# vue3 rotuer

## vue3 router 源码part1

createRouterMatcher

官方使用，

```
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
]

// 3. 创建路由实例并传递 `routes` 配置
// 你可以在这里输入更多的配置，但我们在这里
// 暂时保持简单
const router = VueRouter.createRouter({
  // 4. 内部提供了 history 模式的实现。为了简单起见，我们在这里使用 hash 模式。
  history: VueRouter.createWebHashHistory(),
  routes, // `routes: routes` 的缩写
})
```

进入 createRouter函数

```
//router.js
export function createRouter(options: RouterOptions): Router {
  const matcher = createRouterMatcher(options.routes, options)
  const parseQuery = options.parseQuery || originalParseQuery
  const stringifyQuery = options.stringifyQuery || originalStringifyQuery
  const routerHistory = options.history
  if (__DEV__ && !routerHistory)
    throw new Error(
      'Provide the "history" option when calling "createRouter()":' +
        ' https://next.router.vuejs.org/api/#history.'
    )
```

本来应该开始讲解createRouter，只是代码很长，而且为了更好的理解createRouter,需要先了解比如createRouterMatcher 等的实现。所以优先需要理解createRouterMatcher。

```
export function createRouterMatcher(
  routes: RouteRecordRaw[],
  globalOptions: PathParserOptions
): RouterMatcher {
  // normalized ordered array of matchers
  const matchers: RouteRecordMatcher[] = []
  const matcherMap = new Map<RouteRecordName, RouteRecordMatcher>()
  globalOptions = mergeOptions(
    { strict: false, end: true, sensitive: false } as PathParserOptions,
    globalOptions
  )
```

1. 创建matchers 数组，放入所有的router record 包括alias （后面有讲解）
2. 创建matcherMap 用来存放original router record，没有alias （后面有讲解）
3. 融合用户传入的选项和默认的选项（默认选项比如stict，end， sensitive），如果用户提供了相同选项，则优先选择用户提供的选项。

```
// src/matcher/index.ts
function getRecordMatcher(name: RouteRecordName) {
    return matcherMap.get(name)
  }
```

1. 通过name 获取 matcherMap 里面对应的值。里面存放的都是rotuer record

```
function addRoute(
    record: RouteRecordRaw,
    parent?: RouteRecordMatcher,
    originalRecord?: RouteRecordMatcher
  ) {
    // used later on to remove by name
    const isRootAdd = !originalRecord
    const mainNormalizedRecord = normalizeRouteRecord(record)
    // we might be the child of an alias
    mainNormalizedRecord.aliasOf = originalRecord && originalRecord.record
    const options: PathParserOptions = mergeOptions(globalOptions, record)
    // generate an array of records to correctly handle aliases
    const normalizedRecords: typeof mainNormalizedRecord[] = [
      mainNormalizedRecord,
    ]
    if ('alias' in record) {
      const aliases =
        typeof record.alias === 'string' ? [record.alias] : record.alias!
      for (const alias of aliases) {
        normalizedRecords.push(
          assign({}, mainNormalizedRecord, {
            // this allows us to hold a copy of the `components` option
            // so that async components cache is hold on the original record
            components: originalRecord
              ? originalRecord.record.components
              : mainNormalizedRecord.components,
            path: alias,
            // we might be the child of an alias
            aliasOf: originalRecord
              ? originalRecord.record
              : mainNormalizedRecord,
            // the aliases are always of the same kind as the original since they
          // are defined on the same record
        }) as typeof mainNormalizedRecord
      )
    }
  }

  let matcher: RouteRecordMatcher
  let originalMatcher: RouteRecordMatcher | undefined

  for (const normalizedRecord of normalizedRecords) {
    const { path } = normalizedRecord
    // Build up the path for nested routes if the child isn't an absolute
    // route. Only add the / delimiter if the child path isn't empty and if the
    // parent path doesn't have a trailing slash
    if (parent && path[0] !== '/') {
      const parentPath = parent.record.path
      const connectingSlash =
        parentPath[parentPath.length - 1] === '/' ? '' : '/'
      normalizedRecord.path =
        parent.record.path + (path && connectingSlash + path)
    }

    if (__DEV__ && normalizedRecord.path === '*') {
      throw new Error(
        'Catch all routes ("*") must now be defined using a param with a custom regexp.\
          'See more at https://next.router.vuejs.org/guide/migration/#removed-star-or-ca
      )
    }

    // create the object before hand so it can be passed to children
    matcher = createRouteRecordMatcher(normalizedRecord, parent, options)

    if (__DEV__ && parent && path[0] === '/')
      checkMissingParamsInAbsolutePath(matcher, parent)

    // if we are an alias we must tell the original record that we exist
    // so we can be removed
    if (originalRecord) {
      originalRecord.alias.push(matcher)
      if (__DEV__) {
        checkSameParams(originalRecord, matcher)
      }
    } else {
      // otherwise, the first record is the original and others are aliases
      originalMatcher = originalMatcher || matcher
      if (originalMatcher !== matcher) originalMatcher.alias.push(matcher)

      // remove the route if named and only for the top record (avoid in nested calls)
      // this works because the original record is the first one
      if (isRootAdd && record.name && !isAliasRecord(matcher))
        removeRoute(record.name)
    }

