**WebUploader API文档**

Web Uploader内部类的详细说明，以下提及的功能类，都可以在WebUploader这个变量中访问到。

As you know, Web Uploader的每个文件都是用过[AMD](https://github.com/amdjs/amdjs-api/wiki/AMD)规范中的define组织起来的, 每个Module都会有个module id. 默认module id为该文件的路径，而此路径将会转化成名字空间存放在WebUploader中。如：

* module base：WebUploader.Base
* module file: WebUploader.File
* module lib/dnd: WebUploader.Lib.Dnd
* module runtime/html5/dnd: WebUploader.Runtime.Html5.Dnd

以下文档中对类的使用可能省略掉了WebUploader前缀。

**Uploader**

* new Uploader( opts ) ⇒ Uploader

上传入口类。

var uploader = WebUploader.Uploader({

swf: 'path\_of\_swf/Uploader.swf',

// 开起分片上传。

chunked: true

});

**参数说明**

* dnd {Selector} [可选] [默认值：undefined]

指定Drag And Drop拖拽的容器，如果不指定，则不启动。

* disableGlobalDnd {Selector} [可选] [默认值：false]

是否禁掉整个页面的拖拽功能，如果不禁用，图片拖进来的时候会默认被浏览器打开。

* paste {Selector} [可选] [默认值：undefined]

指定监听paste事件的容器，如果不指定，不启用此功能。此功能为通过粘贴来添加截屏的图片。建议设置为document.body.

* pick {Selector, Object} [可选] [默认值：undefined]

指定选择文件的按钮容器，不指定则不创建按钮。

* + id {Seletor|dom} 指定选择文件的按钮容器，不指定则不创建按钮。**注意** 这里虽然写的是 id, 但是不是只支持 id, 还支持 class, 或者 dom 节点。
  + label {String} 请采用 innerHTML 代替
  + innerHTML {String} 指定按钮文字。不指定时优先从指定的容器中看是否自带文字。
  + multiple {Boolean} 是否开起同时选择多个文件能力。
* accept {Arroy} [可选] [默认值：null]

指定接受哪些类型的文件。 由于目前还有ext转mimeType表，所以这里需要分开指定。

* + title {String} 文字描述
  + extensions {String} 允许的文件后缀，不带点，多个用逗号分割。
  + mimeTypes {String} 多个用逗号分割。

如：

{

title: 'Images',

extensions: 'gif,jpg,jpeg,bmp,png',

mimeTypes: 'image/\*'

}

* thumb {Object} [可选]

配置生成缩略图的选项。

默认为：

{

width: 110,

height: 110,

// 图片质量，只有type为`image/jpeg`的时候才有效。

quality: 70,

// 是否允许放大，如果想要生成小图的时候不失真，此选项应该设置为false.

allowMagnify: true,

// 是否允许裁剪。

crop: true,

// 为空的话则保留原有图片格式。

// 否则强制转换成指定的类型。

type: 'image/jpeg'

}

* compress {Object} [可选]

配置压缩的图片的选项。如果此选项为false, 则图片在上传前不进行压缩。

默认为：

{

width: 1600,

height: 1600,

// 图片质量，只有type为`image/jpeg`的时候才有效。

quality: 90,

// 是否允许放大，如果想要生成小图的时候不失真，此选项应该设置为false.

allowMagnify: false,

// 是否允许裁剪。

crop: false,

// 是否保留头部meta信息。

preserveHeaders: true,

// 如果发现压缩后文件大小比原来还大，则使用原来图片

// 此属性可能会影响图片自动纠正功能

noCompressIfLarger: false,

// 单位字节，如果图片大小小于此值，不会采用压缩。

compressSize: 0

}

* auto {Boolean} [可选] [默认值：false]

设置为 true 后，不需要手动调用上传，有文件选择即开始上传。

* runtimeOrder {Object} [可选] [默认值：html5,flash]

指定运行时启动顺序。默认会想尝试 html5 是否支持，如果支持则使用 html5, 否则则使用 flash.

可以将此值设置成 flash，来强制使用 flash 运行时。

* prepareNextFile {Boolean} [可选] [默认值：false]

是否允许在文件传输时提前把下一个文件准备好。 对于一个文件的准备工作比较耗时，比如图片压缩，md5序列化。 如果能提前在当前文件传输期处理，可以节省总体耗时。

* chunked {Boolean} [可选] [默认值：false]

是否要分片处理大文件上传。

* chunkSize {Boolean} [可选] [默认值：5242880]

如果要分片，分多大一片？ 默认大小为5M.

