```
!(function (G, ImportModule) {

    const functionMap = {};
    //========================================================================
    // isPlainObject
    const isPlainObject = function (obj) {
        debugger;

        if (typeof obj != "object") {
            return false;
        }

        if (obj == null) {
            return false;
        }

        let res = Object.prototype.toString.call(obj);

        if (!/^\[object Object\]$/.test(res)) {
            return false;
        }

        if (obj.constructor !== {}.constructor) {
            return false;
        }

        return true;
    };
    functionMap['isPlainObject'] = isPlainObject;
    //========================================================================
    // getClassName

    functionMap['getClassName'] = function (data) {
        let _toString = Object.prototype.toString;

        let type = typeof (data);

        if (/object/.test(type)) {

            if (data === null) {
                type = "null";
            } else {
                type = _toString.call(data);

                let res = /\[\w+\s+(\w+)\]/.exec(type);
                if (res && res[1]) {
                    type = res[1];
                }
            }
        }
        return type;
    };

    //========================================================================
    // ptimeout
    //
    // job: [function|promise]
    functionMap['ptimeout'] = function (job, timeLimit, context) {
        debugger;

        let msg;

        let p2;

        if (typeof timeLimit != 'number') {
            throw new TypeError("timeout arg[1] must be number");
        }

        if (typeof (job) == "function") {

            if (context != null) {
                job = job.bind(context);
            }
            p2 = new Promise(job);
        } else if (job instanceof Promise) {
            p2 = job;
        } else {
            throw new TypeError("timeout arg[0] must be promise or function");
        }
        //----------------------
        let _res;
        let _rej;

        let p1 = new Promise(function (res, rej) {
            debugger;
            _res = res;
            _rej = rej;
        });
        //----------------------
        p2.then(function (data) {
            debugger;
            _res(data);
        }, function (err) {
            _rej(err);
        });

        setTimeout(function () {
            _rej(new Error('timeout'));
        }, timeLimit);
        //----------------------

        return p1;
    };

    //==========================================================================
    // promise
    //
    // callback: [function(返回 promise)|promise[]]
    // context: 背後執行對象
    functionMap['promise'] = function (callback, context) {
        let p;

        if (callback instanceof Promise) {
            p = Promise.resolve(callback);
        } else if (typeof (callback) == "function") {
            callback = (context == null) ? callback : callback.bind(context);

            p = new Promise(callback);
        } else if (Array.isArray(callback)) {

            if (context != null) {
                callback = callback.map(function (fn) {
                    return fn.bind(context);
                });
            }

            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);
                    });
                };
            }
            //----------------------------------------------------------------------
            // promise.alwaysWith()
            if (typeof Promise.prototype.alwaysWith === 'undefined') {
                Promise.prototype.alwaysWith = function (callback, context) {

                    callback = callback.bind(context);

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

        const Deferred = (function () {
            // 模組範圍

            class Deferred {

                constructor() {
                    this.fn = Deferred;
                    this._reject;
                    this._resolve;
                    this._promise;

                    this._init();
                }

                get allStatusList() {
                    return ['pending', 'fulfilled', 'rejected'];
                }

                _init = function () {
                    let $this = this;

                    this._promise = _.promise(function (resolve, reject) {
                        this._resolve = resolve;
                        this._reject = reject;
                    }, this);

                    this._setStatus(0);

                    this._setStatusGet();

                    this._promise.then(function (data) {
                        $this._setStatus(1);
                        return data;
                    }, function (err) {
                        $this._setStatus(2);
                    });
                }

                _setStatusGet() {
                    // 防止修改 this.status

                    let target = this._promise;

                    Object.defineProperty(this, 'status', {
                        enumerable: true,
                        configurable: true,
                        get: function () {
                            return target['$status'];
                        },
                        set: function () {
                            return;
                        }
                    });
                }

                promise() {
                    return this._promise;
                };
                //------------------------------------------------------------------
                resolve(arg) {
                    this._resolve(arg);
                };
                //------------------------------------------------------------------
                reject(err) {
                    this._reject(err);
                };
                //------------------------------------------------------------------
                then = function (onFulfilled, onRejected) {
                    var def = Deferred();
                    var p = this.promise();

                    p = p.then(this._getCallback(onFulfilled),
                        this._getErrorCallback(onRejected));
                    //----------------------
                    p.then(function (data) {
                        def.resolve(data);
                    }, function (error) {
                        def.reject(error);
                    });
                    return def;
                };
                //------------------------------------------------------------------
                thenWith = function (onFulfilled, onRejected, context) {
