유틸리티 함수

jQuery의 유틸리티 함수

- jQuery 함수의 정적 메서드처럼 사용하는 것이 많음
- 예) $.extend()
- 18장에서는 앞에서 다루지 않았던 유틸리티 함수를 살펴 봄

배열과 객체 관련 함수(1)

배열과 객체 관련 함수 목록

[표 18-01]

함수	설명
$.grep( )	배열의 내용을 콜백 함수를 이용해 필터링한다. $.grep(arr, callback) $.grep(arr, callback, invert)
$.makeArray( )	객체를 배열로 변환한다. $.makeArray(obj)
$.map( )	배열 또는 객체의 내용에 대해 콜백 함수를 이용해 새로운 배열을 생성한다. $.map(arr, callback) $.map(obj, callback)
$.inArray( )	전달한 값이 배열 안에 존재한다면 인덱스 번호를 리턴한다. $.inArray(val, arr) $.inArray(val, arr, fromIndex)
$.uniqueSort( )	배열 안에서 중복된 값을 제거하고 정렬한다. jQuery 2.x까지는 $.unique( ) 함수를 사용했다. $.uniqueSort(arr)

배열과 객체 관련 함수(2)

$.grep() : 예제 18-01

- 배열 중에서 특정한 조건에 맞는 배열만으로 필터링하여 새로운 배열을 생성한다.
- 예제 18-01

```
05:     var data = [
06:        { name:"홍길동", age:17, email:"gdhong@opensg.net" },
07:        { name:"이몽룡", age:16, email:"mrlee@opensg.net" },
08:        { name:"성춘향", age:16, email:"chsung@opensg.net" },
09:        { name:"박문수", age:25, email:"mspark@opensg.net" },
10:        { name:"변학도", age:32, email:"hdbyun@opensg.net" },
11:        { name:"방자", age:19, email:"banja@opensg.net" },
12:        { name:"향단이", age:20, email:"hyangdan@opensg.net" }
13:     ];
14:
15:     var overs = $.grep(data, function(elem, index) {
16:         return elem.age >=20;
17:     });
18:
19:     var str = "";
20:     for (var i=0; i < overs.length; i++) {
21:        str += overs[i].name + ":" +overs[i].age + ", " +overs[i].email + "<br />";
22:     }
23:     $("#result").html(str);
```

박문수:25, mspark@opensg.net
변학도:32, hdbyun@opensg.net
향단이:20, hyangdan@opensg.net

배열과 객체 관련 함수(3)

$.map()

- 배열 또는 객체의 내용을 이용해 새로운 배열을 만든다.
- 예제 18-02

```
02:     var data = [
03:         { name:"홍길동", age:17, email:"gdhong@opensg.net" },
04:         { name:"박문수", age:25, email:"mspark@opensg.net" },
05:         { name:"방자", age:19, email:"banja@opensg.net" },
06:         { name:"향단이", age:20, email:"hyangdan@opensg.net" }
07:     ];
09:     var arr =$.map(data, function(elem, index) {
10:         var clone =$.extend({}, elem);
11:         if (clone.age >= 20) {
12:             clone.status = "over 20";
13:         } else {
14:             clone.status = "not over 20";
15:         }
16:         delete clone.age;
17:         return clone;
18:     });
19:
20:     console.log(data);
21:     console.log(arr);
```

```
▼ [Object, Object, Object, Object]                    18-02.html:27
  ▼ 0: Object
      age: 17
      email: "gdhong@opensg.net"
      name: "홍길동"
    ▶ __proto__: Object
  ▶ 1: Object
  ▶ 2: Object
  ▶ 3: Object
    length: 4
  ▶ __proto__: Array[0]
▼ [Object, Object, Object, Object]                    18-02.html:28
  ▼ 0: Object
      email: "gdhong@opensg.net"
      name: "홍길동"
      status: "not over 20"
    ▶ __proto__: Object
  ▶ 1: Object
  ▶ 2: Object
  ▶ 3: Object
    length: 4
  ▶ __proto__: Array[0]
```

배열과 객체 관련 함수(4)

- $.map() 함수와 유사한 기능을 자바스크립트 배열에서 제공했지만 배열만 지원했다.
  - jQuery $.map()은 배열이 아닌 유사배열에서도 사용가능함.
- 예제 18-03
  - 객체의 속성명과 속성값을 이용해 새로운 배열을 만들어냄