* chunkRetry {Boolean} [可选] [默认值：2]

如果某个分片由于网络问题出错，允许自动重传多少次？

* threads {Boolean} [可选] [默认值：3]

上传并发数。允许同时最大上传进程数。

* formData {Object} [可选] [默认值：{}]

文件上传请求的参数表，每次发送都会发送此对象中的参数。

* fileVal {Object} [可选] [默认值：'file']

设置文件上传域的name。

* method {Object} [可选] [默认值：'POST']

文件上传方式，POST或者GET。

* sendAsBinary {Object} [可选] [默认值：false]

是否已二进制的流的方式发送文件，这样整个上传内容php://input都为文件内容， 其他参数在$\_GET数组中。

* fileNumLimit {int} [可选] [默认值：undefined]

验证文件总数量, 超出则不允许加入队列。

* fileSizeLimit {int} [可选] [默认值：undefined]

验证文件总大小是否超出限制, 超出则不允许加入队列。

* fileSingleSizeLimit {int} [可选] [默认值：undefined]

验证单个文件大小是否超出限制, 超出则不允许加入队列。

* duplicate {Boolean} [可选] [默认值：undefined]

去重， 根据文件名字、文件大小和最后修改时间来生成hash Key.

* disableWidgets {String, Array} [可选] [默认值：undefined]

默认所有 Uploader.register 了的 widget 都会被加载，如果禁用某一部分，请通过此 option 指定黑名单。

**事件说明**

|  |  |  |
| --- | --- | --- |
| **事件名** | **参数说明** | **描述** |
| dndAccept | * items{DataTransferItemList}   DataTransferItem | 阻止此事件可以拒绝某些类型的文件拖入进来。目前只有 chrome 提供这样的 API，且只能通过 mime-type 验证。 |
| beforeFileQueued | * file {File}   File对象 | 当文件被加入队列之前触发，此事件的handler返回值为false，则此文件不会被添加进入队列。 |
| fileQueued | * file {File}   File对象 | 当文件被加入队列以后触发。 |
| filesQueued | * files {File}   数组，内容为原始File(lib/File）对象。 | 当一批文件添加进队列以后触发。 |
| fileDequeued | * file {File}   File对象 | 当文件被移除队列后触发。 |
| reset |  | 当 uploader 被重置的时候触发。 |
| startUpload |  | 当开始上传流程时触发。 |
| stopUpload |  | 当开始上传流程暂停时触发。 |
| uploadFinished |  | 当所有文件上传结束时触发。 |
| uploadStart | * file {File}   File对象 | 某个文件开始上传前触发，一个文件只会触发一次。 |
| uploadBeforeSend | * object {Object} * data {Object}   默认的上传参数，可以扩展此对象来控制上传参数。   * headers {Object}   可以扩展此对象来控制上传头部。 | 当某个文件的分块在发送前触发，主要用来询问是否要添加附带参数，大文件在开起分片上传的前提下此事件可能会触发多次。 |
| uploadAccept | * object {Object} * ret {Object}   服务端的返回数据，json格式，如果服务端不是json格式，从ret.\_raw中取数据，自行解析。 | 当某个文件上传到服务端响应后，会派送此事件来询问服务端响应是否有效。如果此事件handler返回值为false, 则此文件将派送server类型的uploadError事件。 |
| uploadProgress | * file {File}   File对象   * percentage {Number}   上传进度 | 上传过程中触发，携带上传进度。 |
| uploadError | * file {File}   File对象   * reason {String}   出错的code | 当文件上传出错时触发。 |
| uploadSuccess | * file {File}   File对象   * response {Object}   服务端返回的数据 | 当文件上传成功时触发。 |
| uploadComplete | * file {File} [可选]   File对象 | 不管成功或者失败，文件上传完成时触发。 |
| error | * type {String}   错误类型。 | 当validate不通过时，会以派送错误事件的形式通知调用者。通过upload.on('error', handler)可以捕获到此类错误，目前有以下错误会在特定的情况下派送错来。   * Q\_EXCEED\_NUM\_LIMIT 在设置了fileNumLimit且尝试给uploader添加的文件数量超出这个值时派送。 * Q\_EXCEED\_SIZE\_LIMIT 在设置了Q\_EXCEED\_SIZE\_LIMIT且尝试给uploader添加的文件总大小超出这个值时派送。 * Q\_TYPE\_DENIED 当文件类型不满足时触发。。 |