                    var def = Deferred();
                    var p = this.promise();

                    p = p.then(this._getCallback(onFulfilled, context),
                        this._getErrorCallback(onRejected, context));
                    //----------------------
                    p.then(function (data) {
                        def.resolve(data);
                    }, function (error) {
                        def.reject(error);
                    });
                    return def;
                };
                //------------------------------------------------------------------
                catch = function (onRejected) {
                    var def = Deferred();
                    var p = this.promise();

                    p = p.catch(this._getErrorCallback(onRejected));
                    //----------------------
                    p.then(function (data) {
                        def.resolve(data);
                    }, function (error) {
                        def.reject(error);
                    });
                    return def;
                };
                //-------------------------------------------------------------------
                catchWith = function (onRejected, context) {
                    var def = Deferred();
                    var p = this.promise();

                    p = p.catch(this._getErrorCallback(onRejected, context));
                    //----------------------
                    p.then(function (data) {
                        def.resolve(data);
                    }, function (error) {
                        def.reject(error);
                    });
                    return def;
                };
                //-------------------------------------------------------------------
                always = function (callback) {
                    var def = Deferred();
                    var p = this.promise();

                    p = p.then(this._getAlwaysCallback(callback, false),
                        this._getAlwaysCallback(callback, true));
                    //----------------------
                    p.then(function (data) {
                        def.resolve(data);
                    }, function (error) {
                        def.reject(error);
                    });
                    return def;
                };
                //-------------------------------------------------------------------
                alwaysWith = function (callback, context) {
                    callback = callback.binf(context);

                    var def = Deferred();
                    var p = this.promise();

                    p = p.then(this._getAlwaysCallback(callback, false, context),
                        this._getAlwaysCallback(callback, true, context));
                    /*------------------------*/
                    p.then(function (data) {
                        def.resolve(data);
                    }, function (error) {
                        def.reject(error);
                    });
                    return def;
                };
                //-------------------------------------------------------------------
                isPending = function () {
                    return (this._promise['$status'] == 0);
                };
                //-------------------------------------------------------------------
                isFulfilled = function () {
                    return (this._promise['$status'] == 1);
                };
                //-------------------------------------------------------------------
                isRejected = function () {
                    return (this._promise['$status'] == 2);
                };
                //-------------------------------------------------------------------
                _setStatus = function (status) {
                    this._promise['$status'] = status;
                };
                //----------------------------------------------------------------
                _getCallback = function (callback, context) {
                    if (callback == null) {
                        return null;
                    }

                    callback = (context === undefined ? callback : callback.bind(context));

                    return function (d) {
                        return callback(d);
                    };
                };
                //----------------------------------------------------------------
                _getErrorCallback = function (callback, context) {
                    if (callback == null) {
                        return null;
                    }

                    callback = (context === undefined ? callback : callback.bind(context));

                    return function (err) {
                        return callback(err);
                    };
                };
                //----------------------------------------------------------------
                _getAlwaysCallback = function (callback, args, context) {
                    if (callback == null) {
                        return null;
                    }

                    callback = (context === undefined ? callback : callback.bind(context));

                    return function (d) {
                        return callback(args, d);
                    };
                }
            }

            return Deferred;
        })();


        return function () {
            return new Deferred();
        };
    })();
    //======================================================================
    if (ImportModule) {
        // 注入功能
        ImportModule(importFactory);
    } else {
        // nodejs 的引入窗口
        module.exports = function (ImportModule) {
            ImportModule(importFactory);
        };
    }
    //---------------------------
    // 對外曝露工廠
    function importFactory(_) {
        // 當前環境

        if(typeof _.$extension1 == "undefined"){
            throw new Error("no import _");
        }

        const _extension1 = _.$extension1;

        const environment = _extension1.info.environment;

        for (let funKey in functionMap) {
            let m = functionMap[funKey];

            if (Array.isArray(m.unsupportEnvironment) &&

                // 確定各函式能執行的環境
                m.unsupportEnvironment.includes(environment)) {
                delete functionMap[funKey];
            }

            if (funKey in _) {
                // 避免衝突
                delete functionMap[funKey];
            }
        }
        _.mixin(functionMap);
    }

})(this, (typeof _$ImportModules != "undefined" ? _$ImportModules : null));
```