[예제 18-03 : 객체를 배열로 변환하기]

```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:     var obj = { name:"홍길동", age:20, email : "gdhong@opensg.net" };
04:     var arr = $.map(obj, function(prop, key) {
05:         var t = { name:key, value:prop };
06:         return t;
07:     });
08:     var json = JSON.stringify(arr);
09:     console.log(json);
10: });
11: </script>
```

```
[{"name":"name","value":"홍길동"},{"name":"age","value":20},
{"name":"email","value":"gdhong@opensg.net"}]
```

18-03.html:15

배열과 객체 관련 함수(5)

- $.merge()
  - 두개의 배열을 이어붙여 하나의 배열을 만들어냄
  - 배열 객체의 concat() 메서드와 유사. 배열 뿐만 아니라 유사배열에 대해서도 이어붙일 수 있음

타입 관련 함수

타입 관련 함수 목록

[**표 18-02** : 타입 관련 함수]

함수	설명
$.isArray(obj)	객체가 배열인지를 확인한다. 유사 배열은 false를 리턴한다.
$.isFunction(obj)	객체가 함수인지를 확인한다.
$.isEmptyObject(obj)	객체가 속성을 하나도 포함하지 않은 빈 객체인지를 확인한다.
$.isPlainObject(obj)	객체가 Plain Object인지를 확인한다. Plain Object는 객체 리터럴 또는 Object 생성자 함수로 만들어진 객체를 의미한다.
$.isWindow(obj)	객체가 Window 객체인지를 확인한다.
$.isNumeric(value)	인자로 전달한 값이 숫자로 변환 가능한 형식인지를 확인한다.
$.isXMLDoc(obj)	인자로 전달한 XML 문서 형식인지를 확인한다.
$.type(obj)	객체의 타입을 문자열로 리턴한다. typeof 키워드와의 차이점은 내장 객체를 이용해 호출한 경우 "object"가 아닌 "date", "regexp"와 같은 객체 타입을 리턴한다는 것이다.

기타 함수(1)

기타 함수 목록

[**표 18-03** : 기타 함수]

함수	설명
$.now( )	현재 시간 정보를 1970.1.1(epoch)로부터 현재까지 흐른 시간을 밀리초 단위로 표현한 값을 리턴한다. Date 객체의 getTime() 값과 동일하다.
$.trim(str)	전달한 문자열의 앞뒤 공백 문자를 제거한다.
$.noop( )	비어 있는 함수를 리턴한다. 아무것도 수행할 수 없는 함수이다.
$.noConflict( )	$ 식별자를 사용하지 않도록 한다. 이 함수를 호출하고 나면 jQuery라는 식별자만을 사용할 수 있다.
$.proxy( )	특정한 함수에 바인딩되는 this를 지정할 수 있는 기능을 제공한다. 리턴값은 this가 새롭게 바인딩된 함수이다. $.proxy(function, context)

기타 함수(2)

$.proxy()

- 지정한 함수에 파라미터로 전달한 객체를 this로 바인딩한 메서드를 리턴함.
- 자바스크립트의 bind() 메서드와 유사한 기능

$.noop()

- 아무런 기능을 수행하지 않는 빈 함수를 리턴함

[예제 18-04]

```
01: <script type="text/javascript">
02: var message = "전역 메시지";
03: $(document).ready(function() {
04:     var obj = { message:"객체 메시지", v : 100 };
05:     function test() {
06:         console.log(this.message);
07:     }
08:     test();         //"전역 메시지"
09:
10:     //this가 obj로 바인딩된 새로운 함수 리턴
11:     var s = $.proxy(test, obj);
12:     s();          //"객체 메시지"
13: });
14: </script>
```

지연 객체(1)

비동기 콜백 함수

- jQuery AJAX, 효과를 다루면서 비동기 콜백 사용
- 콜백은 어떠한 이벤트(특정 상황의 발생) 시점에 실행하는 함수
  - 처리가 얼마나 걸릴지 예측하기 힘들고
  - 처리가 성공,실패인 경우에 각기 다른 처리가 요구되는 상황일 때 효과적임
- 콜백 함수가 편리하긴 하지만
  - 함수들의 중첩으로 인해 코드가 복잡해지고, 에러처리가 불편한 측면이 있음
  - 이와 같은 상황을 14장에서 jqXHR 객체를 통해 살펴보았음
  - 이런 경우에 지연객체(Deferred Object)가 편리함.

