

JSDeferred

```
1  /* Header::
2  * JSDeferred
3  * Copyright (c) 2007 cho45 ( www.lowreal.net )
4  *
5  * http://coderepos.org/share/wiki/JSDeferred
6  *
7  * Version:: 0.2.2
8  * License:: MIT
9  *
10 * Permission is hereby granted, free of charge, to any person obtaining a copy
11 * of this software and associated documentation files (the "Software"), to deal
12 * in the Software without restriction, including without limitation the rights
13 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
14 * copies of the Software, and to permit persons to whom the Software is
15 * furnished to do so, subject to the following conditions:
16 *
17 * The above copyright notice and this permission notice shall be included in
18 * all copies or substantial portions of the Software.
19 *
20 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
21 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
22 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
23 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
24 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
25 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
26 * THE SOFTWARE.
27 */
28 /* Usage (with jQuery)::
29 *
30 * $.deferred.define();
31 *
32 * $.get("/hoge").next(function (data) {
33 *     alert(data);
34 * });
35 *
36 * parallel([$get("foo.html"), $get("bar.html")]).next(function (values) {
37 *     log($.map(values, function (v) { return v.length }));
38 *     if (values[1].match(/nextUrl:\s*(\S+)/)) {
39 *         return $.get(RegExp.$1).next(function (d) {
40 *             return d;
41 *         });
42 *     }
43 * });
44 * next(function (d) {
45 *     log(d.length);
46 * });
47 */
48
49
50
51 /* function Deferred () //=> constructor
52 *
53 * `Deferred` function is constructor of Deferred.
54 *
55 * Sample:
56 * var d = new Deferred();
57 * // or this is shothand of above.
58 * var d = Deferred();
59 */
60 /* function Deferred.prototype.next (fun) //=> Deferred
61 *
62 * Create new Deferred and sets `fun` as its callback.
63 */
64 /* function Deferred.prototype.error (fun) //=> Deferred
65 *
66 * Create new Deferred and sets `fun` as its errorback.
67 */
68 * If `fun` not throws error but returns normal value, Deferred treats
69 * the given error is recovery and continue callback chain.
70 */
71 /* function Deferred.prototype.call (val) //=> this
72 *
73 * Invokes self callback chain.
74 */
75 /* function Deferred.prototype.fail (err) //=> this
76 *
77 * Invokes self errorback chain.
78 */
79 /* function Deferred.prototype.cancel (err) //=> this
80 *
81 * Cancels self callback chain.
82 */
83 function Deferred () { return (this instanceof Deferred) ? this.init() : new Deferred() }
84 Deferred.ok = function (x) { return x };
85 Deferred.ng = function (x) { throw x };
86 Deferred.prototype = {
87     init : function () {
88         this._next = null;
89         this.callback = {
90             ok: Deferred.ok,
91             ng: Deferred.ng
92         };
93         return this;
94     },
95
96     next : function (fun) { return this._post("ok", fun) },
97     error : function (fun) { return this._post("ng", fun) },
98     call : function (val) { return this._fire("ok", val) },
99     fail : function (err) { return this._fire("ng", err) },
100
101     cancel : function () {
102         (this.canceller || function () {});
103         return this.init();
104     },
105
106     _post : function (okng, fun) {
107         this._next = new Deferred();
108         this._next.callback[okng] = fun;
109         return this._next;
110     },
111
112     _fire : function (okng, value) {
113         var next = "ok";
114         try {
115             value = this.callback[okng].call(this, value);
116         } catch (e) {
117             next = "ng";
118             value = e;
119         }
120         if (value instanceof Deferred) {
121             value._next = this._next;
122         } else {
123             if (this._next) this._next._fire(next, value);
124         }
125         return this;
126     }
127 };
128
129 /* function parallel (deferredlist) //=> Deferred
130 *
131 * `parallel` wraps up `deferredlist` to one deferred.
132 * This is useful when some asynchronous resources required.
133 *
134 * `deferredlist` can be Array or Object (Hash).