**option**

* option( key ) ⇒ \*
* option( key, val ) ⇒ self

获取或者设置Uploader配置项。

// 初始状态图片上传前不会压缩

var uploader = new WebUploader.Uploader({

compress: null;

});

// 修改后图片上传前，尝试将图片压缩到1600 \* 1600

uploader.option( 'compress', {

width: 1600,

height: 1600

});

**getStats**

* getStats() ⇒ Object

获取文件统计信息。返回一个包含一下信息的对象。

* successNum 上传成功的文件数
* progressNum 上传中的文件数
* cancelNum 被删除的文件数
* invalidNum 无效的文件数
* uploadFailNum 上传失败的文件数
* queueNum 还在队列中的文件数
* interruptNum 被暂停的文件数

**destroy**

* destroy() ⇒ undefined

销毁 webuploader 实例

**addButton**

* addButton( pick ) ⇒ Promise

添加文件选择按钮，如果一个按钮不够，需要调用此方法来添加。参数跟[options.pick](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_Uploader_options)一致。

uploader.addButton({

id: '#btnContainer',

innerHTML: '选择文件'

});

**makeThumb**

* makeThumb( file, callback ) ⇒ undefined
* makeThumb( file, callback, width, height ) ⇒ undefined

生成缩略图，此过程为异步，所以需要传入callback。 通常情况在图片加入队里后调用此方法来生成预览图以增强交互效果。

当 width 或者 height 的值介于 0 - 1 时，被当成百分比使用。

callback中可以接收到两个参数。

* 第一个为error，如果生成缩略图有错误，此error将为真。
* 第二个为ret, 缩略图的Data URL值。

**注意** Date URL在IE6/7中不支持，所以不用调用此方法了，直接显示一张暂不支持预览图片好了。 也可以借助服务端，将 base64 数据传给服务端，生成一个临时文件供预览。

uploader.on( 'fileQueued', function( file ) {

var $li = ...;

uploader.makeThumb( file, function( error, ret ) {

if ( error ) {

$li.text('预览错误');

} else {

$li.append('&lt;img alt="" src="' + ret + '" />');

}

});

});

**md5File**

* md5File( file[, start[, end]] ) ⇒ promise

计算文件 md5 值，返回一个 promise 对象，可以监听 progress 进度。

uploader.on( 'fileQueued', function( file ) {

var $li = ...;

uploader.md5File( file )

// 及时显示进度

.progress(function(percentage) {

console.log('Percentage:', percentage);

})

// 完成

.then(function(val) {

console.log('md5 result:', val);

});

});

**addFiles**

* addFiles( file ) ⇒ undefined
* addFiles( [file1, file2 ...] ) ⇒ undefined

参数:

* files {Array of File or File} [可选]

Files 对象 数组

添加文件到队列

**removeFile**

* removeFile( file ) ⇒ undefined
* removeFile( id ) ⇒ undefined
* removeFile( file, true ) ⇒ undefined
* removeFile( id, true ) ⇒ undefined

参数:

* file {File, id}

File对象或这File对象的id

移除某一文件, 默认只会标记文件状态为已取消，如果第二个参数为 true 则会从 queue 中移除。

$li.on('click', '.remove-this', function() {

uploader.removeFile( file );

})

**getFiles**

* getFiles() ⇒ Array
* getFiles( status1, status2, status... ) ⇒ Array

返回指定状态的文件集合，不传参数将返回所有状态的文件。

console.log( uploader.getFiles() ); // => all files

console.log( uploader.getFiles('error') ) // => all error files.

