jQuery AJAX

- 13장에서 만든 연락처 서비스를 활용해 jQuery AJAX 기능을 살펴봄
  - 요청 기능을 제공하는 5개의 단축 메서드와 1개의 저수준 메서드
    - $.ajax()
    - $.get(), $.post()
    - $.getJSON()
    - $.getScript()
    - $.prototype.load()

저수준 인터페이스 메서드(1)

$.ajax() 메서드

- 저수준 메서드이므로 이 메서드를 이용하면 모든 AJAX 관련 처리가 가능하다
- 사용방법이 조금 복잡하지만 단축 메서드로 수행할 수 없는 작업인 경우에는 꼭 필요하다.
- 기본 사용 방법

  - $.ajax(url, settings) → 리턴값 : jqXHR
  - $.ajax(settings) → 리턴값 : jqXHR

    url　요청하고자 하는 서비스의 주소

    settings　자바스크립트 객체. 요청 정보, 옵션 정보 등을 포함한다.

저수준 인터페이스 메서드(2)

▪ Settings 옵션 값들

- **url** 요청하고자 하는 서비스 URL(기본값 : 현재 페이지).
- **type** 요청 메서드. GET, POST, PUT, DELETE 등을 지정한다(기본값 : GET).
- **async** 비동기 전송을 수행할지를 지정. 동기적인 처리를 원할 경우 이 값을 false로 지정한다(기본값 : true).
- **data** 서버로 전송하는 데이터. 문자열, 객체, 배열 형식을 지원한다. 문자열인 경우 값을 직접 URL 인코딩해야 한다.

```
ex) String : "keyword=%EC%98%A4&mode=2"
    객체 : { keyword:"오", mode:2 }
    배열 : [ { name:"keyword", value: "오" }, { name:"mode", value: "2" } ]
```

- **contentType** 요청 시 서버로 전달하는 데이터의 형식(기본값 : application/x-www-form-urlencoded)
- **dataType** 서버로부터 응답받기를 원하는 데이터 타입을 지정(기본값 : 자동 추정). xml, json, script, html 등을 지정할 수 있음.
- **statusCode** 서버로부터 응답의 status 코드에 따라 호출할 함수를 지정.

```
ex) $.ajax({
      ...
      statusCode : {
         404 : function() {
            alert("요청하신 경로의 데이터가 존재하지 않습니다");
         }
      }
    })
```

- **timeout** 요청 제한 시간(ms). 지정한 시간 이내에 응답하지 않으면 요청을 중단시킨다.
- **headers** 요청 시에 전달할 헤더값을 객체 형태로 지정.

```
ex) $.ajax({
       headers : { Authorization:"[credential]",  Accept : "application/json" }
    });
```

- **beforeSend** 요청을 서버로 전송하기 전에 실행될 콜백 함수를 지정. 콜백 함수의 두 번째 파라미터는 요청 시에 사용한 옵션 정보이다.

```
ex) beforeSend : function(xhr, settings) {  }
```

- **success** 요청이 성공했을 때 실행될 콜백 함수를 지정. 콜백 함수의 첫 번째 파라미터는 수신한 데이터이다.

```
ex) success : function(data, status, xhr) {  }
```

- **error** 요청이 실패했을 때 실행될 콜백 함수를 지정. 콜백 함수의 두 번째 파라미터는 상태 정보를 "timeout", "error", "abort"와 같이 문자열로 전달한다. 세 번째 파라미터는 에러 정보를 문자열로 전달한다.

```
ex) error : function(xhr, status, error) {  }
```

- **complete** 요청이 완료되었을 때 실행될 콜백 함수를 지정. 요청의 성공/실패와 관계없이 항상 실행되는 콜백 함수다. 두 번째 파라미터는 상태 정보를 전달하며 "success", "error", "abort", "parseerror" 등의 문자열을 가질 수 있다.

```
ex) function(xhr, status) {  }
```

- 이 속성들을 모두 암기하는 것은 필요하지 않다. 사용해나가면서 파악하자.

저수준 인터페이스 메서드(3)

- 지금부터 작성하는 예제는 13장에서 작성한 서비스 이용
  - 크로스 도메인 문제로 인해 웹서버 없이 작동할 때는 정상적으로 데이터를 수신할 수 없다
  - 따라서 이클립스에서 Dynamic Web Project를 추가하고 예제를 작성하도록 하자.
  - 14~16장까지의 예제는 Eclipse로 작성한다.

저수준 인터페이스 메서드(4)

- 예제 14-01 : 기본 예제(GET 방식)
  - GET /contact/list.jsp?pageno=2&pagesize=5 로 요청함.
  - data 옵션을 "pageno=2&pagesize=5"으로 지정해도 되지만 예제와 같이 자바스크립트 객체를 전달하는 것이 더 편리함. 특히 인코딩을 직접하지 않아도 됨.
  - success 콜백함수는 인자가 3개가 있지만 data 이외의 인자를 사용하지 않는다면 생략해도 됨.

```
09:     $.ajax({
10:         url : "/contact/list.jsp",
11:         type : "GET",
12:         data : { pageno:2, pagesize:5 },
13:         success : function(data) {
14:             console.log(data);
15:         }
16:     });
```

저수준 인터페이스 메서드(5)

- 예제 14-02 : POST 방식 연락처 추가
  - data를 전달하는 방법이 Query String이 아니라 HTTP Body가 됨.

```
03:     var param = {};
04:     param.name = "홍길동";
05:     param.tel = "010-7788-7788";
06:     param.address = "제주도";
07:
08:     $.ajax({
09:         url : "/contact/add.jsp",
10:         type : "POST",
11:         data : param,
12:         success : function(data) {
13:             console.log(data);
14:         }
15:     });
```

저수준 인터페이스 메서드(6)

contentType의 지정

- contentType : 상대방에게 전달하는 데이터 형식 지정
  - jQuery AJAX 측면에서는 브라우저가 서버에 전달하는 데이터 형식을 가리킨다.
  - 한번에 여러건의 데이터를 전달하는 경우에는 application/x-www-form-urlencoded 형식보다 더 편리함.

```
03:     var send_data = {
04:         contacts : [
05:             { no:13, name:"오바마", tel : "010-9090-8989", address : "미국" },
06:             { no:14, name:"힐러리", tel : "010-9090-8988", address : "미국" },
07:             { no:15, name:"샌더스", tel : "010-9090-8987", address : "미국" }
08:         ]
09:     };
10:
11:     $.ajax({
12:         url : "/contact/update_batch.jsp",
13:         type : "POST",
14:         contentType : "application/json",
15:         data : JSON.stringify(send_data),
16:         success : function(data) {
17:             console.log(data);
18:         }
19:     });
```