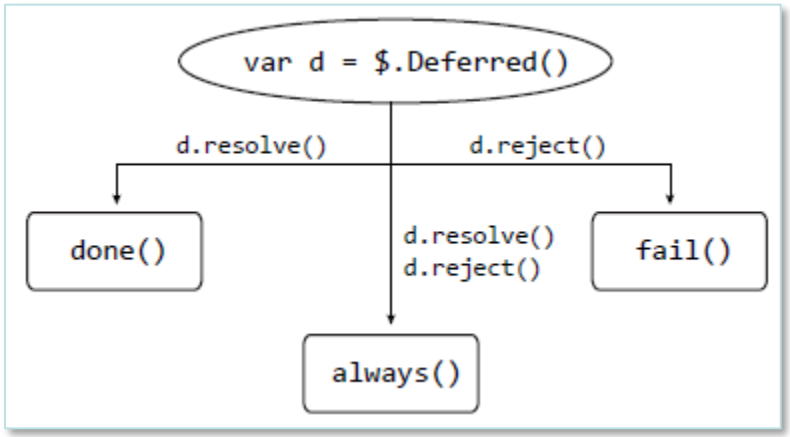
지연 객체(2)

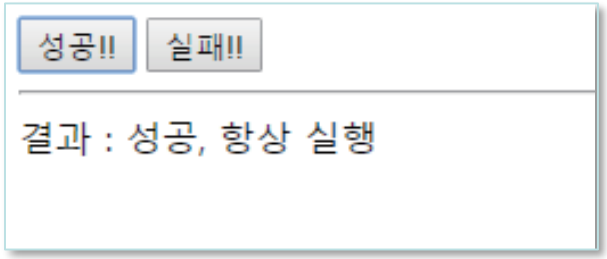
지연 객체(Deferred Object)의 사용

- 지연 객체를 만들기 위해 $.Deferred() 함수를 이용한다는 점을 제외하면 14장에서 다루었던 jqXHR 객체를 이용하는 방법과 동일함
- 예제 18-05

```
04:     var d = $.Deferred();
05:     d.done(function() {
06:         $("#result").html("성공");
07:     }).fail(function() {
08:         $("#result").html("실패");
09:     }).always(function() {
10:         $("#result").append(", 항상 실행")
11:     });
12:
13:     $("#success").click(function() {
14:         d.resolve();
15:     });
16:
17:     $("#fail").click(function() {
18:         d.reject();
19:     });
```

```
24:     <button id="success">성공!!</button>
25:     <button id="fail">실패!!</button>
26:     <hr />
27:     결과 : <span id="result"></span>
```





지연 객체(3)

- 지연 객체를 생성하고 done, fail, always 이벤트를 설정하여 실행할 함수 지정
- 메서드 체이닝(method chaining) 방식으로 호출하거나 다음과 같이 개별적으로 호출할 수 있음

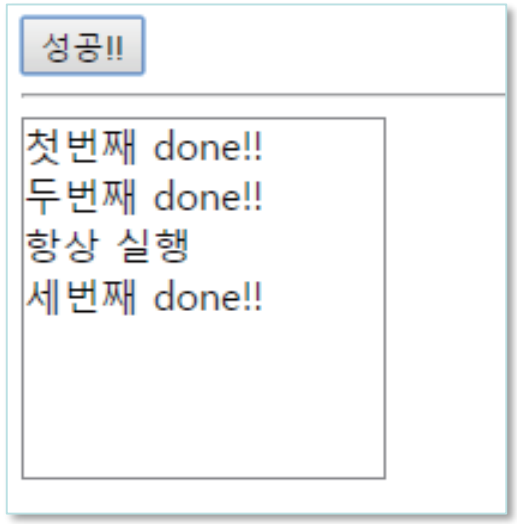
```
d.done(function() {
   $("#result").html("성공");
});
d.fail(function() {
   $("#result").html("실패");
});
d.always(function() {
   $("#result").append(", 항상 실행")
});
```

지연 객체(4)

- 다중 이벤트 콜백 등록
  - 하나의 이벤트에 여러개 함수 등록 가능
  - 예제 18-06

```
06:     var d = $.Deferred();
07:
08:     d.done(function() {
09:         $("#result").append("첫번째 done!!<br />");
10:     });
11:     d.done(function() {
12:         $("#result").append("두번째 done!!<br />");
13:     })
14:     d.always(function() {
15:         $("#result").append("항상 실행<br/ >")
16:     });
17:     d.done(function() {
18:         $("#result").append("세번째 done!!<br />");
19:     })
20:
21:     $("#success").click(function() {
22:         d.resolve();
23:     });
```

```
28:     <button id="success">성공!!</button>
29:     <hr />
30:     <div id="result"></div>
```



지연 객체(5)

- 주의할 점
  - always 이벤트에 바인딩된 함수가 마지막에 실행되지 않을 수 있다.
  - 반면 jQuery AJAX의 콜백인 success, error, complete 는 순서가 정해져 있음
  - always는 resolve(), reject() 인 경우 모두 실행됨.

