```
 1 !(function (G, ImportModule) {
 2
 3     const functionMap = {};
 4     //==========================================================================
 5     // isPlainObject
 6     const isPlainObject = function (obj) {
 7         debugger;
 8
 9         if (typeof obj != "object") {
10             return false;
11         }
12
13         if (obj == null) {
14             return false;
15         }
16
17         let res = Object.prototype.toString.call(obj);
18
19         if (!/^\[object Object\]$/.test(res)) {
20             return false;
21         }
22
23         if (obj.constructor !== {}.constructor) {
24             return false;
25         }
26
27         return true;
28     };
29     functionMap['isPlainObject'] = isPlainObject;
30     //==========================================================================
31     // getClassName
32
33     functionMap['getClassName'] = function (data) {
34         let _toString = Object.prototype.toString;
35
36         let type = typeof (data);
37
38         if (/object/.test(type)) {
39
40             if (data === null) {
41                 type = "null";
42             } else {
43                 type = _toString.call(data);
44
45                 let res = /\[\w+\s+(\w+)\]/.exec(type);
46                 if (res && res[1]) {
47                     type = res[1];
48                 }
49             }
50         }
51         return type;
52     };
53
54     //==========================================================================
55     // ptimeout
56     //
57     // job: [function|promise]
58     functionMap['ptimeout'] = function (job, timeLimit, context) {
59         debugger;
60
61         let msg;
62
```

```
63          let p2;
64
65          if (typeof timeLimit != 'number') {
66              throw new TypeError("timeout arg[1] must be number");
67          }
68
69          if (typeof (job) == "function") {
70
71              if (context != null) {
72                  job = job.bind(context);
73              }
74              p2 = new Promise(job);
75          } else if (job instanceof Promise) {
76              p2 = job;
77          } else {
78              throw new TypeError("timeout arg[0] must be promise or function");
79          }
80          //-----------------------
81          let _res;
82          let _rej;
83
84          let p1 = new Promise(function (res, rej) {
85              debugger;
86              _res = res;
87              _rej = rej;
88          });
89          //-----------------------
90          p2.then(function (data) {
91              debugger;
92              _res(data);
93          }, function (err) {
94              _rej(err);
95          });
96
97          setTimeout(function () {
98              _rej(new Error('timeout'));
99          }, timeLimit);
100         //-----------------------
101
102         return p1;
103     };
104
105     //=============================================================================
106     // promise
107     //
108     // callback: [function(返回 promise)|promise[]]
109     // context: 背後執行對象
110     functionMap['promise'] = function (callback, context) {
111         let p;
112
113         if (callback instanceof Promise) {
114             p = Promise.resolve(callback);
115         } else if (typeof (callback) == "function") {
116             callback = (context == null) ? callback : callback.bind(context);
117
118             p = new Promise(callback);
119         } else if (Array.isArray(callback)) {
120
121             if (context != null) {
122                 callback = callback.map(function (fn) {
123                     return fn.bind(context);
124                 });
```

```
            }

            p = Promise.all(callback);
        } else {
            p = Promise.resolve(callback);
        }
        //-----------------------
        if (p['$status'] == null) {
            Object.defineProperty(p, '$status', {
                value: 0,
                enumerable: false,
                writable: true,
                configurable: true
            });
        }

        p.then(function () {
            p['$status'] = 1;
        }, function (err) {
            p['$status'] = 2;
            err = (err instanceof Error) ? err : new Error(err);
            throw err;
        });

        return p;
    };
    //==========================================================================
    // deferred
    functionMap['deferred'] = (function () {

        (function () {
            // 對系統的 promise 擴增 API
            if (typeof Promise.prototype.thenWith == 'undefined') {
                Promise.prototype.thenWith = function (onFulfilled, onRejected, context) {

                    onFulfilled = onFulfilled.bind(context);
                    onRejected = onRejected.bind(context);

                    return this.then(onFulfilled, onRejected);
                };
            }
            //-----------------------------------------------------------------
            // promise.catchWith()
            if (typeof Promise.prototype.catchWith == 'undefined') {
                Promise.prototype.catchWith = function (onRejected, context) {

                    onRejected = onRejected.bind(context);

                    return this.then(null, onRejected);
                };
            }
            //-----------------------------------------------------------------
            // promise.always()
            if (typeof Promise.prototype.always == 'undefined') {
                Promise.prototype.always = function (callback) {

                    return this.then(function (data) {
                        callback(false, data);
                    }, function (error) {