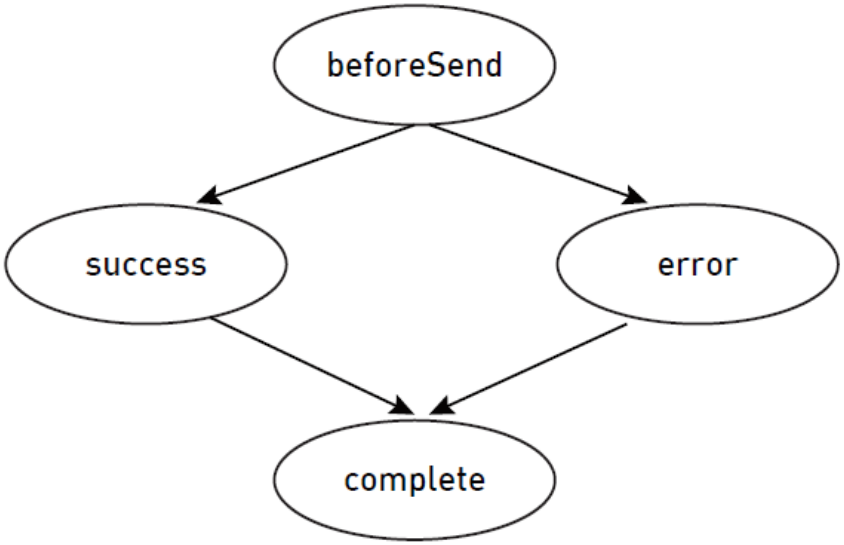
저수준 인터페이스 메서드(6)

다양한 콜백 함수 사용

- success 콜백 함수 이외의 콜백 함수
  - beforeSend, error, complete
- 콜백함수의 실행 순서
  - 정상 상황

    beforeSend -> success -> complete
  - 에러 상황

    beforeSend -> error -> complete



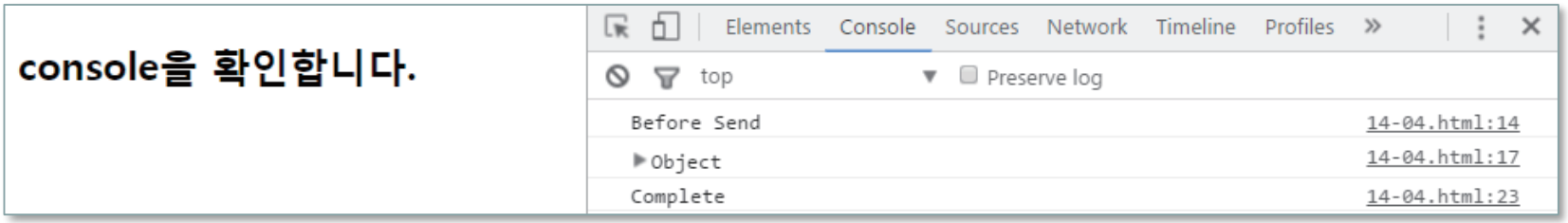
저수준 인터페이스 메서드(7)

- 예제 14-04 : 콜백함수

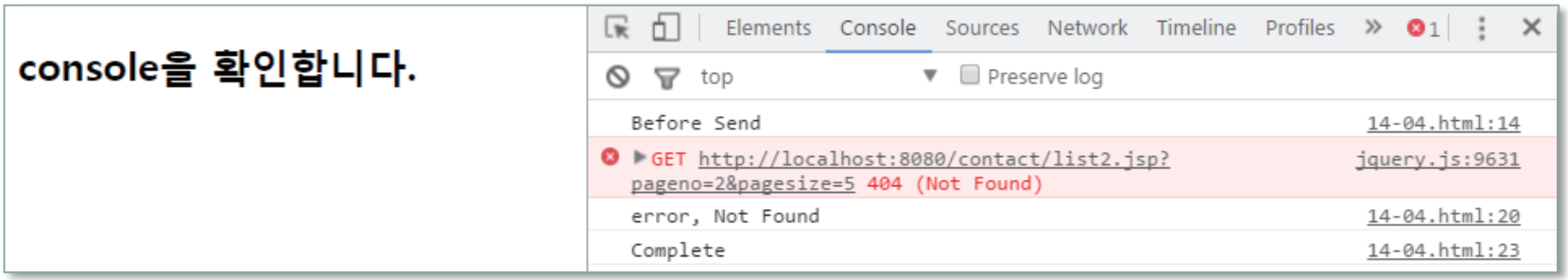
```
03:     $.ajax({
04:         url : "/contact/list.jsp",
05:         type : "GET",
06:         data : { pageno:2, pagesize:5 },
07:         beforeSend : function(xhr, settings) {
08:             console.log("Before Send");
09:         },
10:         success : function(data) {
11:             console.log(data);
12:         },
13:         error : function(xhr, status, error) {
14:             console.log(status + ", " + error);
15:         },
16:         complete : function(xhr, status) {
17:             console.log("Complete");
18:         }
19:     });
```

저수준 인터페이스 메서드(8)

- 예제 14-04 실행 결과
  - 정상적인 실행시



  - 오류 발생시



저수준 인터페이스 메서드(9)

timeout의 지정

- 서버로 요청 후 서버로부터 응답이 오지 않는다면?
  - 브라우저는 계속해서 응답을 기다린다.
- timeout 속성
  - 일정 시간 이내에 서버로부터의 응답이 오지 않으면 연결을 중단(abort) 시킨다.
- 13장 프로젝트에 예제 추가(list_long.jsp)
  - list.jsp를 복사한 다음 Thread.sleep(2000); 코드 추가

