='/memo/ajaxList.do' />",
            dataType:'json',
            success: function(res){
                if (res.memoList.length > 0) {
                    $('#div1').empty();
                    $.each(res.memoList, function() {
                        var info = "<p>"+ this['no']+"-"+this['name']+"-"+
                          this['content']+ "-"+ "</p>";
                          $('#div1').append( info );
                    });
                }
            },
            error: function(xhr, status, error){
                alert(error);
            }
        });
    });
</script>  <div id="div1"></div>
```

```
$.each(res.memoList, function(idx, item) {
        var info = "<p>"+ item.no +"-" +
item.name+"-"+  item.content+ "</p>";

        $('#div1').append( info );
});
```


```
{"memoList":[{"no":1,"name":"홍길동","content":"안녕"},{"no":2,"name":"김길동","content":"hi"},{"no":3,"name":"이길동","content":"hello"}]}
```

[1] MappingJacksonJsonView 이용

```
@RequestMapping("/memo/ajaxList.do")
public ModelAndView memoList() {
        logger.debug("ajax 목록 조회");

        List<MemoVO> memoList = new ArrayList<MemoVO>();
        MemoVO meno1=new MemoVO(1, "홍길동","안녕");
        memoList.add(meno1);
        MemoVO meno2=new MemoVO(2, "김길동","hi");
        memoList.add(meno2);
        MemoVO meno3=new MemoVO(3, "이길동","hello");
        memoList.add(meno3);

        return new ModelAndView("jsonView", "memoList", memoList);
}
```

servlet-context.xml

```
<beans:bean id="jsonView"
      class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />

<beans:bean class="org.springframework.web.servlet.view.BeanNameViewResolver">
          <beans:property name="order" value="0" />
</beans:bean>
```

[2] @ResponseBody 이용

```
@RequestMapping("/memo/ajaxList.do")
@ResponseBody
public List<MemoVO> memoList() {
        logger.debug("ajax 목록 조회");

        List<MemoVO> memoList = new ArrayList<MemoVO>();
        MemoVO meno1=new MemoVO(1, "홍길동","안녕");
        memoList.add(meno1);
        MemoVO meno2=new MemoVO(2, "김길동","hi");
        memoList.add(meno2);
        MemoVO meno3=new MemoVO(3, "이길동","hello");
        memoList.add(meno3);

        return memoList;
}
```

[{"no":1,"name":"홍길동","content":"안녕"},{"no":2,"name":"김길동","content":"hi"},{"no":3,"name":"이길동","content":"hello"}]

ajaxTest3.jsp

```
$.ajax({
        type:"GET",
        url :"<c:url value='/memo/ajaxList.do' />",
        dataType:'json',
        success: function(res){
                if (res.length > 0) {
                        $('#div1').empty();

                        $.each(res, function(idx, item) {
                        var info = "<p>"+ item.no +"-" + item.name+"-"+
                                item.content+ "</p>";

                                $('#div1').append( info );
                        });
                }
        },
        error: function(xhr, status, error){
                alert(error);
        }
});
```

memoinfo.json

```
[
{"no":"1", "name" : "홍길동", "content" : "hong입니다"},
{"no":"2", "name" : "김길동", "content" : "kim입니다"},
{"no":"3", "name" : "이길동", "content" : "lee입니다"}
]
```

```
$.ajax({
        type:"GET",
        url :"<c:url value='/inc/memoinfo.json' />",
        dataType:'json',
....
```

번호 : 3 조회

메모를 남기세요

이름 :

메모 :

입력

결과

번호 : 3
이름 : 홍길동
메모 : 안녕

번호 : 조회

메모를 남기세요

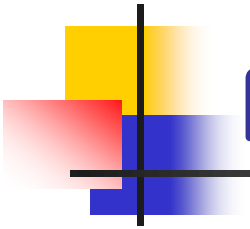
이름 : 강길동

메모 : 환영합니다

입력

결과

데이터 등록 성공
번호 : 4
이름 : 강길동
메모 : 환영합니다



ResultVO

