// Copyright Joyent, Inc. and other Node contributors.

//

// Permission is hereby granted, free of charge, to any person obtaining a

// copy of this software and associated documentation files (the

// "Software"), to deal in the Software without restriction, including

// without limitation the rights to use, copy, modify, merge, publish,

// distribute, sublicense, and/or sell copies of the Software, and to permit

// persons to whom the Software is furnished to do so, subject to the

// following conditions:

//

// The above copyright notice and this permission notice shall be included

// in all copies or substantial portions of the Software.

//

// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS

// OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

// MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN

// NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,

// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR

// OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE

// USE OR OTHER DEALINGS IN THE SOFTWARE.

/\*\*

杩佺Щ鑷虫祻瑙堝櫒绔娇鐢�

\*\*/

//var isArray = Array.isArray;

var isArray = Array.isArray || function(obj){

return Object.prototype.toString.call(obj) === '[object Array]' ? true : false;

};

function EventEmitter(){}

// By default EventEmitters will print a warning if more than

// 10 listeners are added to it. This is a useful default which

// helps finding memory leaks.

//

// Obviously not all Emitters should be limited to 10. This function allows

// that to be increased. Set to zero for unlimited.

var defaultMaxListeners = 20;

EventEmitter.prototype.setMaxListeners = function(n) {

if (!this.\_events) this.\_events = {};

this.\_maxListeners = n;

};

EventEmitter.prototype.emit = function() {

var type = arguments[0];

// If there is no 'error' event listener then throw.

if (type === 'error') {

if (!this.\_events || !this.\_events.error ||

(isArray(this.\_events.error) && !this.\_events.error.length))

{

if (arguments[1] instanceof Error) {

throw arguments[1]; // Unhandled 'error' event

} else {

throw new Error("Uncaught, unspecified 'error' event.");

}

return false;

}

}

if (!this.\_events) return false;

var handler = this.\_events[type];

if (!handler) return false;

if (typeof handler == 'function') {

switch (arguments.length) {

// fast cases

case 1:

handler.call(this);

break;

case 2:

handler.call(this, arguments[1]);

break;

case 3:

handler.call(this, arguments[1], arguments[2]);

break;

// slower

default:

var l = arguments.length;

var args = new Array(l - 1);

for (var i = 1; i < l; i++) args[i - 1] = arguments[i];

handler.apply(this, args);

}

return true;

} else if (isArray(handler)) {

var l = arguments.length;

var args = new Array(l - 1);

for (var i = 1; i < l; i++) args[i - 1] = arguments[i];

var listeners = handler.slice();

for (var i = 0, l = listeners.length; i < l; i++) {

listeners[i].apply(this, args);

}

return true;

} else {

return false;

}

};

EventEmitter.prototype.addListener = function(type, listener) {

if ('function' !== typeof listener) {

throw new Error('addListener only takes instances of Function');

}

if (!this.\_events) this.\_events = {};

// To avoid recursion in the case that type == "newListeners"! Before

// adding it to the listeners, first emit "newListeners".

this.emit('newListener', type, listener);

if (!this.\_events[type]) {

// Optimize the case of one listener. Don't need the extra array object.

this.\_events[type] = listener;

} else if (isArray(this.\_events[type])) {

// If we've already got an array, just append.

this.\_events[type].push(listener);

} else {

// Adding the second element, need to change to array.

this.\_events[type] = [this.\_events[type], listener];

}

// Check for listener leak

if (isArray(this.\_events[type]) && !this.\_events[type].warned) {

var m;

if (this.\_maxListeners !== undefined) {

m = this.\_maxListeners;

} else {

m = defaultMaxListeners;

}

if (m && m > 0 && this.\_events[type].length > m) {

this.\_events[type].warned = true;

console.error('Possible mem-leak detected. event[' + type +'] %d listeners added. ' +

'Use emitter.setMaxListeners() to increase limit.',

this.\_events[type].length);

console.trace();

}

}

return this;

};

EventEmitter.prototype.on = EventEmitter.prototype.addListener;

EventEmitter.prototype.once = function(type, listener) {

if ('function' !== typeof listener) {

throw new Error('.once only takes instances of Function');

}

var self = this;

function g() {

self.removeListener(type, g);

listener.apply(this, arguments);

};

g.listener = listener;

self.on(type, g);

return this;

};

EventEmitter.prototype.removeListener = function(type, listener) {

if ('function' !== typeof listener) {

throw new Error('removeListener only takes instances of Function');

}

// does not use listeners(), so no side effect of creating \_events[type]

if (!this.\_events || !this.\_events[type]) return this;

var list = this.\_events[type];

if (isArray(list)) {

var position = -1;

for (var i = 0, length = list.length; i < length; i++) {

if (list[i] === listener ||

(list[i].listener && list[i].listener === listener))

{

position = i;

break;

}

}

if (position < 0) return this;

list.splice(position, 1);

if (list.length == 0)

delete this.\_events[type];

} else if (list === listener ||

(list.listener && list.listener === listener))

{

delete this.\_events[type];

}

return this;

};

EventEmitter.prototype.removeAllListeners = function(type) {

if (arguments.length === 0) {

this.\_events = {};

return this;

}

// does not use listeners(), so no side effect of creating \_events[type]

if (type && this.\_events && this.\_events[type]) this.\_events[type] = null;

return this;

};

EventEmitter.prototype.listeners = function(type) {

if (!this.\_events) this.\_events = {};

if (!this.\_events[type]) this.\_events[type] = [];

if (!isArray(this.\_events[type])) {

this.\_events[type] = [this.\_events[type]];

}

return this.\_events[type];

};