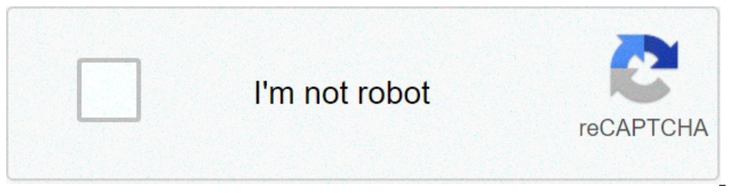

Continue

Nodejs date from string

Why date-fns?With the function-per-file style, you can pick just what you need and stop bloating your project with useless functionality.It works well with modern module bundlers such as webpack, Browserify, and Rollup and also supports tree-shaking.date-fns uses the native Date type and doesn't reinvent the wheel. It doesn't extend core objects for safety's sake.Functions in date-fns work predictably and stick to ECMAScript behavior in edge cases.date-fns is built using pure functions and always returns a new date instance instead of changing the passed one.It helps to prevent bugs and avoid long debugging sessions.date-fns supports both TypeScript and Flow.The typings are generated from the source code and bundled with the package, so they're always up-to-date.The functional programming submodule provides a better alternative to chaining: composition; which makes your code clean and safe and doesn't bloat your build.With support from the amazing community, date-fns has dozens of locales. Only the ones that you use will be included in your project.date-fns always returns a date in the same time zone, no matter what's passed - a timestamp, a string or a date object.The API is tailored to have predictable names and arguments order.date-fns respects timezones & DST.It follows semantic versioning so, always backward compatible.Each build CI checks more than 650 000 examples in about 400 time zones.The best API is an API that doesn't exist. With date-fns you always have one function that does one thing.The API is unambiguous, and there is always a single approach to a problem.In addition to tiny size, date-fns is fast. You can be sure that your users will have the best user experience.Every date-fns function has a detailed description with examples. The documentation is accessible both online on the website and offline thanks to JSDoc annotations.date-fns is an active development and we are constantly adding new features. If you know the format of an input string, you can use that to parse a date. This dependent on CustomParseFormat plugin to work dayjs.extend(customParseFormat) dayjs("12-25-1995", "MM-DD-YYYY") Pass the locale key as the third parameter to parse locale-aware date time string. require('dayjs/locale/es') dayjs('2018 Enero 15', 'YYYY MMMM DD', 'es') You may specify a boolean for the last argument to use strict parsing. Strict parsing requires that the format and input match exactly, including delimiters. dayjs('1970-00-00', 'YYYY-MM-DD').isValid() // true dayjs('1970-00-00', 'YYYY-MM-DD', true).isValid() // false dayjs('1970-00-00', 'YYYY-MM-DD', 'es', true).isValid() // false If you don't know the exact format of an input string, but know it could be one of many, you can use an array of formats. dayjs("12-25-2001", ["YYYY", "YYYY-MM-DD"], 'es', true); List of all available parsing tokens InputExampleDescription YY18Two-digit year YYYY2018Four-digit year M1-12Month, beginning at 1 MM01-12Month, 2-digits MMMJan-DecThe abbreviated month name MMMMJanuary-DecemberThe full month name D1-31Day of month DD01-31Day of month, 2-digits H0-23Hours HH00-23Hours, 2-digits h1-12Hours, 12-hour clock hh01-12Hours, 12-hour clock, 2-digits m0-59Minutes mm00-59Minutes, 2-digits s0-59Seconds ss00-59Seconds, 2-digits S0-9Hundreds of milliseconds, 1-digit SS00-99Tens of milliseconds, 2-digits SSS000-999Milliseconds, 3-digits Z-05:00Offset from UTC ZZ-0500Compact offset from UTC, 2-digits AAM PMPost or ante meridiem, upper-case aam pmPost or ante meridiem, lower-case Do1st... 31stDay of Month with ordinal The Date.parse() method parses a string representation of a date, and returns the number of milliseconds since January 1, 1970, 00:00:00 UTC or NaN if the string is unrecognized or, in some cases, contains illegal date values (e.g. 2015-02-31). It is not recommended to use Date.parse as until ES5, parsing of strings was entirely implementation dependent. There are still many differences in how different hosts parse date strings, therefore date strings should be manually parsed (a library can help if many different formats are to be accommodated). Direct call: Implicit call: A number representing the milliseconds