```
public class ResultVO {
    private String message;
    private MemoVO data;
...
}
```

AjaxTestController

```
@RequestMapping("/memo/ajaxDetail.do")
@ResponseBody
public MemoVO memoDetail(@RequestParam int no) {
        logger.debug("ajax 상세 조회, no={}", no);

        //db에서 no에 해당하는 정보를 조회한 결과를 리턴
        MemoVO memo=new MemoVO(no, "홍길동","안녕");

        return memo;
}
@RequestMapping("/memo/ajaxWrite.do")
@ResponseBody
public ResultVO memoWrite(@ModelAttribute MemoVO memo) {
        logger.debug("ajax 등록, memo={} db에 insert함", memo);

        //db에 등록 후, 등록한 정보를 다시 리턴
        memo.setNo(4);

        ResultVO result = new ResultVO();
        result.setMessage("데이터 등록 성공");
        result.setData(memo);

        return result;
}
```

데이터 요청 방식-GET 요청, POST요청

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<script src="<c:url value='/jquery/jquery-1.11.1.js' />" type="text/javascript"></script>
    <script>
        $(document).ready(function () {
            $('#query').click(function () {
              var data='no='+$('#no').val();
               $.ajax({
                    url: '<c:url value="/memo/ajaxDetail.do"/>',
                    type: 'get',
                    dataType: 'json',
                    data:data ,
                    success: function (res) {
                            var output="번호 : "+ res.no+"<br>"
                                    +"이름 : "+res.name +"<br>"
                                    +"메모 : " + res.content;
                            $('#result').html(output);
                    },
                     error: function(xhr, status, error){
                            alert(error);
                     }
                });
            });
```

```
{"no":1,"name":"홍길동","content":"안녕"}
```

```
$('#frm1').submit(function () {
    $.ajax({
        url: '<c:url value="/memo/ajaxWrite.do" />',
        type: 'post',
        dataType: 'json',
        /* data: {
            name: $('#name').val(),
            content: $('#content').val()
        }, */
        data:$(this).serializeArray(),
        success: function (res) {
                var output=res.message+"<br>"
                        +"번호 : "+ res.data.no+"<br>"
                        +"이름 : "+res.data.name +"<br>"
                        +"메모 : " + res.data.content;
                $('#result').html(output);
        },
        error: function(xhr, status, error){
                alert(error);
        }
    }); //ajax

    // 기본 이벤트 제거
    event.preventDefault();
});

});//ready
</script>
```

```
{"message":"데이터 등록 성공","data":{"no":4,"name":"강길동","content":"환영합니다"}}
```

```
<body>
<form id="frm1">
	번호 : <input type="text" id="no" size="7"/>
	<input type="button" id="query" value="조회"><br>

	<h2>메모를 남기세요</h2>
	이름 : <input type="text" id="name" name="name"/><br>
	메모 : <input type="text" id="content" name="content" size="50" /><br>
	<input type="submit" value="입력">

	<h2>결과</h2>
	<div id="result" style="background:#eeeeee;width:500px"></div>
</form>
</body>
```

보조 메서드

메서드 이름	설명
serialize()	입력 양식의 내용을 요청 매개변수 문자열(**쿼리문자열**)**로** 만든다
serializeArray()	입력 양식의 내용을 **객체로** 만든다
$.param()	객체의 내용을 요청 매개변수 문자열로 만든다 객체를 쿼리 문자열로 바꿈

serialize(), serializeArray() 메서드 - 입력 양식과 함께 사용하는 메서드

$.param()

name=hongkildon&id=abc123

```
<head>
    <script src="../../jquery/jquery-1.11.1.js" type="text/javascript"></script>
     <script>
        $(document).ready(function () {
             var data = {
                name: 'hongkildon',
                id: 'abc123',
             };

            $('<p></p>').text($.param(data)).appendTo('div');
        });
    </script>
</head>
<body>
     <div></div>
</body>
```