```

```
      if ('children' in mainNormalizedRecord) {
        const children = mainNormalizedRecord.children
        for (let i = 0; i < children.length; i++) {
          addRoute(
            children[i],
            matcher,
            originalRecord && originalRecord.children[i]
          )
        }
      }

      // if there was no original record, then the first one was not an alias and all
      // other alias (if any) need to reference this record when adding children
      originalRecord = originalRecord || matcher

      // TODO: add normalized records for more flexibility
      // if (parent && isAliasRecord(originalRecord)) {
      //   parent.children.push(originalRecord)
      // }

      insertMatcher(matcher)
    }

    return originalMatcher
      ? () => {
          // since other matchers are aliases, they should be removed by the original matc
          removeRoute(originalMatcher!)
        }
      : noop
  }
```

1. 调用normalizeRouterRecord 创建一个基于用户传入配置对象的完整对象，里面存放作为参数传入的record的相关信息，比如path, redirect, name, meta, children, 等等，也添加了其他相关信息，比如aliasOf等等（342行）
2. 确认用户传入配置对象是否包含alias, 如果有，不管是数组还是字符串，一律转换成数组。然后遍历alias数组，对每一个alias都创建一个类似第一步的配置对象，所有的配置选项都相同除了path，path的值变成了用户传入的alias里的值。这样就完成了对alias里面没每一个元素都创建一个配置对象。
3. 然后基于第二步创建好的配置对象，normalizedRecords又一次进行遍历，目的就是处理alias的相对和绝对路径的问题。（https://router.vuejs.org/zh/guide/essentials/redirect-and-alias.html）
4. 然后调用createRouteRecordMatcher方法给每一个alias都创建一个rotuer 记录，它们都有着相同的配置比如component和其他配置，不同的是path。这样就能完成多个alias元素都能渲染同一个component的功能。
5. createRouteRecordMatcher内部实现非常有意思。使用了有限状态机对path进行词法分析，这么做理由是path可能包含了正则，也可能包含了param(:param), 所以path的计算模型非常复杂，使用有限状态机能清晰的解决状态之间的逻辑和转换。源码里面使用了while配合switch根据当前读取的char，决定当前

   的状态已经当前应该做的操作，比如路径 /abc/:user ，读取 /,a,b,c,/ 这样按照顺序, 在第二个/读取出来，就作为分隔，可以提取abc, 这样每一个提取出来的产物称作token,token的类型是一个对象，形如{type:

Tokentype.Static, value: buffer}，然后继续读取，第二个/之后的的char就是:, 状态从 TokenizerState.Static 转变成为 TokenizerState.Param，那么当前的状态变成了读取param了。确实 /abc/:user, 读取了: 之后开始就是param。但是更加复杂的path比如 /a/:user(\\d+)，那么当读取\\就会改变状态成TokenizerState.ParamRegExp，类似这样的按照一定逻辑提取出来自己需要的部分。

6. 然后第一个的matcher就是originalMatcher，然后其他创建出来的alias router记录，就存入了originalMatcher的alias属性。
7. 如果当前用户传入的的rotuer有children，则递归调用addRoute进行添加。
8. 然后通过insertMatcher函数添加原版matcher进一开始创建的matcherMap里面。

```
function removeRoute(matcherRef: RouteRecordName | RouteRecordMatcher) {
   if (isRouteName(matcherRef)) {
     const matcher = matcherMap.get(matcherRef)
     if (matcher) {
       matcherMap.delete(matcherRef)
       matchers.splice(matchers.indexOf(matcher), 1)
       matcher.children.forEach(removeRoute)
       matcher.alias.forEach(removeRoute)
     }
   } else {
     const index = matchers.indexOf(matcherRef)
     if (index > -1) {
       matchers.splice(index, 1)
       if (matcherRef.record.name) matcherMap.delete(matcherRef.record.name)
       matcherRef.children.forEach(removeRoute)
       matcherRef.alias.forEach(removeRoute)
     }
   }
 }
```

1. 如果传入的matcherRef 类型 string 或者symbol则做下面 2-5
2. 从map获取需要删除的matcher (router record)
3. 然后删除，也对应的从matchers删除 (router record)
4. 从当前的matcher的children 删除
5. 从当前的matcher的alias删除
6. 如果不是string 或者symbol，就是对象，则从matchers数组找出相同的对象，如果有则从matcherMap, 从当前的matcher的children 删除，从当前的matcher的alias删除