지연 객체(6)

시작 함수의 등록과 아규먼트 전달

- 지연 객체를 생성할 때 시작 작업을 등록하고자 하는 경우 $.Deferred()를 호출할 때 함수를 파라미터로 전달할 수 있음
- 전달된 함수는 즉시 호출됨
- resolve(), reject() 메서드를 호출할 때 아규먼트를 전달할 수도 있음

[예제 18-07 : 지연 객체를 통한 아규먼트 전달]

```
01: <script type="text/javascript">
02: $(document).ready(function() {
03:     function beforeStart(deferred) {
04:         console.log("Before Start");
05:         setTimeout(function() {
06:             var num = Math.floor(Math.random() * 1000);
07:             var sum = 0;
08:             for (var i=0; i < num; i++) {
09:                 sum += i;
10:             }
11:             deferred.resolve({
12:                 num:num,
13:                 result:sum
14:             });
15:         }, 3000);
16:     }
17:
18:     var d =$.Deferred(beforeStart);
19:
20:     d.done(function(args) {
21:         console.log(args);
22:     });
23: });
24: </script>
```

```
Before Start                                    18-07.html:10
Object {num: 401, result: 80200}                18-07.html:27
```

지연 객체(7)

진행 단계 처리

- 긴 실행 시간이 필요한 백그라운드 작업은 실행되는 동안 사용자에게 진행상태를 계속해서 알려주는 것이 바람직함
- 지연객체의 아규먼트를 이용해 진행상태를 알려주는 기능 제공
  - notify() 함수를 호출
  - progress 이벤트에 바인딩된 함수를 통해 진행 상태를 받아 처리
- 차이를 비교하기 위해 지연객체를 사용하지 않는 코드와 사용하는 코드를 비교해보자
- 사용하지 않는 코드 : 예제 18-08
  - 아무런 문제 없이 실행되지만 몇가지 구조적 문제점을 안고 있다.
  - 화면 UI와 관련된 코드가 연산 실행코드와 섞여 있어서 가독성도 나쁘고 유지보수도 힘들다
- 사용하는 코드 : 예제 18-09

지연 객체(8)

- 예제 18-09

[예제 18-09 : 지연 객체를 이용한 진행 단계 처리]

```
01: ......
02: <script type="text/javascript">
03: $(document).ready(function() {
04:     function longTimeJob(deferred) {
05:         var num = 0;
06:         (function() {
07:             num++;
08:             deferred.notify({ p: num });
09:             if (num < 100) {
10:                 setTimeout(arguments.callee, 50);
11:             } else {
12:                 deferred.resolve({ p: num });
13:             }
14:         })();
15:     }
16:
17:     $("#long").click(function() {
18:         $("#long").attr("disabled", "disabled");
19:         var d = $.Deferred(longTimeJob);
20:
21:         d.progress(function(args) {
22:             $("#status").html(args.p);
23:         });
24:
25:         d.done(function(args) {
26:             $("#long").removeAttr("disabled");
27:             alert("작업 완료");
28:         });
29:     });
30:
31:     $("#other").click(function() {
32:         var current = new Date();
33:         $("#time").html(current.toJSON());
34:     });
35: });
36: </script>
37: ......
```

지연 객체(9)

진행상태 예제

- jQuery UI 다이얼로그, 진행바 플러그인을 조합하여 작성
- 예제 18-10 : HTML 마크업 작성

[예제 18-10 : HTML 마크업 작성]

```
01: <link rel="stylesheet"
02:     href="http://code.jquery.com/ui/1.12.0/themes/humanity/jquery-ui.css" />
03: <style>
04: body { font-size: 70%; }
05: </style>
06: <script src="http://code.jquery.com/jquery-3.1.0.js"></script>
07: <script src="http://code.jquery.com/ui/1.12.0/jquery-ui.js"></script>
08: <script type="text/javascript">
09: $(document).ready(function() {
10:
11: });
12: </script>
13: </head>
14: <body>
15:     <button id="long">긴시간 작업</button>
16:     <hr />
17:     <div style="width:400px">
18:         <div id="progressbar"></div>
19:     </div>
20:     <div id="dialog1" title="작업 완료">
21:           <p>
22:               <span class="ui-icon ui-icon-circle-check"
23:  style="float:left; margin:0 7px 20px 0;"></span>
24:               처리 작업이 완료되었습니다!
25:           </p>
26:     </div>
27: </body>
```