쿼리 문자열이 생성돼 출력됨

[1] 쿼리 문자열로 전송 : serialize(), $.param()
[2] 객체로 전송: serializeArray()

jQuery Ajax 로 입력 양식 전송하는 방법

- [1] 각각의 입력양식에서 value 속성을 직접 가져온 뒤 url 생성
- [2] 각각의 입력양식에서 value 속성을 직접 가져온 뒤 value 속성을 이용해 객체를 만들고, $.param() 메서드를 사용해 쿼리문자열로 만든 후에 ajax 와 관련된 메서드의 data 속성에 넣는 방법
- [3] 각각의 입력양식에서 value 속성을 가져온 뒤 value 속성을 이용해 객체를 만들고 곧바로 ajax 관련 메서드의 data 속성에 넣는 방법
  - 많이 사용됨, 하지만 submit 이벤트와 연결하는 경우는 많지 않다
- [4] serialize() 메서드를 사용하면 입력 양식에 적힌 값을 쿼리 문자열로 바꿈
- [5] serializeArray() 메서드를 사용하면 입력 양식에 적힌 값을 객체로 만든다, 이어서 ajax 관련 메서드의 data 속성에 넣는다

```
    <script>
      $(document).ready(function () {
          $('#frm1').submit(function (event) {
              var name = $('#name').val();
              var content = $('#content').val();

              // Ajax 요청 수행
              //var url = '/memo/ajaxWrite.do?name=' + name + '&content=' + content;  [1]
              //$('#result').load(url);

              var url = '/memo/ajaxWrite.do';
              var data = { name: name, content: content };  //[2]
              var params = $.param(data);
              $('#result').load(url, params);

              // 기본 이벤트 제거
              event.preventDefault();
          });
      });
   </script>
<body>    <form id="frm1">
      <input type="text"  id="name" name="name" />
      <input type="text"  id="content" name="content"  />
      <input type="submit" value="전송" />
   </form>
   <div id="result"></div> </body>
```

var data=`name='+ name + '&content=' + content;

serialize(), serializeArray()

```
//[3]
var url = '/memo/ajaxWrite.do';
var data = { name: name, content: content };
$('#result').load(url, data);

//[4]
$('#result').load('/memo/ajaxWrite.do', $(this).serialize());

//[5]
$('#result').load('/memo/ajaxWrite.do', $(this).serializeArray());
```

serialize(), serializeArray()의 차이

```
<script>
        $(document).ready(function () {
            $('#frm1').submit(function (event) {
                var serialize = $(this).serialize();
                var serializeArray = $(this).serializeArray();
                // 출력
                $('<p></p>').text(serialize).appendTo('#result'); //쿼리 문자열을 리턴
                $('<p></p>').text(serializeArray).appendTo('#result'); //배열로 리턴

                // 기본 이벤트 제거
                event.preventDefault();
            });
        });
    </script>
<form id="frm1">
        이름 : <input type="text" id="name" name="name"/><br>
        메모 : <input type="text" id="content" name="content" size="50" /><br>
        <input type="submit"  value="입력"><br>
        <h2>결과</h2>
        <div id="result"></div>
</form>
```

Note
- 폼 요소는 반드시 name 속성을 가지고 있어야 직렬화에 포함됨.
- checkboxes 와 radio button 은 선택된 것만 포함됨
- type='file' 요소는 포함되지 않는다

serialize(), serializeArray()의 차이

```
$('<p></p>').text(serialize).appendTo('#result');
⇒쿼리 문자열을 리턴

$('<p></p>').text(serializeArray).appendTo('#result');
=> 배열로 리턴
```

```
serializeArray() 가 리턴한 객체의 형태
[
        {name:'hong', content:'hi'}
]
```