[예제 14-05 : list_long.jsp 작성]

```
01: <%@ page language="java" contentType="application/json; charset=utf-8"
02:     pageEncoding="utf-8"%>
03: <%@ page import="java.util.*" %>
04: <%@ page import="com.ruby.contact.domain.*" %>
05: <%@ page import="com.ruby.contact.util.*" %>
06: <%
07:     Thread.sleep(2000);
08:     String strPageno = request.getParameter("pageno");
09:     String strPagesize = request.getParameter("pagesize");
......
27: %><%=json %>
```

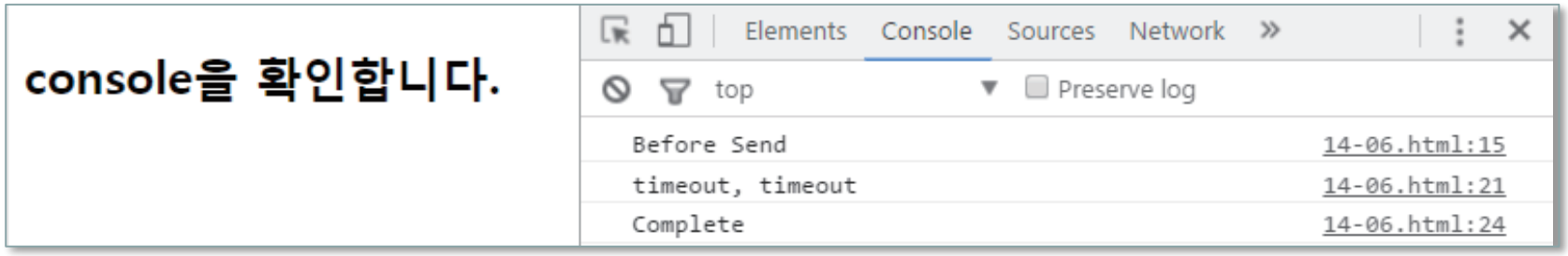
저수준 인터페이스 메서드(10)

- 예제 14-06 : 예제 14-04 활용. timeout 옵션만 추가

```
03:     $.ajax({
04:         url : "/contact/list_long.jsp",
05:         type : "GET",
06:         timeout : 1900,
07:         data : { pageno:2, pagesize:5 },
08:         beforeSend : function(xhr, settings) {
09:             console.log("Before Send");
10:         },
11:         success : function(data) {
12:             console.log(data);
13:         },
14:         error : function(xhr, status, error) {
15:             console.log(status + ", " + error);
16:         },
17:         complete : function(xhr, status) {
18:             console.log("Complete");
19:         }
20:     });
```

저수준 인터페이스 메서드(11)

- 예제 14-06 실행 결과



- timeout 속성과 관련하여 주의할 점
  - 추가, 수정, 삭제와 같은 작업에는 적용하기 어렵다
  - 데이터를 추가하도록 요청한 뒤 응답 대기 시간을 초과하여 timeout에 의해 중단(abort) 처리되었다 하더라도 실제로는 데이터 추가가 완료되었을 수 있다
  - 왜냐하면 데이터 추가 요청은 정상적으로 처리되고, 그 후 응답하는 과정에서만 중단되었을 수 있기 때문이다

단축 메서드(1)

- $.ajax()가 강력하긴 하지만 사용방법이 복잡하다.
  - 그렇기 때문에 간단하게 사용할 수 있는 단축 메서드가 필요

[**표 14-01** : 단축 메서드]

메서드	설명
$.get( )	GET 방식으로 요청을 전송하고 서버로부터 데이터를 로딩한다.
$.post( )	POST 방식으로 요청을 전송하고 서버로부터 데이터를 로딩한다.
$.getJSON( )	GET 방식으로 JSON 형식의 인코딩된 데이터를 로딩한다. $.get( ) 기능과 크로스 도메인 문제를 해결하기 위한 JSONP 기법을 지원한다.
$.getScript( )	GET 방식으로 서버로부터 자바스크립트 파일을 로딩한 후 실행시킨다.
load( )	서버로부터 응답 받은 데이터를 선택된 HTML 요소의 콘텐츠로 대체한다.

- $.get(url, successCallback)
- $.get(url, data, successCallback)
- $.get(settings) jQuery 1.12/2.2에서 추가된 사용 방법

단축 메서드(2)

- 예제 14-07

```
04:     $.get("/contact/list.jsp", function(data) {
05:         console.log(data);
06:     });
07:
08:     var param = {};
09:     param.name = "지숙";
10:     param.tel = "010-5656-4545";
11:     param.address = "충남";
12:     $.post("/contact/add.jsp", param, function(data) {
13:         console.log(data);
14:     });
```

```
16:     //1.12, 2.2 버전부터 지원하는 사용 형태
17:     $.get({
18:         url : "/contact/list_long.jsp",
19:         data : { pageno:2, pagesize:5 },
20:         timeout : 1900,
21:         success : function(data) {
22:             console.log(data);
23:         },
24:         error : function(xhr, status, error) {
25:             console.log(status + ", " + error);
26:         }
27:     });
```

단축 메서드(3)

- $.getJSON()
  - $.get() + JSONP 지원 기능
  - xml, xhtml 등의 데이터는 처리하지 못함.
  - 자세한 내용은 15장에서 다룸
- $.getScript()
  - 동적으로 자바스크립트 파일을 로드하기 위한 메서드
  - <script></script> 태그를 이용하지 않고도 자바스크립트 라이브러리를 로딩할 수 있음
  - 자바스크립트 라이브러리가 필요한 시점으로 로딩할 수 있음
  - 한번에 많은 라이브러리를 로딩하는 것을 막는 목적으로 사용할 수도 있음

    - $.getScript(url)
    - $.getScript(url, successCallback)

단축 메서드(4)