지연 객체(10)

- 예제 18-11 : jQuery UI 플러그인 초기화
  - value, max 값 지정 : 백분율로 진행단계 표현. 진행단계 0으로 초기화

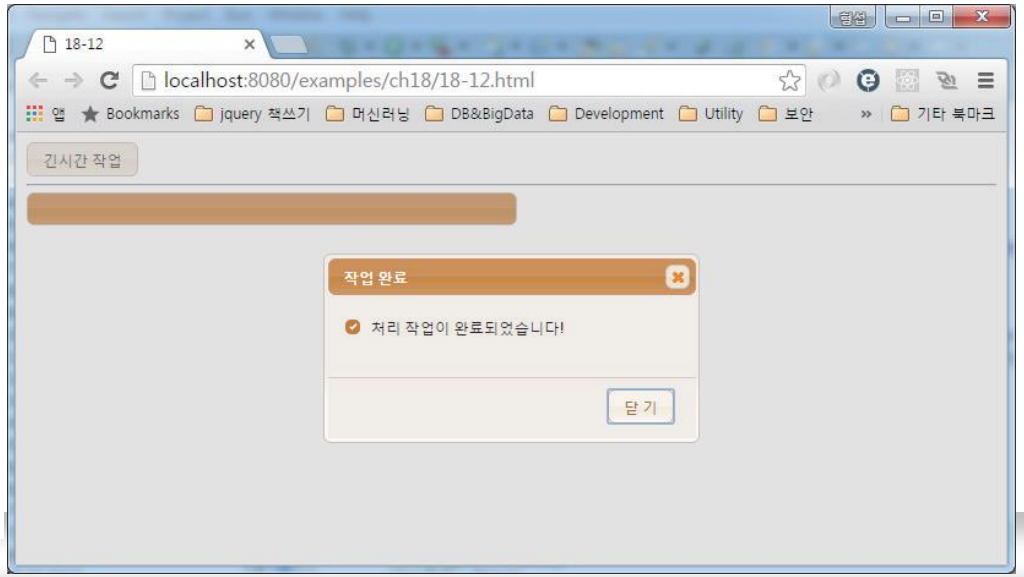
```
03:     $("#progressbar").progressbar({
04:         value : 0,
05:         max : 100
06:     });
07:
08:     $("#dialog1").dialog({
09:         autoOpen: false,
10:         modal : true,
11:         buttons : {
12:             " 닫 기 " : function() {
13:                 $(this).dialog("close");
14:                 $("#progressbar").progressbar({ value:0 });
15:             }
16:         }
17:     });
18:
19:     $("button").button();
```

지연 객체(11)

- 예제 18-12 : 지연 객체를 이용하는 코드 추가
  - 예제 18-11의 20행 위치에 코드 삽입

```
04:     function longTimeJob(deferred) {
05:         var num = 0;
06:         (function() {
07:             num++;
08:             deferred.notify({ p: num });
09:             if (num < 100) {
10:                 setTimeout(arguments.callee, 50);
11:             } else {
12:                 deferred.resolve({ p: num });
13:             }
14:         })();
15:     }
16:
17:     $("#long").click(function() {
18:         $("#long").button("disable");
19:         var d = $.Deferred(longTimeJob);
20:
21:         d.progress(function(args) {
22:             $( "#progressbar" ).progressbar({
23:                 value: args.p
24:             });
25:         });
```

```
26:
27:         d.done(function(args) {
28:             $("#long").button("enable");
29:             $("#dialog1").dialog("open");
30:         });
31:     });
```