**retry**

* retry() ⇒ undefined
* retry( file ) ⇒ undefined

重试上传，重试指定文件，或者从出错的文件开始重新上传。

function retry() {

uploader.retry();

}

**sort**

* sort( fn ) ⇒ undefined

排序队列中的文件，在上传之前调整可以控制上传顺序。

**reset**

* reset() ⇒ undefined

重置uploader。目前只重置了队列。

uploader.reset();

**predictRuntimeType**

* predictRuntimeType() ⇒ String

预测Uploader将采用哪个Runtime

**upload**

* upload() ⇒ undefined
* upload( file | fileId) ⇒ undefined

开始上传。此方法可以从初始状态调用开始上传流程，也可以从暂停状态调用，继续上传流程。

可以指定开始某一个文件。

**stop**

* stop() ⇒ undefined
* stop( true ) ⇒ undefined
* stop( file ) ⇒ undefined

暂停上传。第一个参数为是否中断上传当前正在上传的文件。

如果第一个参数是文件，则只暂停指定文件。

**cancelFile**

* cancelFile( file ) ⇒ undefined
* cancelFile( id ) ⇒ undefined

参数:

* file {File, id}

File对象或这File对象的id

标记文件状态为已取消, 同时将中断文件传输。

$li.on('click', '.remove-this', function() {

uploader.cancelFile( file );

})

**isInProgress**

* isInProgress() ⇒ Boolean

判断Uplaoder是否正在上传中。

**skipFile**

* skipFile( file ) ⇒ undefined

掉过一个文件上传，直接标记指定文件为已上传状态。

**request**

* request( command, args ) ⇒ \* | Promise
* request( command, args, callback ) ⇒ Promise

发送命令。当传入callback或者handler中返回promise时。返回一个当所有handler中的promise都完成后完成的新promise。

**Uploader.register**

* Uploader.register(proto);
* Uploader.register(map, proto);

参数:

* responseMap {object}

API 名称与函数实现的映射

* proto {object}

组件原型，构造函数通过 constructor 属性定义

添加组件

Uploader.register({

'make-thumb': 'makeThumb'

}, {

init: function( options ) {},

makeThumb: function() {}

});

Uploader.register({

'make-thumb': function() {

}

});

**Uploader.unRegister**

* Uploader.unRegister(name);

参数:

* name {string}

组件名字

删除插件，只有在注册时指定了名字的才能被删除。

Uploader.register({

name: 'custom',

'make-thumb': function() {

}

});

Uploader.unRegister('custom');

**Base**

基础类，提供一些简单常用的方法。

**create**

* Base.create( opts ) ⇒ Uploader

创建Uploader实例，等同于new Uploader( opts );

**version**

当前版本号。

**$**

引用依赖的jQuery或者Zepto对象。

**browser**

简单的浏览器检查结果。

* webkit webkit版本号，如果浏览器为非webkit内核，此属性为undefined。
* chrome chrome浏览器版本号，如果浏览器为chrome，此属性为undefined。
* ie ie浏览器版本号，如果浏览器为非ie，此属性为undefined。**暂不支持ie10+**
* firefox firefox浏览器版本号，如果浏览器为非firefox，此属性为undefined。
* safari safari浏览器版本号，如果浏览器为非safari，此属性为undefined。
* opera opera浏览器版本号，如果浏览器为非opera，此属性为undefined。

**os**

操作系统检查结果。

* android 如果在android浏览器环境下，此值为对应的android版本号，否则为undefined。
* ios 如果在ios浏览器环境下，此值为对应的ios版本号，否则为undefined。

**inherits**

* Base.inherits( super ) ⇒ child
* Base.inherits( super, protos ) ⇒ child
* Base.inherits( super, protos, statics ) ⇒ child

参数:

* super {Class}

父类

* protos {Object, Function} [可选]

子类或者对象。如果对象中包含constructor，子类将是用此属性值。

* + constructor {Function} [可选]

子类构造器，不指定的话将创建个临时的直接执行父类构造器的方法。

* statics {Object} [可选]

静态属性或方法。

返回值:

* {Class}

返回子类。

实现类与类之间的继承。

function Person() {

console.log( 'Super' );

}

Person.prototype.hello = function() {

console.log( 'hello' );

};

var Manager = Base.inherits( Person, {

world: function() {

console.log( 'World' );

}

});

// 因为没有指定构造器，父类的构造器将会执行。

var instance = new Manager(); // => Super

// 继承子父类的方法

instance.hello(); // => hello

instance.world(); // => World

// 子类的\_\_super\_\_属性指向父类

console.log( Manager.\_\_super\_\_ === Person ); // => true

**noop**

一个不做任何事情的方法。可以用来赋值给默认的callback.