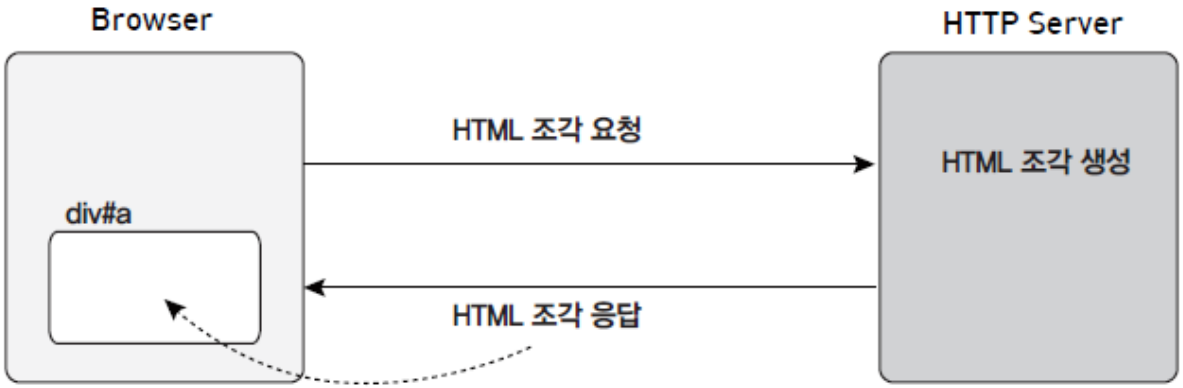
- 예제 14-09 : $.getScript() 예제

[예제 14-09 : $.getScript()의 사용]

```
<script type="text/javascript" src="http://code.jquery.com/jquery-3.1.0.js"></script>
<script type="text/javascript">
$(document).ready(function() {
   $.getScript("js/htmlencoding.js", function() {
      $.htmlencode("<h1>Hello</h1>");
   });
         ...
});
</script>
```

단축 메서드(5)

- load() 메서드
  - 요청후 응답 결과를 선택된 요소의 컨텐트로 설정함.
  - 요청의 단위가 HTML 문서 전체가 아니라 HTML 조각임.
  - 이 방법은 HTTP Server가 UI를 직접 생성한다
  - JSON이나 XML과는 달리 HTML 조각 정보엔 HTML 마크업과 각종 디자인 정보 등이 포함되므로 데이터 전송 트래픽이 늘어나므로 비효율적이다.
  - 다른 유형의 앱(모바일 앱 등)에서 데이터를 재사용하기가 쉽지 않다.




```
$("#a").load(url, param, complete)
```

jqXHR 객체(1)

- jQuery AJAX 메서드의 리턴값은?
  - jqXHR(jQuery XMLHttpRequest)객체!!
    - XHR 객체의 속성을 그대로 가지고 있으면서 jQuery가 제공하는 Promise 인터페이스를 따르고 있음 --> jQuery 지연(Deferred) 객체
    - 비동기 처리 기능을 제공함.
    - 지연 객체에 대해서는 18장에서 제공함
  - 지연객체를 사용하면?
    - 콜백함수의 중첩 구조는 간단해지고 예외 처리도 간단해진다.

jqXHR 객체(2)

예제 14-10

- done, fail, always 이벤트 처리
  - success, error, complete 콜백함수를 대신함.
- 여러개의 서비스를 순서대로 호출할 때 장점이 드러남.
  - 콜백 지옥(Callback Hell) 문제 해결
  - 예제 14-11과 14-12를 살펴보자.

```
03: $(document).ready(function() {
04:     var jqxhr = $.ajax({
05:         url : "/contact/list_long.jsp",
06:         type : "GET",
07:         data : { pageno:2, pagesize:5 },
08:     });
09:
10:     jqxhr.done(function(data, status, xhr) {
11:         console.log(data);
12:     });
13:
14:     jqxhr.fail(function(xhr, status, error) {
15:         console.log(status + ", " + error);
16:     });
17:
18:     jqxhr.always(function() {
19:         console.log("Complete!!");
20:     });
21: });
```

jqXHR 객체(3)

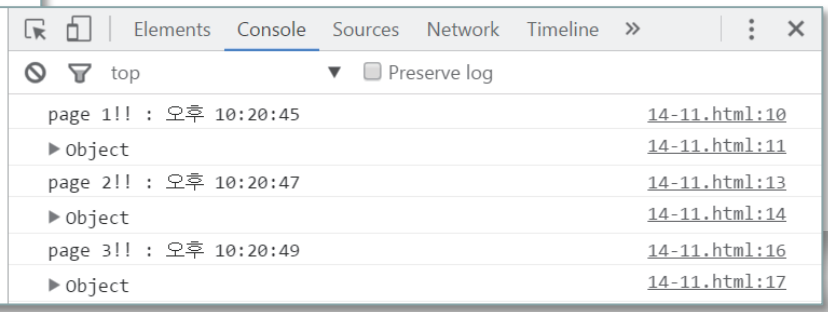
예제 14-11 : 콜백 지옥

- 문제점!!
  - 중첩구조가 복잡해지면서 변수의 스코프 처리에 문제가 발생할 수 있음
  - 가독성이 떨어지고 유지보수가 어려움
  - 예외처리까지 이루어질 경우 코드의 복잡도가 증가하고 디버깅이 쉽지 않음

[예제 14-11 : 콜백 지옥(Callback Hell)]

```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:     $.get("/contact/list_long.jsp", {pageno:1}, function(page1_data) {
04:         console.log("page 1!! : "+ (new Date()).toLocaleTimeString());
05:         console.log(page1_data);
06:         $.get("/contact/list_long.jsp", {pageno:2}, function(page2_data) {
07:             console.log("page 2!! : "+ (new Date()).toLocaleTimeString());
08:             console.log(page2_data);
09:             $.get("/contact/list_long.jsp", {pageno:2}, function(page3_data) {
10:                 console.log("page 3!! : "+ (new Date()).toLocaleTimeString());
11:                 console.log(page3_data);
12:             });
13:         });
14:     });
15: });
16: </script>
```

console을 확인합니다.



jqXHR 객체(4)

예제 14-12

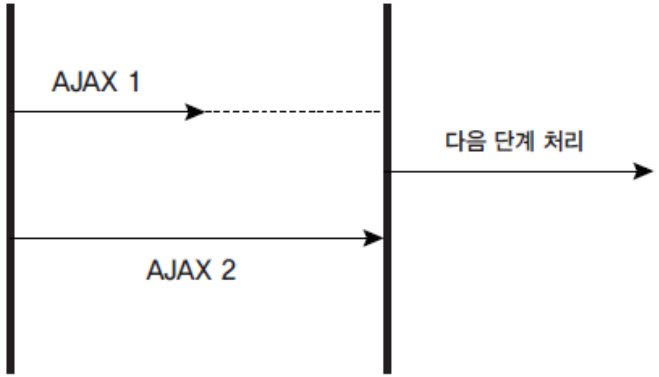
- jQuery Promise를 사용한 문제 해결
  - then() 메서드 내부에서 마지막 부분에 다음 AJAX 호출한 후 jqXHR객체를 리턴
  - 리턴된 jqXHR 객체를 이용해 다시 then() 메서드 처리