135 *
136 * Sample:
137 * parallel([
138 *     $.get("foo.html"),
139 *     $.get("bar.html")
140 * ]).next(function (values) {
141 *     values[0] //=> foo.html data
142 *     values[1] //=> bar.html data
143 * });
144 *
145 * parallel({
146 *     foo: $.get("foo.html"),
147 *     bar: $.get("bar.html")
148 * }).next(function (values) {
149 *     values.foo //=> foo.html data
150 *     values.bar //=> bar.html data
151 * });
152 */
153 Deferred.parallel = function (dl) {
154     var ret = new Deferred(), values = {}, num = 0;
155     for (var i in dl) if (dl.hasOwnProperty(i)) (function (d, i) {
156         d.next(function (v) {
157             values[i] = v;
158             if (--num <= 0) {
159                 if (dl instanceof Array) {
160                     values.length = dl.length;
161                     values = Array.prototype.slice.call(values, 0);
162                 }
163                 ret.call(values);
164             }
165         }).error(function (e) {
166             ret.fail(e);
167         });
168         num++;
169     })(dl[i], i);
170
171     if (!num) Deferred.next(function () { ret.call() });
172     ret.canceller = function () {
173         for (var i in dl) if (dl.hasOwnProperty(i)) {
174             dl[i].cancel();
175         }
176     };
177     return ret;
178 };
179
180 /* function wait (sec) //=> Deferred
181 *
182 * `wait` returns deferred that will be called after `sec` elapsed
183 * with real elapsed time (msec)
184 *
185 * Sample:
186 * wait(1).next(function (elapsed) {
187 *     log(elapsed); //=> may be 990-1100
188 * });
189 */
190 Deferred.wait = function (n) {
191     var d = new Deferred(), t = new Date();
192     var id = setTimeout(function () {
193         d.call(new Date().getTime() - t.getTime());
194     }, n * 1000);
195     d.canceller = function () { clearTimeout(id) };
196     return d;
197 };
198
199 /* function next (fun) //=> Deferred
200 *
201 * `next` is shorthand for creating new deferred which
202 * is called after current queue.
203 */
204 Deferred.next_default = function (fun) {
205     var d = new Deferred();
206     var id = setTimeout(function () { d.call() }, 0);
207     d.canceller = function () { clearTimeout(id) };
208     if (fun) d.callback.ok = fun;
209     return d;
210 };
211
212 Deferred.next_faster_way_readystatechange = (!window.opera && /\bMSIE\b/.test(navigator.userAgent)) && function (fun) {
213     // MSIE
214     var d = new Deferred();
215     var t = new Date().getTime();
216     if (t - arguments.callee._prev_timeout_called < 150) {
217         var cancel = false;
218         var script = document.createElement("script");
219         script.type = "text/javascript";
220         script.src = "javascript:.";
221         script.onreadystatechange = function () {
222             if (!cancel) {
223                 d.canceller();
224                 d.call();
225             }
226         };
227         d.canceller = function () {
228             if (!cancel) {
229                 cancel = true;
230                 script.onreadystatechange = null;
231                 document.body.removeChild(script);
232             }
233         };
234     } else {
235         arguments.callee._prev_timeout_called = t;
236         var id = setTimeout(function () { d.call() }, 0);
237         d.canceller = function () { clearTimeout(id) };
238     }
239     if (fun) d.callback.ok = fun;
240     return d;
241 };
242
243 Deferred.next_faster_way_Image = ((typeof Image) != "undefined") && document.addEventListener && function (fun) {
244     // Modern Browsers
245     var d = new Deferred();
246     var img = new Image();
247     var handler = function () {
248         d.canceller();
249         d.call();
250     };
251     img.addEventListener("load", handler, false);
252     img.addEventListener("error", handler, false);
253     d.canceller = function () {
254         img.removeEventListener("load", handler, false);
255         img.removeEventListener("error", handler, false);
256     };
257     img.src = "data: / _ / X";
258     if (fun) d.callback.ok = fun;
259     return d;
260 };
261
262 Deferred.next = Deferred.next_faster_way_readystatechange ||
263     Deferred.next_faster_way_Image ||
264     Deferred.next_default;
265
266 /* function call (fun [, args...]) //=> Deferred
267 *
268 * `call` function is for calling function asynchronous.