**bindFn**

* Base.bindFn( fn, context ) ⇒ Function

返回一个新的方法，此方法将已指定的context来执行。

var doSomething = function() {

console.log( this.name );

},

obj = {

name: 'Object Name'

},

aliasFn = Base.bind( doSomething, obj );

aliasFn(); // => Object Name

**log**

* Base.log( args... ) ⇒ undefined

引用Console.log如果存在的话，否则引用一个[空函数noop](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_Base_noop)。

**slice**

* Base.slice( target, start[, end] ) ⇒ Array

被[uncurrythis](http://www.2ality.com/2011/11/uncurrying-this.html)的数组slice方法。 将用来将非数组对象转化成数组对象。

function doSomthing() {

var args = Base.slice( arguments, 1 );

console.log( args );

}

doSomthing( 'ignored', 'arg2', 'arg3' ); // => Array ["arg2", "arg3"]

**guid**

* Base.guid() ⇒ String
* Base.guid( prefx ) ⇒ String

生成唯一的ID

**formatSize**

* Base.formatSize( size ) ⇒ String
* Base.formatSize( size, pointLength ) ⇒ String
* Base.formatSize( size, pointLength, units ) ⇒ String

参数:

* size {Number}

文件大小

* pointLength {Number} [可选] [默认值: 2]

精确到的小数点数。

* [units=[ {Array}

'B', 'K', 'M', 'G', 'TB' ]] 单位数组。从字节，到千字节，一直往上指定。如果单位数组里面只指定了到了K(千字节)，同时文件大小大于M, 此方法的输出将还是显示成多少K.

格式化文件大小, 输出成带单位的字符串

console.log( Base.formatSize( 100 ) ); // => 100B

console.log( Base.formatSize( 1024 ) ); // => 1.00K

console.log( Base.formatSize( 1024, 0 ) ); // => 1K

console.log( Base.formatSize( 1024 \* 1024 ) ); // => 1.00M

console.log( Base.formatSize( 1024 \* 1024 \* 1024 ) ); // => 1.00G

console.log( Base.formatSize( 1024 \* 1024 \* 1024, 0, ['B', 'KB', 'MB'] ) ); // => 1024MB

**Deferred**

* Base.Deferred() ⇒ Deferred

创建一个[Deferred](http://api.jquery.com/category/deferred-object/)对象。 详细的Deferred用法说明，请参照jQuery的API文档。

Deferred对象在钩子回掉函数中经常要用到，用来处理需要等待的异步操作。

// 在文件开始发送前做些异步操作。

// WebUploader会等待此异步操作完成后，开始发送文件。

Uploader.register({

'before-send-file': 'doSomthingAsync'

}, {

doSomthingAsync: function() {

var deferred = Base.Deferred();

// 模拟一次异步操作。

setTimeout(deferred.resolve, 2000);

return deferred.promise();

}

});

**isPromise**

* Base.isPromise( anything ) ⇒ Boolean

参数:

* anything {\*}

检测对象。

返回值:

* {Boolean}

判断传入的参数是否为一个promise对象。

console.log( Base.isPromise() ); // => false

console.log( Base.isPromise({ key: '123' }) ); // => false

console.log( Base.isPromise( Base.Deferred().promise() ) ); // => true

// Deferred也是一个Promise

console.log( Base.isPromise( Base.Deferred() ) ); // => true

**when**

* Base.when( promise1[, promise2[, promise3...]] ) ⇒ Promise

返回一个promise，此promise在所有传入的promise都完成了后完成。 详细请查看[这里](http://api.jquery.com/jQuery.when/)。

**Mediator**

中介者，它本身是个单例，但可以通过[installTo](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_Mediator_installTo)方法，使任何对象具备事件行为。 主要目的是负责模块与模块之间的合作，降低耦合度。

**on**

* on( name, callback[, context] ) ⇒ self

参数:

* name {String}

事件名，支持多个事件用空格隔开

* callback {Function}

事件处理器

* context {Object} [可选]

事件处理器的上下文。

返回值:

* {self}

返回自身，方便链式

绑定事件。

callback方法在执行时，arguments将会来源于trigger的时候携带的参数。如

var obj = {};

// 使得obj有事件行为

Mediator.installTo( obj );

obj.on( 'testa', function( arg1, arg2 ) {

console.log( arg1, arg2 ); // => 'arg1', 'arg2'

});

obj.trigger( 'testa', 'arg1', 'arg2' );