                        callback(true, error);
                    });
                };
```

```
187             }
188             //---------------------------------------------------------------------
189             // promise.alwaysWith()
190             if (typeof Promise.prototype.alwaysWith === 'undefined') {
191                 Promise.prototype.alwaysWith = function (callback, context) {
192 
193                     callback = callback.bind(context);
194 
195                     return this.then(function (data) {
196                         callback(false, data);
197                     }, function (error) {
198                         callback(true, error);
199                     });
200                 };
201             }
202         })();
203 
204         const Deferred = (function () {
205             // 模組範圍
206 
207             class Deferred {
208 
209                 constructor() {
210                     this.fn = Deferred;
211                     this._reject;
212                     this._resolve;
213                     this._promise;
214 
215                     this._init();
216                 }
217 
218                 get allStatusList() {
219                     return ['pending', 'fulfilled', 'rejected'];
220                 }
221 
222                 _init = function () {
223                     let $this = this;
224 
225                     this._promise = _.promise(function (resolve, reject) {
226                         this._resolve = resolve;
227                         this._reject = reject;
228                     }, this);
229 
230                     this._setStatus(0);
231 
232                     this._setStatusGet();
233 
234                     this._promise.then(function (data) {
235                         $this._setStatus(1);
236                         return data;
237                     }, function (err) {
238                         $this._setStatus(2);
239                     });
240                 }
241 
242                 _setStatusGet() {
243                     // 防止修改 this.status
244 
245                     let target = this._promise;
246 
247                     Object.defineProperty(this, 'status', {
248                         enumerable: true,
```

```
249                         configurable: true,
250                         get: function () {
251                             return target['$status'];
252                         },
253                         set: function () {
254                             return;
255                         }
256                     });
257                 }
258 
259                 promise() {
260                     return this._promise;
261                 };
262                 //-------------------------------------------------------------------
263                 resolve(arg) {
264                     this._resolve(arg);
265                 };
266                 //-------------------------------------------------------------------
267                 reject(err) {
268                     this._reject(err);
269                 };
270                 //-------------------------------------------------------------------
271                 then = function (onFulfilled, onRejected) {
272                     var def = Deferred();
273                     var p = this.promise();
274 
275                     p = p.then(this._getCallback(onFulfilled),
276                         this._getErrorCallback(onRejected));
277                     //-----------------------
278                     p.then(function (data) {
279                         def.resolve(data);
280                     }, function (error) {
281                         def.reject(error);
282                     });
283                     return def;
284                 };
285                 //-------------------------------------------------------------------
286                 thenWith = function (onFulfilled, onRejected, context) {
287                     var def = Deferred();
288                     var p = this.promise();
289 
290                     p = p.then(this._getCallback(onFulfilled, context),
291                         this._getErrorCallback(onRejected, context));
292                     //-----------------------
293                     p.then(function (data) {
294                         def.resolve(data);
295                     }, function (error) {
296                         def.reject(error);
297                     });
298                     return def;
299                 };
300                 //-------------------------------------------------------------------
301                 catch = function (onRejected) {
302                     var def = Deferred();
303                     var p = this.promise();
304 
305                     p = p.catch(this._getErrorCallback(onRejected));
306                     //-----------------------
307                     p.then(function (data) {
308                         def.resolve(data);
309                     }, function (error) {
310                         def.reject(error);
```

```
311                     });
312                     return def;
313                 };
314                 //-------------------------------------------------------------------
315                 catchWith = function (onRejected, context) {
316                     var def = Deferred();
317                     var p = this.promise();
318 
319                     p = p.catch(this._getErrorCallback(onRejected, context));
320                     //-----------------------
321                     p.then(function (data) {
322                         def.resolve(data);
