Funktionen der Javascript Bibliothek 'async.js'

Funktionen zur Nutzung von Kollektionen

- `each`, `eachSeries`, `eachLimit`
  - Applies the function iteratee to each item in coll, in parallel
- `forEachOf`, `forEachOfSeries`, `forEachOfLimit`
  - Like each, except that it passes the key (or index) as the second argument to the iteratee.
- `map`, `mapSeries`, `mapLimit`
  - Produces a new collection of values by mapping each value in coll through the iteratee function.
- `filter`, `filterSeries`, `filterLimit`
  - Returns a new array of all the values in coll which pass an async truth test.
- `reject`, `rejectSeries`, `rejectLimit`
  - The opposite of filter. Removes values that pass an async truth test.
- `reduce`
  - Reduces coll into a single value using an async iteratee to return each successive step.
- `reduceRight`
  - Same as reduce, only operates on coll in reverse order.
- `detect`, `detectSeries`, `detectLimit`
  - Returns the first value in coll that passes an async truth test.
- `sortBy`
  - Sorts a list by the results of running each coll value through an async iteratee.
- `some`, `someLimit`, `someSeries`
  - Returns true if at least one element in the coll satisfies an async test.
- `every`, `everyLimit`, `everySeries`
  - Returns true if every element in coll satisfies an async test.
- `concat`, `concatSeries`
  - Applies iteratee to each item in coll, concatenating the results.

Kontrolle der Steuerung

- `series`
  - Run the functions in the tasks collection in series, each one running once the previous function has completed.
- `parallel`, `parallelLimit`
  - Run the tasks collection of functions in parallel, without waiting until the previous function has completed.
- `whilst`
  - Repeatedly call fn, while test returns true.
- `doWhilst`
  - The post-check version of whilst. To reflect the difference in the order of operations, the arguments test and fn are switched.
- `until`
  - Repeatedly call fn until test returns true.
- `doUntil`
  - Like doWhilst, except the test is inverted.
- `during`
  - Like whilst, except the test is an asynchronous function that is passed a callback in the form of function (err, truth)
- `doDuring`
  - The post-check version of during.
- `forever`
  - Calls the asynchronous function fn with a callback parameter that allows it to call itself again, in series, indefinitely.
- `waterfall`
  - Runs the tasks array of functions in series, each passing their results to the next in the array.
- `compose`
  - Creates a function which is a composition of the passed asynchronous functions.
- `seq`
  - Version of the compose function that is more natural to read.
- `applyEach`, `applyEachSeries`

    - Applies the provided arguments to each function in the array, calling callback after all functions have completed.
- `queue`
    - Creates a queue object with the specified concurrency.
- `priorityQueue`
    - The same as queue only tasks are assigned a priority and completed in ascending priority order.
- `cargo`
    - Creates a cargo object with the specified payload.
- `auto`
    - Determines the best order for running the functions in tasks, based on their requirements.
- `autoInject`
    - A dependency-injected version of the auto function.
- `retry`
    - Attempts to get a successful response from task no more than times times before returning an error.
- `retryable`
    - A close relative of retry.
- `iterator`
    - Creates an iterator function which calls the next function in the tasks
- `times`, `timesSeries`, `timesLimit`
    - Calls the iteratee function n times, and accumulates results in the same manner you would use with map.
- `race`
    - Runs the tasks array of functions in parallel, without waiting until the previous function has completed.

Werkzeuge

- `apply`
    - Creates a continuation function with some arguments already applied.
- `nextTick`
    - Calls callback on a later loop around the event loop.
- `memoize`
    - Caches the results of an async function.
- `unmemoize`
    - Undoes a memoized function, reverting it to the original, unmemoized form.
- `ensureAsync`
    - Wrap an async function and ensure it calls its callback on a later tick of the event loop.
- `constant`
    - Returns a function that when called, calls-back with the values provided.
- `asyncify`
    - Take a sync function and make it async, passing its return value to a callback.
- `wrapSync`
- `log`
    - Logs the result of an async function to the console.
- `dir`
    - Logs the result of an async function to the console using console.dir to display the properties of the resulting object.
- `noConflict`
    - Changes the value of async back to its original value, returning a reference to the async object.
- `timeout`
    - Sets a time limit on an asynchronous function.