[예제 14-12 : promise를 사용한 콜백 지옥 해결]

```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:
04:     $.get("/contact/list_long.jsp", {pageno:1}).then(function(data) {
05:         console.log("promise page 1!! : "+ (new Date()).toLocaleTimeString());
06:         console.log(data);
07:         return $.get("/contact/list_long.jsp", {pageno:2});
08:     }).then(function(data) {
09:         console.log("promise page 2!! : "+ (new Date()).toLocaleTimeString());
10:         console.log(data);
11:         return $.get("/contact/list_long.jsp", {pageno:3});
12:     }).then(function(data) {
13:         console.log("promise page 3!! : "+ (new Date()).toLocaleTimeString());
14:         console.log(data);
15:     });
16: });
17: </script>
```

jqXHR 객체(5)

jqXHR 객체의 또 다른 장점

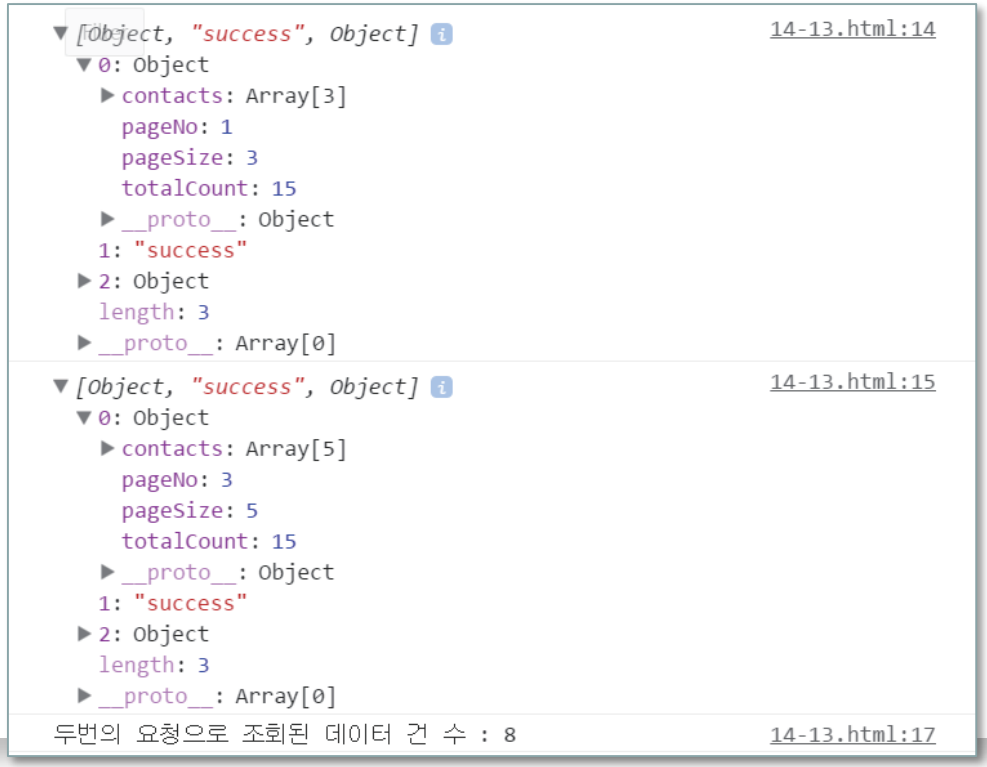


- 예제 14-13

```
04:     var xhr1 = $.get("/contact/list_long.jsp", {pageno:1});
05:     var xhr2 = $.get("/contact/list.jsp", {pageno:3, pagesize:5 });
06:
07:     $.when(xhr1, xhr2).done(function(res1, res2) {
08:         console.log(res1);
09:         console.log(res2);
10:         var totalcount = res1[0].contacts.length + res2[0].contacts.length;
11:         console.log("두 번의 요청으로 조회된 데이터 건수의 합 : " + totalcount);
12:     });
```

jqXHR 객체(6)

- 예제 14-13 실행 결과
  - done() 메서드의 콜백함수는 $.when() 처리하는 AJAX 호출 결과의 리스트이다.
  - 각각의 호출 결과는 배열구조
    - 0번 인덱스 값 : 응답결과
    - 1번 인덱스 값 : 상태(status)
    - 2번 인덱스 값 : 사용된 jqXHR 객체
  -



jqXHR 객체(7)

jqXHR 객체 정리

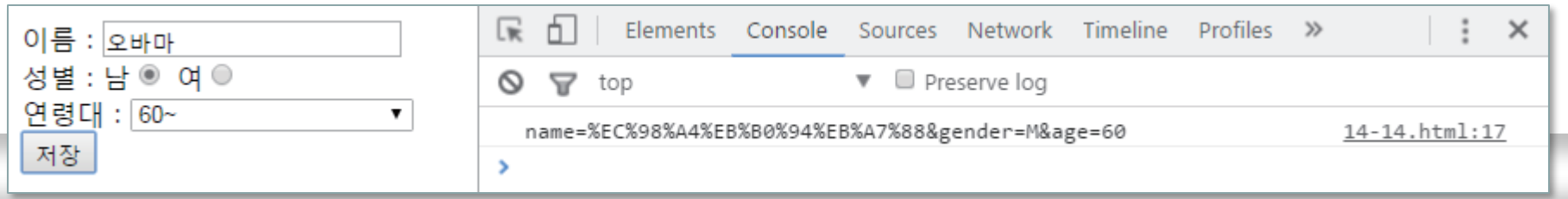
- 콜백 함수의 중첩 구조가 복잡하지 않다면 기존의 콜백 함수 패턴을 사용해도 문제될 것이 없음
- 함수 중첩이 부담스럽다면 jqXHR 객체를 이용한 promise 패턴을 적용할 것을 권장
  - 코드의 가독성도 높이고 유지 보수도 편리하기 때문이다.