아이디 중복확인하기

회원 가입

성명	홍길동
회원ID	h 아이디 규칙에 맞지 않습니다

회원 가입

성명	홍길동
회원ID	hong 이미 등록된 아이디

회원 가입

성명	홍길동
회원ID	hong2 사용가능한 아이디

```
<script type="text/javascript">
  $(document).ready(function() {
            $('#userid').keyup(function () {
                        var data=$('#userid').val();
                        if(validate_userid(data) && data.length>=2){
                                    data='userid='+data;
                                    $.ajax({
                                                type:"POST",
                                                url:"<c:url value='/member/ajaxCheckId.do' />",
                                                data:data,
                                                success: function (result) {
                                                            if(result){
                                                                        output = "이미 등록된 아이디";
                                                                        $("#chkId").val('N');
                                                            }else{
                                                                        output = "사용가능한 아이디";
                                                                        $("#chkId").val('Y');
                                                            }
                                                            $('#message').text(output);
                                                }
                                    });
                        }else{
                                    $('#message').text("아이디 규칙에 맞지 않습니다");
                                    $("#chkId").val('N');
                        }
            });
    });
```

```
    function validate_userid(uid)
    {
      var pattern= new RegExp(/^[a-z0-9_]+$/g);
      return pattern.test(uid);
    }
</script>
```

var pattern= new RegExp(/^[a-z0-9_]+$/g);
⇒ 정규식 : 텍스트 필드의 값은 a에서 z 사이의 문자, 0에서 9사이의 숫자나 밑줄문자(_)로 시작하거나
끝나야 한다는 의미(^ 는 시작, $는 끝을 의미함)

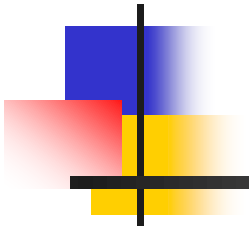
닫기 대괄호 "] " 뒤의 + 기호는 이 패턴이 한 번 또는 그 이상 반복된다는 의미

=> 즉, 문자, 숫자, 또는 밑줄문자(_)를 사용자 아이디 입력란에 여러 번 입력할 수 있다

```
<p>
        <label for="userid">회원ID</label>
        <input type="text" name="userid" id="userid">
                <!--<input type="button" value="중복확인" name="btnChkId"
                        onclick="useridCheck()" title="새창열림"> -->

        <span id="message"></span>
        <form:errors path="userid"/>
</p>

<input type ="hidden" name="chkId" id="chkId">
```

MemberController

```
@ResponseBody
@RequestMapping("/ajaxCheckId.do")
public boolean ajaxCheckId2(@RequestParam(required=false) String userid){
		//1. 파라미터
		logger.debug("ajax 이용2 - 아이디 중복 확인 : userid={}", userid);

		//2. db작업 - select
		boolean isExistId =false;
		if(userid!=null && !userid.isEmpty()){
			isExistId = memberService.checkUserid(userid);
			logger.debug("아이디 중복 확인 결과, isExistId={}", isExistId);
		}

		return isExistId;
}
```



자바스크립트 Ajax 코드

testClient01.html

```
<html>
<head>
	<script language="javascript">
	var xmlHttp = null;

	function getXMLHttp() {
		if (window.ActiveXObject) {
			try {
				return new ActiveXObject("Msxml2.XMLHTTP");
			} catch(e) {
				try {
					return new ActiveXObject("Microsoft.XMLHTTP");
				} catch(e1) {
					return null;
				}
			}
		} else if (window.XMLHttpRequest) {
			return new XMLHttpRequest();
		} else {
			return null;
		}
	}
```

testClient01.html

```
function load() {
        xmlHttp = getXMLHttp();
        xmlHttp.onreadystatechange = viewMessage;
        xmlHttp.open("GET", "test01.jsp", true);
        xmlHttp.send(null);
}
```

testClient01.html

```
    function viewMessage() {
        if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
                alert(xmlHttp.responseText);
            }
            else {
                alert("실패: "+ xmlHttp.status);
            }
        }
    }
    </script>
</head>
<body>
<h2>Ajax 예제</h2>
<input type="button" value="전 송" onclick="load()"/>
</body>
</html>
```