elapsed since January 1, 1970, 00:00:00 UTC and the date obtained by parsing the given string representation of a date. If the argument doesn't represent a valid date, NaN is returned.The parse() method takes a date string (such as "2011-10-10T14:48:00") and returns the number of milliseconds since January 1, 1970, 00:00:00 UTC. This function is useful for setting date values based on string values, for example in conjunction with the setTime() method and the Date object.The standard string representation of a date time string is a simplification of the ISO 8601 calendar date extended format. (See the section Date Time String Format in the ECMAScript specification for more details.) For example, "2011-10-10" (date-only form), "2011-10-10T14:48:00" (date-time form), or "2011-10-10T14:48:00.000+09:00" (date-time form with milliseconds and time zone) can be passed and will be parsed. When the time zone offset is absent, date-only forms are interpreted as a UTC time and date-time forms are interpreted as local time. While time zone specifiers are used during date string parsing to interpret the argument, the value returned is always the number of milliseconds between January 1, 1970 00:00:00 UTC and the point in time represented by the argument or NaN. Because parse() is a static method of Date, it is called as Date.parse() rather than as a method of a Date instance. Note: This section contains implementation-specific behavior that can be inconsistent across implementations. The ECMAScript specification states: If the String does not conform to the standard format the function may fall back to any implementation–specific heuristics or implementation–specific parsing algorithm. Unrecognizable strings or dates containing illegal element values in ISO formatted strings shall cause Date.parse() to return NaN. However, invalid values in date strings not recognized as simplified ISO format as defined by ECMA-262 may or may not result in NaN, depending on the browser and values provided, e.g.: will be treated as a local date of 25 November, 2015 in Firefox 30 and an invalid date in Safari 7. However, if the string is recognized as an ISO format string and it contains invalid values, it will return NaN in all browsers compliant with ES5 and later: new Date('2014-25-23').toISOString(); SpiderMonkey's implementation-specific heuristic can be found in jsdate.cpp. The string "10 06 2014" is an example of a non-conforming ISO format and thus falls back to a custom routine. See also this rough outline on how the parsing works. will be treated as a local date of 6 October, 2014, and not 10 June, 2014. Other examples: new Date('foo-bar 2014').toString(); Date.parse('foo-bar 2014'); Note: This section contains implementation-specific behavior that can be inconsistent across implementations. Given a non-standard date string of "March 7, 2014", parse() assumes a local time zone, but given a simplification of the ISO 8601 calendar date extended format such as "2014-03-07", it will assume a time zone of UTC (ES5 and ECMAScript 2015). Therefore Date objects produced using those strings may represent different moments in time depending on the version of ECMAScript supported unless the system is set with a local time zone of UTC. This means that two date strings that appear equivalent may result in two different values depending on the format of the string that is being converted.The following calls all return 1546300800000. The first according to ES5 will imply UTC time, and the others are specifying UTC timezone via the ISO date specification (Z and +00:00) Date.parse("2019-01-01") Date.parse("2019-01-01T00:00:00.000Z") Date.parse("2019-01-01T00:00:00.000+00:00") The following call, which does not specify a time zone will be set to 2019-01-01 at 00:00:00 in the local timezone of the system. Date.parse("2019-01-01T00:00:00") Note: This section contains implementation-specific behavior that can be inconsistent across implementations. If IPOdate is an existing Date object, it can be set to August 9, 1995 (local time) as follows: IPOdate.setTime(Date.parse('Aug 9, 1995')); Some other examples of parsing non-standard date strings: Date.parse('Aug 9, 1995'); Returns 807937200000 in time zone