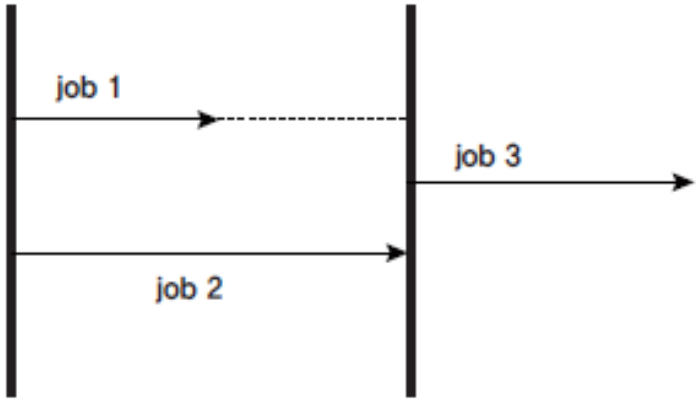
지연 객체(12)

- jQuery UI 플러그인을 활용하여 화면 UI를 동적으로 변경했지만 4~15행의 코드는 달라진 부분이 없음.
  - 18~30행의 코드만 변경되었음

지연 객체(13)

지연 객체의 결합 처리

- 예제 14-13에서 이미 다뤄본 적이 있음
  - job1, job2 가 모두 완료되면 job3가 진행되도록 함



- $.when() 메서드와 done 이벤트로 손쉽게 처리 가능

지연 객체(14)

- 예제 18-13

```
03:     function job1(deferred) {
04:         console.log("job1 start!!");
05:         setTimeout(function() {
06:             deferred.resolve({ data : "job1" });
07:         }, 1000);
08:     }
09:
10:     function job2(deferred) {
11:         console.log("job2 start!!");
12:         setTimeout(function() {
13:             deferred.resolve({ data : "job2" });
14:         }, 5000);
15:     }
```

```
16:
17:     $("#start").click(function() {
18:         var j1 = $.Deferred(job1);
19:         var j2 = $.Deferred(job2);
20:
21:         $.when(j1, j2).done(function(result1, result2) {
22:             console.log("결과1 : " + result1.data + ", timestamp: " + $.now());
23:             console.log("결과2 : " + result2.data + ", timestamp: " + $.now());
24:         });
25:     });
```

```
job1 start!!                                                    18-13.html:10
job2 start!!                                                    18-13.html:17
결과1 : job1, timestamp: 1467967116617                          18-13.html:28
결과2 : job2, timestamp: 1467967116618                          18-13.html:29
```

지연 객체(15)

- $.when() 메서드로 결합된 지연 객체를 사용하면서 개별적인 지연객체를 동시에 사용할 수 있음
- 예제 18-14

[예제 18-14]

```
j1.done(function() {
   console.log("job1 complete : " + $.now());
});
j2.done(function() {
   console.log("job2 complete : " + $.now());
});
```

```
job1 start!!                                              18-14.html:10
job2 start!!                                              18-14.html:17
job1 complete : 1467967852051                             18-14.html:28
job2 complete : 1467967856051                             18-14.html:32
결과1 : job1, timestamp: 1467967856052                    18-14.html:36
결과2 : job2, timestamp: 1467967856053                    18-14.html:37
```

지연 객체(16)

지연 객체 이벤트 통합 처리

- then() 메서드 : resolve(), reject(), notify() 에 대한 처리를 이 메서드 하나로 모두 처리할 수 있음

```
deferred.then(done[, fail] [, progress])
```

- done은 필수 전달 요소. fail, progress는 선택적으로 전달할 수 있음
- 예제 18-15
  - 예제 18-12의 17~31행을 다음과 같이 변경할 수 있음

[예제 18-15]

```
01:     $("#long").click(function() {
02:         $("#long").button("disable");
03:         var d = $.Deferred(longTimeJob);
04:
05:         d.then(
06:             function(args) {
07:                 $("#long").button("enable");
08:                 $("#dialog1").dialog("open");
09:             },
10:             $.noop(),
11:             function(args) {
12:                 $( "#progressbar" ).progressbar({
13:                     value: args.p
14:                 });
15:             }
16:         );
17:     });
```

정리

- jQuery 유틸리티 함수와 지연 객체는 자주 사용하지는 않지만 알아두면 좋은 기능을 제공한다.
  - 부족한 자바스크립트의 유틸리티 기능을 보완
  - 지연 객체는 작업의 진행 상황, 처리 결과를 비동기적으로 처리할 수 있는 유용한 기능을 제공
- 백그라운드 처리를 수행할 때 유용함
  - 최근 웹 앱이 백그라운드 처리를 수행하는 경우가 많으므로 지연객체가 많은 도움이 됨.