269 *
270 * Sample:
271 * next(function () {
272 *     function pow (x, n) {
273 *         function _pow (n, r) {
274 *             print([n, r]);
275 *             if (n == 0) return r;
276 *             return call(_pow, n - 1, x * r);
277 *         }
278 *         return call(_pow, n, 1);
279 *     });
280 *     return call(pow, 2, 10);
281 * });
282 *
283 * next(function (r) {
284 *     print([r, "end"]);
285 * });
286 */
287
288 Deferred.call = function (f /*, args... */) {
289     var args = Array.prototype.slice.call(arguments, 1);
290     return Deferred.next(function () {
291         return f.apply(this, args);
292     });
293 };
294
295 /* function loop (n, fun) //=> Deferred
296 *
297 * `loop` function provides browser-non-blocking loop.
298 * This loop is slow but not stop browser's appearance.
299 *
300 * Sample:
301 * //=> loop 1 to 100
302 * loop({begin:1, end:100, step:10}, function (n, o) {
303 *     for (var i = 0; i < o.step; i++) {
304 *         log(n+i);
305 *     }
306 * });
307 *
308 * //=> loop 10 times
309 * loop(10, function (n) {
310 *     log(n);
311 * });
312 */
313
314 Deferred.loop = function (n, fun) {
315     var o = {
316         begin : n.begin || 0,
317         end : (typeof n.end == "number") ? n.end : n - 1,
318         step : n.step || 1,
319         last : false,
320     };
321     prev : null
322 };
323
324 var ret, step = o.step;
325 return Deferred.next(function () {
326     function _loop (i) {
327         if (i <= o.end) {
328             if ((i + step) > o.end) {
329                 o.last = true;
330                 o.step = o.end - i + 1;
331             }
332             o.prev = ret;
333             ret = fun.call(this, i, o);
334             if (ret instanceof Deferred) {
335                 return ret.next(function (r) {
336                     ret = r;
337                     return Deferred.call(_loop, i + step);
338                 });
339             } else {
340                 return Deferred.call(_loop, i + step);
341             }
342         } else {
343             return ret;
344         }
345     }
346     return (o.begin <= o.end) ? Deferred.call(_loop, o.begin) : null;
347 });
348
349 /* function Deferred.register (name, fun) //=> void 0
350 *
351 * Register `fun` to Deferred prototype for method chain.
352 *
353 * Sample:
354 * // Deferred.register("loop", loop);
355 *
356 * // Global Deferred function
357 * loop(10, function (n) {
358 *     print(n);
359 * });
360 *
361 * // Registered Deferred.prototype.loop
362 * loop(10, function (n) {
363 *     print(n);
364 * });
365 */
366
367 Deferred.register = function (name, fun) {
368     this.prototype[name] = function () {
369         return this.next(Deferred.wrap(fun).apply(null, arguments));
370     };
371 };
372
373 /* function Deferred.wrap (dfun) //=> Function
374 *
375 * Create and return function which run `dfun` with arguments.
376 *
377 * Sample:
378 * var dloop = Deferred.wrap(loop);
379 *
380 * next(dloop(10, function (n) {
381 *     print(n);
382 * }));
383 */
384
385 Deferred.wrap = function (dfun) {
386     return function () {
387         var a = arguments;
388         return dfun.apply(null, a);
389     };
390 };
391
392 Deferred.register("loop", Deferred.loop);
393 Deferred.register("wait", Deferred.wait);
394
395 Deferred.define = function (obj, list) {
396     if (!list) list = ["parallel", "wait", "next", "call", "loop"];
397     if (!obj) obj = (function getGlobal () { return this })();
398     for (var i = 0; i < list.length; i++) {
399         var n = list[i];
400         obj[n] = Deferred[n];
401     }
402     return Deferred;
403 };
404
405 }
```