GMT-0300, and other values in other time zones, since the string does not specify a time zone and is not ISO format, therefore the time zone defaults to local. Date.parse('Wed, 09 Aug 1995 00:00:00 GMT'); Returns 807926400000 no matter the local time zone as GMT (UTC) is provided. Date.parse('Wed, 09 Aug 1995 00:00:00'); Returns 807937200000 in time zone GMT-0300, and other values in other time zones, since there is no time zone specifier in the argument and it is not ISO format, so is treated as local. Date.parse('Thu, 01 Jan 1970 00:00:00 GMT'); Returns 0 no matter the local time zone as a time zone GMT (UTC) is provided. Date.parse('Thu, 01 Jan 1970 00:00:00'); Returns 14400000 in time zone GMT-0400, and other values in other time zones, since no time zone is provided and the string is not in ISO format, therefore the local time zone is used. Date.parse('Thu, 01 Jan 1970 00:00:00 GMT-0400'); Returns 14400000 no matter the local time zone as a time zone GMT (UTC) is provided.SpecificationECMAScript Language Specification (ECMAScript)# sec-date.parseBCD tables only load in the browserCompatibility notes Firefox 49 changed the parsing of 2-digit years to be aligned with the Google Chrome browser instead of Internet Explorer. Now, 2-digit years that are less than 50 are parsed as 21st century years. For example, 04/16/17, previously parsed as April 16, 1917, will be April 16, 2017 now. To avoid any interoperability issues or ambiguous years, it is recommended to use the ISO 8601 format like "2017-04-16" (bug 1265136). Google Chrome will accept a numerical string as a valid dateString parameter. This means that, for instance, while !!Date.parse("42") evaluates to false in Firefox, it evaluates to true in Google Chrome because "42" is interpreted as the first of January 2042. Show / Hide Table of Contents There was an error loading this resource. Please try again later. Reference > Operators > Aggregation Pipeline Operators$dateFromString¶Converts a date/time string to a date object.The $dateFromString expression has the following syntax:{ $dateFromString: { dateString: , format: , timezone: , onError: , onNull: } }The $dateFromString takes a document with the following fields:FieldDescriptiondateStringThe date/time string to convert to a date object. See Date for more information on date/time formats.If specifying the timezone option to the operator, do not include time zone information in the dateString.formatOptional. The date format specification of the dateString. The format can be any expression that evaluates to a string literal, containing 0 or more format specifiers. For a list of specifiers available, see Format Specifiers.If unspecified, $dateFromString uses "%Y-%m-%dT%H:%M:%S.%LZ" as the default format.timezoneOptional. The time zone to use to format the date.If the dateString argument is formatted like '2017-02-08T12:10:40.787Z', in which the 'Z' at the end indicates Zulu time (UTC time zone), you cannot specify the timezone argument. allows for the following options and expressions that evaluate to them:an Olson Timezone Identifier, such as "Europe/London" or "America/New_York", ora UTC offset in the form:+/-[hh]:[mm], e.g. "+04:45", or+/-[hh][mm], e.g. "-0530", or+/-[hh], e.g. "+03", orThe strings "Z", "UTC", or "GMT"For more information on expressions, see Expressions.onErrorOptional. If $dateFromString encounters an error while parsing the given dateString, it outputs the result value of the provided onError expression. This result value can be of any type.If you do not specify onError, $dateFromString throws an error if it cannot parse dateString.onNullOptional. If the dateString provided to $dateFromString is null or missing, it outputs the result value of the provided onNull expression. This result value can be of any type.If you do not specify onNull and dateString is null or missing, then $dateFromString outputs null.TipSee also: ExampleResults{ $dateFromString: { dateString: "2017-02-08T12:10:40.787"} }ISODate("2017-02-08T12:10:40.787Z"){ $dateFromString: { dateString: "2017-02-08T12:10:40.787", timezone: "America/New_York"} }ISODate("2017-02-08T17:10:40.787Z"){ $dateFromString: { dateString: "2017-02-08"} }ISODate("2017-02-08T00:00:00Z"){ $dateFromString: { dateString: "06-15-2018", format: "%m-%d-%Y"} }ISODate("2018-06-15T00:00:00Z"){ $dateFromString: { dateString: "15-06-2018", format: "%d-%m-%Y"} }ISODate("2018-06-15T00:00:00Z")The following format specifiers are available for use in the :SpecifiersDescriptionPossible Values%dDay of Month (2 digits, zero padded)01-31%GYear in ISO 8601 format0000-9999%HHour (2 digits, zero padded, 24-hour clock)00-23%LMillisecond (3 digits, zero padded)000-999%mMonth (2 digits, zero padded)01-12%MMinute (2 digits, zero padded)00-59%SSecond (2 digits, zero padded)00-60%uDay of week number in ISO 8601 format (1-Monday, 7-Sunday)1-7%VWeek of Year in ISO 8601 format1-53%YYear (4 digits, zero padded)0000-9999%zThe timezone offset from UTC.+/-[hh][mm]%ZThe minutes offset from UTC as a number. For example, if the timezone offset (+/-[hhmm]) was +0445, the minutes offset is +285.+/-mmm%%Percent Character as a Literal%Consider a collection logmessages that contains the following documents with dates.{ _id: 1, date: "2017-02-08T12:10:40.787", timezone: "America/New_York", message: "Step 1: Started" },{ _id: 2, date: "2017-02-08", timezone: "-05:00", message: "Step 1: Ended" },{ _id: 3, message: " Step 1: Ended " },{ _id: 4, date: "2017-02-09", timezone: "Europe/London", message: "Step 2: Started"},{ _id: 5, date: "2017-02-09T03:35:02.055", timezone: "+0530", message: "Step 2: In Progress"}The following aggregation uses $dateFromString to convert the date value to a date object:db.logmessages.aggregate([{ $project: { date: { $dateFromString: { dateString: '$date', timezone: 'America/New_York' } } } }])The above aggregation returns the following documents and converts each date field to the Eastern Time Zone:{ "_id" : 1, "date" : ISODate("2017-02-08T17:10:40.787Z") }{ "_id" : 2, "date" : ISODate("2017-02-08T05:00:00Z") }{ "_id" : 3, "date" : null }{ "_id" : 4, "date" : ISODate("2017-02-09T05:00:00Z") }{ "_id" : 5, "date" : ISODate("2017-02-09T08:35:02.055Z") }The timezone argument can also be provided through a document field instead of a hard coded argument. For example:db.logmessages.aggregate([{ $project: { date: { $dateFromString: { dateString: '$date', timezone: '$timezone' } } } }])The above aggregation returns the following documents and converts each date field to their respective UTC representations.{ "_id" : 1, "date" : ISODate("2017-02-08T17:10:40.787Z") }{ "_id" : 2, "date" : ISODate("2017-02-08T05:00:00Z") }{ "_id" : 3, "date" : null }{ "_id" : 4, "date" : ISODate("2017-02-09T00:00:00Z") }{ "_id" : 5, "date" : ISODate("2017-02-08T22:05:02.055Z") }If your collection contains documents with unparsable date strings, $dateFromString throws an error unless you provide an aggregation expression to the optional onError parameter.For example, given a collection dates with the following documents:{ "_id" : 1, "date" : "2017-02-08T12:10:40.787", timezone: "America/New_York" },{ "_id" : 2, "date" : "20177-02-09T03:35:02.055", timezone: "America/New_York" }You can use the onError parameter to return the invalid date in its original string form:db.dates.aggregate([{ $project: { date: { $dateFromString: { dateString: '$date', timezone: '$timezone', onError: '$date' } } } }])This returns the following documents:{ "_id" : 1, "date" : ISODate("2017-02-08T17:10:40.787Z") }{ "_id" : 2, "date" : "20177-02-09T03:35:02.055" }If your collection contains documents with null date strings, $dateFromString returns null unless you provide an aggregation expression to the optional onNull parameter.For example, given a collection dates with the following documents:{ "_id" : 1, "date" : "2017-02-08T12:10:40.787", timezone: "America/New_York" },{ "_id" : 2, "date" : null, timezone: "America/New_York" }You can use the onNull parameter to have $dateFromString return a date representing the unix epoch instead of null:db.dates.aggregate([{ $project: { date: { $dateFromString: { dateString: '$date', timezone: '$timezone', onNull: new Date(0) } } } }])This returns the following documents:{ "_id" : 1, "date" : ISODate("2017-02-08T17:10:40.787Z") }{ "_id" : 2, "date" : ISODate("1970-01-01T00:00:00Z") }Give Feedback← $dateToParts (aggregation)$dateToString (aggregation) →On this pageDefinitionBehaviorFormat SpecifiersExamples