헬퍼 함수(1)

사용자로부터 입력받는 입력필드가 많다면?

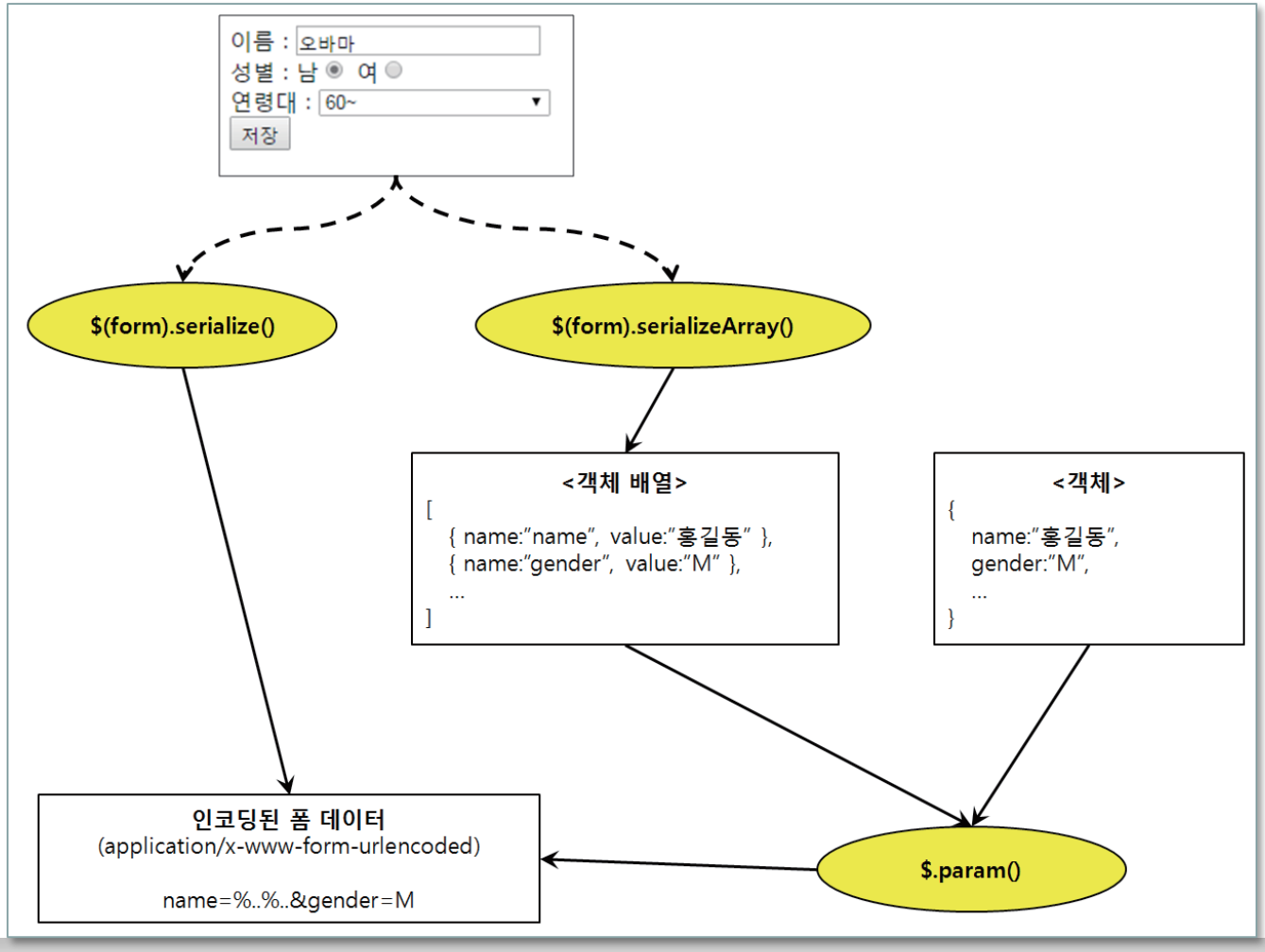
- application/x-www-form-urlencoded 형식의 문자열을 직접 만드는 것은 귀찮고 힘든 일
  - 한글 및 특수 문자는 URL Encoding까지 수행해야 함.
  - 예제 14-14
    - 예제 14-14는 입력필드가 단3개이므로 간단하지만 입력필드가 10-20개 가량 된다면?

```
09:     $("#save").click(function() {
10:         var name = $("#name").val();
11:         var gender = $("input.gender:checked").val();
12:         var age = $("#age > option:selected").val();
13:
14:         var param = "name=" + encodeURIComponent(name);
15:         param += "&gender="+gender + "&age=" + age;
16:
17:         console.log(param);
18:     });
```



헬퍼 함수(2)

헬퍼함수 관계



헬퍼 함수(3)

헬퍼 함수 종류

- 앞 슬라이드의 그림과 비교!!

serialize( )	폼 요소를 선택하고 호출했을 때 입력 필드의 name 특성(attribute)을 이용해 인코딩된 폼 데이터를 생성한다.
serializeArray( )	폼 요소를 선택하고 호출했을 때 입력 필드의 name 특성(attribute)을 이용해 객체로 구성된 배열을 생성한다.
$.param( )	객체의 배열이나 객체를 파라미터로 전달하여 인코딩된 폼 데이터를 생성한다.