如果callback中，某一个方法return false了，则后续的其他callback都不会被执行到。 切会影响到trigger方法的返回值，为false。

on还可以用来添加一个特殊事件all, 这样所有的事件触发都会响应到。同时此类callback中的arguments有一个不同处， 就是第一个参数为type，记录当前是什么事件在触发。此类callback的优先级比脚低，会再正常callback执行完后触发。

obj.on( 'all', function( type, arg1, arg2 ) {

console.log( type, arg1, arg2 ); // => 'testa', 'arg1', 'arg2'

});

**once**

* once( name, callback[, context] ) ⇒ self

参数:

* name {String}

事件名

* callback {Function}

事件处理器

* context {Object} [可选]

事件处理器的上下文。

返回值:

* {self}

返回自身，方便链式

绑定事件，且当handler执行完后，自动解除绑定。

**off**

* off( [name[, callback[, context] ] ] ) ⇒ self

参数:

* name {String} [可选]

事件名

* callback {Function} [可选]

事件处理器

* context {Object} [可选]

事件处理器的上下文。

返回值:

* {self}

返回自身，方便链式

解除事件绑定

**trigger**

* trigger( name[, args...] ) ⇒ self

参数:

* type {String}

事件名

* ... {\*} [可选]

任意参数

返回值:

* {Boolean}

如果handler中return false了，则返回false, 否则返回true

触发事件

**installTo**

参数:

* obj {Object}

需要具备事件行为的对象。

返回值:

* {Object}

返回obj.

可以通过这个接口，使任何对象具备事件功能。

**File**

* new File( source ) ⇒ File

参数:

* source {Lib.File}

[lib.File](http://fex.baidu.com/webuploader/doc/index.html#Lib.File)实例, 此source对象是带有Runtime信息的。

构造函数

**事件说明**

|  |  |  |
| --- | --- | --- |
| **事件名** | **参数说明** | **描述** |
| statuschange |  | 文件状态变化 |

**name**

文件名，包括扩展名（后缀）

**size**

文件体积（字节）

**type**

文件MIMETYPE类型，与文件类型的对应关系请参考<http://t.cn/z8ZnFny>

**lastModifiedDate**

文件最后修改日期

**id**

文件ID，每个对象具有唯一ID，与文件名无关

**ext**

文件扩展名，通过文件名获取，例如test.png的扩展名为png

**statusText**

状态文字说明。在不同的status语境下有不同的用途。

**setStatus**

* setStatus( status[, statusText] );

参数:

* status {File.Status, String}

[文件状态值](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_File_File_Status)

* statusText {String} [可选] [默认值: '']

状态说明，常在error时使用，用http, abort,server等来标记是由于什么原因导致文件错误。

设置状态，状态变化时会触发change事件。

**File.Status**

文件状态值，具体包括以下几种类型：

* inited 初始状态
* queued 已经进入队列, 等待上传
* progress 上传中
* complete 上传完成。
* error 上传出错，可重试
* interrupt 上传中断，可续传。
* invalid 文件不合格，不能重试上传。会自动从队列中移除。
* cancelled 文件被移除。

**Queue**

文件队列, 用来存储各个状态中的文件。

**stats**

统计文件数。

* numOfQueue 队列中的文件数。
* numOfSuccess 上传成功的文件数
* numOfCancel 被取消的文件数
* numOfProgress 正在上传中的文件数
* numOfUploadFailed 上传错误的文件数。
* numOfInvalid 无效的文件数。
* numofDeleted 被移除的文件数。

**append**

参数:

* file {File}

文件对象

将新文件加入对队列尾部

**prepend**

参数:

* file {File}

文件对象

将新文件加入对队列头部

**getFile**

参数:

* fileId {String}

文件ID

返回值:

* {File}

获取文件对象

**fetch**

* fetch( status ) ⇒ File

参数:

* status {String}

[文件状态值](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_File_File_Status)

返回值:

* {File}

[File](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_File)

从队列中取出一个指定状态的文件。

**sort**

* sort( fn ) ⇒ undefined

参数:

* fn {Function}

排序方法

对队列进行排序，能够控制文件上传顺序。

**getFiles**

* getFiles( [status1[, status2 ...]] ) ⇒ Array

参数:

* status {String} [可选]

[文件状态值](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_File_File_Status)

获取指定类型的文件列表, 列表中每一个成员为[File](http://fex.baidu.com/webuploader/doc/index.html#WebUploader_File)对象。

**removeFile**

* removeFile( file ) ⇒ Array

参数:

* 文件对象。 {File}

在队列中删除文件。