323                     }, function (error) {
324                         def.reject(error);
325                     });
326                     return def;
327                 };
328                 //-------------------------------------------------------------------
329                 always = function (callback) {
330                     var def = Deferred();
331                     var p = this.promise();
332 
333                     p = p.then(this._getAlwaysCallback(callback, false),
334                         this._getAlwaysCallback(callback, true));
335                     //-----------------------
336                     p.then(function (data) {
337                         def.resolve(data);
338                     }, function (error) {
339                         def.reject(error);
340                     });
341                     return def;
342                 };
343                 //-------------------------------------------------------------------
344                 alwaysWith = function (callback, context) {
345                     callback = callback.binf(context);
346 
347                     var def = Deferred();
348                     var p = this.promise();
349 
350                     p = p.then(this._getAlwaysCallback(callback, false, context),
351                         this._getAlwaysCallback(callback, true, context));
352                     /*-------------------------*/
353                     p.then(function (data) {
354                         def.resolve(data);
355                     }, function (error) {
356                         def.reject(error);
357                     });
358                     return def;
359                 };
360                 //-------------------------------------------------------------------
361                 isPending = function () {
362                     return (this._promise['$status'] == 0);
363                 };
364                 //-------------------------------------------------------------------
365                 isFulfilled = function () {
366                     return (this._promise['$status'] == 1);
367                 };
368                 //-------------------------------------------------------------------
369                 isRejected = function () {
370                     return (this._promise['$status'] == 2);
371                 };
372                 //-------------------------------------------------------------------
```

```
373                 _setStatus = function (status) {
374                     this._promise['$status'] = status;
375                 };
376                 //--------------------------------------------------------------------
377                 _getCallback = function (callback, context) {
378                     if (callback == null) {
379                         return null;
380                     }
381 
382                     callback = (context === undefined ? callback : callback.bind(context));
383 
384                     return function (d) {
385                         return callback(d);
386                     };
387                 };
388                 //--------------------------------------------------------------------
389                 _getErrorCallback = function (callback, context) {
390                     if (callback == null) {
391                         return null;
392                     }
393 
394                     callback = (context === undefined ? callback : callback.bind(context));
395 
396                     return function (err) {
397                         return callback(err);
398                     };
399                 };
400                 //--------------------------------------------------------------------
401                 _getAlwaysCallback = function (callback, args, context) {
402                     if (callback == null) {
403                         return null;
404                     }
405 
406                     callback = (context === undefined ? callback : callback.bind(context));
407 
408                     return function (d) {
409                         return callback(args, d);
410                     };
411                 }
412             }
413 
414             return Deferred;
415         })();
416 
417 
418         return function () {
419             return new Deferred();
420         };
421     })();
422     //================================================================================
423     if (ImportModule) {
424         // 注入功能
425         ImportModule(importFactory);
426     } else {
427         // nodejs 的引入窗口
428         module.exports = function (ImportModule) {
429             ImportModule(importFactory);
430         };
431     }
432     //----------------------------
433     // 對外曝露工廠
434     function importFactory(_) {
```

```
435         // 當前環境
436 
437         if(typeof _.$extension1 == "undefined"){
438             throw new Error("no import _");
439         }
440 
441         const _extension1 = _.$extension1;
442 
443         const environment = _extension1.info.environment;
444 
445         for (let funKey in functionMap) {
446             let m = functionMap[funKey];
447 
448             if (Array.isArray(m.unsupportEnvironment) &&
449 
450                 // 確定各函式能執行的環境
451                 m.unsupportEnvironment.includes(environment)) {
452                 delete functionMap[funKey];
453             }
454 
455             if (funKey in _) {
456                 // 避免衝突
457                 delete functionMap[funKey];
458             }
459         }
460         _.mixin(functionMap);
461     }
462 
463 })(this, (typeof _$ImportModules != "undefined" ? _$ImportModules : null));
```