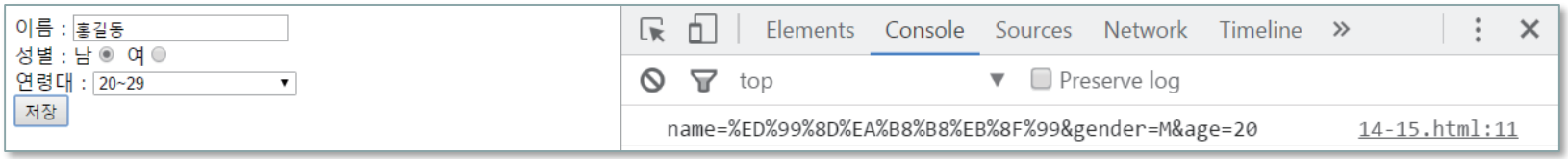
- serialize()와 serializeArray() 함수는 반드시 폼 요소를 선택해야 하고 name 특성을 이용한다는 점을 기억해야 한다
  - name 특성을 사용하지 않으면 인코딩된 폼 데이터를 생성할 때 값이 누락된다

헬퍼 함수(4)

예제 14-15

- 예제 14-14의 HTML 마크업 활용하여 작성

```
09:     $("#save").click(function() {
10:         var param = $("#f1").serialize();
11:         console.log(param);
12:     });
```



- serializeArray() 메서드

```
> var arr = $("#f1").serializeArray();
<- undefined
> JSON.stringify(arr)
<- "[{"name":"name","value":"홍길동"},{"name":"gender","value":"M"},
  {"name":"age","value":"20"}]"
> $.param(arr)
<- "name=%ED%99%8D%EA%B8%B8%EB%8F%99&gender=M&age=20"
```

```
> var obj = { name:"smith", address:"NY" };
<- undefined
> $.param(obj)
<- "name=smith&address=NY"
```

전역 AJAX 이벤트 처리(1)

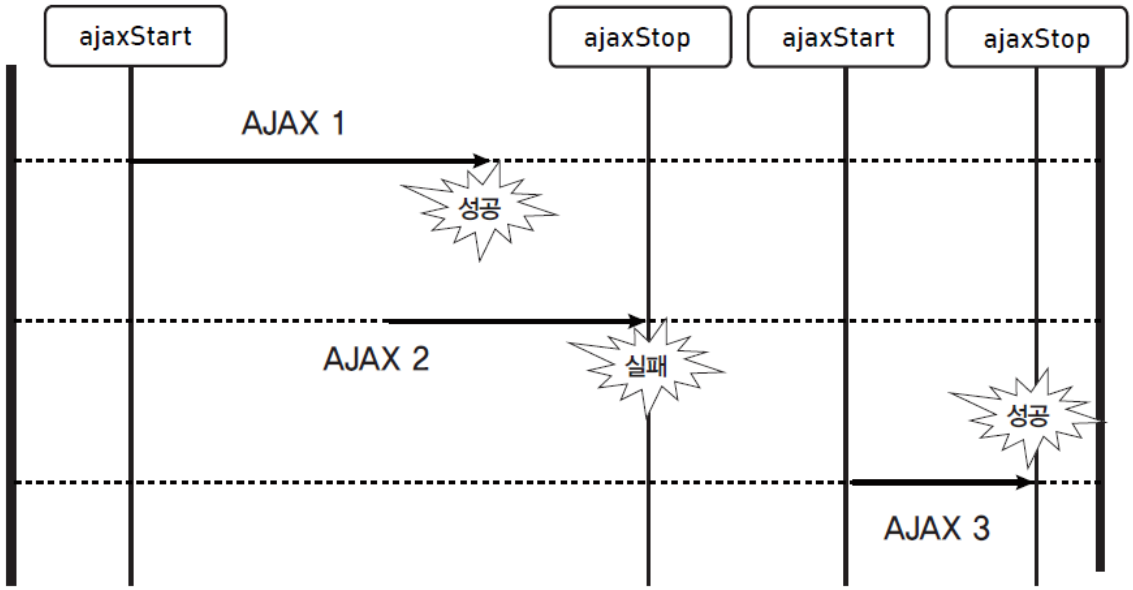
- jQuery AJAX의 콜백 함수는...
  - AJAX 함수 호출코드를 작성할 때마다 매번 콜백함수를 작성해야 함.
- 전역 AJAX 이벤트 처리
  - 페이지 수준에서 모든 AJAX 요청에 대한 공통의 이벤트 처리 기능을 지원

[**표 14-02** : 전역 AJAX 이벤트 처리 메서드]

이벤트	설명
ajaxStart(handler)	처리 중인 AJAX 요청이 없는 상태에서 처음으로 AJAX 요청이 일어날 때 발생되는 이벤트. 이미 AJAX 요청이 진행 중인 상태에서 새로운 AJAX 요청을 하는 경우에는 이 이벤트가 발생하지 않는다. handler : function( ) { }
ajaxSend(handler)	각각의 AJAX 요청이 전송되기 전에 발생되는 이벤트. 현재 AJAX 요청이 진행 중이라 하더라도 매번 이벤트가 발생된다. handler : function(event, xhr, options) { } event : 이벤트 객체 xhr : jQuery XHR 객체 options : ajax 요청 옵션 객체
ajaxSuccess(handler)	각각의 AJAX 요청에 대한 처리가 성공인 경우에 발생되는 이벤트 handler : function(event, xhr, options, data) { } data : 서버로부터 응답된 데이터
ajaxError(handler)	각각의 AJAX 요청에 대해 오류가 일어났을 때 발생되는 이벤트 handler : function(event, xhr, options, error) { } error : 발생된 오류 메시지
ajaxComplete(handler)	각각의 AJAX 요청이 완료되었을 때 발생되는 이벤트. 성공, 오류 여부에 관계없이 항상 발생된다. handler : function(event, xhr, options) { }
ajaxStop(handler)	처리 중인 AJAX 요청이 모두 완료된 상태가 되었을 때 발생되는 이벤트. handler : function( ) { }

전역 AJAX 이벤트 처리(2)

전역 AJAX 이벤트 개념



[AJAX 1 요청]

로컬 콜백 함수	beforeSend	→	success	→	complete	
전역 AJAX 이벤트		ajaxSend →		ajaxSuccess →		ajaxComplete

[AJAX 2 요청]

로컬 콜백 함수	beforeSend	→	error	→	complete	
전역 AJAX 이벤트		ajaxSend →		ajaxError →		ajaxComplete

전역 AJAX 이벤트 처리(3)

예제 14-16 : 전역 AJAX 이벤트와 로컬 이벤트

[예제 14-16]

```
01: <script type="text/javascript" src="http://code.jquery.com/jquery-3.1.0.js"></script>
02: <script type="text/javascript">
03: $(document).ready(function() {
04:   $(document).ajaxStart(function() {
05:     console.log("AJAX START!!");
06:   }).ajaxStop(function() {
07:     console.log("AJAX STOP!!");
08:   }).ajaxSend(function(e,xhr,options) {
09:     console.log("AJAX SEND : " + options.url);
10:   }).ajaxSuccess(function(e,xhr,options,data) {
11:     console.log("AJAX SUCCESS : " + options.url);
12:   }).ajaxComplete(function(e,xhr,options) {
13:     console.log("AJAX COMPLETE : " + options.url);
14:   });
15:
16:   $.get({
17:     url : "/contact/list.jsp",
18:     beforeSend : function() {
19:       console.log("1-before send : list.jsp");
20:     },
21:     success : function(data) {
22:       console.log("1-success : list.jsp");
23:     },
24:     complete : function() {
25:       console.log("1-complete : list.jsp");
26:     }
27:   });
28:
29:   $.get({
30:     url : "/contact/list_long.jsp",
31:     beforeSend : function() {
32:       console.log("2-before send : list_long.jsp");
33:     },
34:     success : function(data) {
35:       console.log("2-success : list_long.jsp");
36:     },
37:     complete : function() {
38:       console.log("2-complete : list_long.jsp");
39:     }
40:   })
41: });
42: </script>
```

전역 AJAX 이벤트 처리(4)

- 예제 14-16 실행 결과
  - list.jsp와 list_long.jsp로 요청 : list_long.jsp는 지연시간 2초!!
  - ajaxStart와 ajaxStop은 첫요청이 실행될 때와 모든 AJAX 요청이 완료될 때만 발생함.
  - 항상 각 요청의 콜백 함수가 먼저 실행되고 전역 이벤트가 실행

```
AJAX START!!                                    14-16.html:10
1-before send : list.jsp                        14-16.html:24
AJAX SEND : /contact/list.jsp                   14-16.html:14
2-before send : list_long.jsp                   14-16.html:37
AJAX SEND : /contact/list_long.jsp              14-16.html:14
1-success : list.jsp                            14-16.html:27
AJAX SUCCESS : /contact/list.jsp                14-16.html:16
1-complete : list.jsp                           14-16.html:30
AJAX COMPLETE : /contact/list.jsp               14-16.html:18
2-success : list_long.jsp                       14-16.html:40
AJAX SUCCESS : /contact/list_long.jsp           14-16.html:16
2-complete : list_long.jsp                      14-16.html:43
AJAX COMPLETE : /contact/list_long.jsp          14-16.html:18
AJAX STOP!!                                     14-16.html:12
```

기타 메서드(1)

- 자주 사용하지는 않지만 알아두면 편리한 메서드
- $.ajaxSetup()
  - 요청시 전달되는 옵션의 기본값을 변경(설정)하는 메서드
  - 권장하지 않은 --> 모든 AJAX 요청에 영향을 줌

```
09:     $.ajaxSetup({
10:         timeout : 1900,
11:         //global : false,
12:         error : function(jqxhr, status, error) {
13:             console.log("ERROR : " + error);
14:         }
15:     });
16:
17:     $.get("/contact/list_long.jsp", function(data) {
18:         console.log(data);
19:     });
```

```
ERROR : timeout                                    14-17.html:18
AJAX STOP!!                                        14-17.html:11
```

기타 메서드(2)

$.ajaxPrefilter()

- 사용자 정의 AJAX 옵션을 사용하고자 할 때
- 기존 AJAX 옵션의 기능을 변경하고자 할 때

$.ajaxPrefilter(handler_function);

handler_function function(options, originalOptions, jqxhr)

options AJAX 요청된 옵션 정보, 기본값까지 모두 적용한 옵션 정보이다.

originalOptions $.ajax() 메서드에 의해 직접 요청된 옵션 정보. 기본값이 반영되는 않는다.

jqxhr jQuery XHR 객체

- 예제 14-18
  - 데이터 중복 입력 방지를 위한 abortOnRetry 옵션을 추가로 정의하여 사용함.
  - abortOnRetry 옵션이 true 이면 currentRequest 객체를 조회하여 진행중인 요청이 있다면 중단(abort) 시킴.

데이터 중복 입력 방지

데이터 중복 입력을 막기 위해 $.ajaxPrefilter( )를 사용할 수도 있지만, 처리가 완료되기 전까지 화면을 비활성화시켜서 원천적으로 입력을 막는 방법도 있다.

기타 메서드(3)

- 예제 14-18

[예제 14-18]

```
01: <script type="text/javascript" src="http://code.jquery.com/jquery-3.1.0.js"></script>
02: <script type="text/javascript">
03: $(document).ready(function() {
04:     var currentRequests = {};
05:
06:     $.ajaxPrefilter(function( options, originalOptions, jqXHR ) {
07:         if ( options.abortOnRetry ) {
08:             if ( currentRequests[ options.url ] ) {
09:                 jqXHR.abort();
10:             } else {
11:                 currentRequests[ options.url ] = jqXHR;
12:             }
13:         }
14:     });
15:
16:     $(document).ajaxComplete(function(e,jqxhr,options) {
17:         if (options.abortOnRetry) {
18:             delete currentRequests[options.url];
19:         }
20:     });
21:
22:     $.get({
23:         url : "/contact/list_long.jsp",
24:         abortOnRetry : true,
25:         success : function(data) {
26:             console.log("## First Request!!");
27:             console.log(data);
28:         },
29:         error : function(xhr, status, error) {
30:             console.log("## First Request Error!!");
31:         }
32:     });
33:
34:     $.get({
35:         url : "/contact/list_long.jsp",
36:         abortOnRetry : true,
37:         success : function(data) {
38:             console.log("## Second Request!!");
39:             console.log(data);
40:         },
41:         error : function(xhr, status, error) {
42:             console.log("## Second Request Error!!");
43:         }
44:     });
45:
46: });
47: </script>
```

정리

- AJAX 요청 메서드
  - $.ajax()
  - $.get(), $.post(), $.getJSON(), $.getScript(), $.prototype.load()
- AJAX 요청 옵션
  - 모두 다 암기할 필요는 없지만 이 책에서 다루는 내용 정도는 숙지해야 함.
  - 콜백 함수는 반드시 숙지하자.
- 기타 기능
  - 헬퍼 함수
  - 전역 AJAX 이벤트

- 14장에서는 jQuery AJAX의 기능만 다루었음
  - 화면 UI가 아닌 통신 기능에 집중하기 위해서
- 화면 UI 기능은 16장으로 잠시 미루어두자.