1、全局守卫： `router.beforeEach`

2、全局解析守卫： `router.beforeResolve`

3、全局后置钩子： `router.afterEach`

4、路由独享的守卫： `beforeEnter`

5、组件内的守卫： `beforeRouteEnter`、`beforeRouteUpdate (2.2 新增)`、`beforeRouteLeave`

完整路由解析过程

`beforeRouteLeave` > `beforeEach` > `beforeRouteUpdate` > `beforeEnter` > 解析路由 > `beforeROuteEnter` > `beforeResolve` > `afterEach` > 渲染dom > 给`beforeRouteEnter`的`next`传递回调

导航表示路由正在发生改变，`vue-router` 提供的导航守卫主要用来:通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中：`全局的`, `单个路由独享的`, 或者`组件级的`。

注意：**参数或查询的改变并不会触发进入/离开的导航守卫。** 你可以通过 观察 $route 对象 来应对这些变化，或使用 `beforeRouteUpdate`的组件内守卫。

1、全局守卫：

使用 `router.beforeEach` 注册一个全局前置守卫：

```
const router = new VueRouter({ ... })
  router.beforeEach((to, from, next) => {
  // ...
})
```

当一个导航触发时，`全局前置守卫`按照创建顺序调用。守卫是异步解析执行，此时导航在所有守卫 `resolve 完之前`一直处于`等待中`。

每个守卫方法接收三个参数：

`to: Route`: 即将要进入的目标 路由对象

`from: Route`: 当前导航正要离开的路由

`next: Function`: 一定要调用该方法来`resolve`这个钩子。执行效果`依赖 next 方法的调用参数`。

- `next()`: 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是`confirmed` (确认的)。

- `next(false)`: 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 `from` 路由对应的地址。
- `next('/')` 或者 `next({ path: '/' })`: 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 `next` 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在`router-link` 的 `to prop`或 `router.push`中的选项。
- `next(error)`: (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 `router.onError()`注册过的回调。

确保要调用 `next`方法，否则钩子就不会被 `resolved`。

2、全局解析守卫：

2.5.0 新增

在 `2.5.0+` 你可以用 `router.beforeResolve` 注册一个全局守卫。这和 `router.beforeEach` 类似，区别是：在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被调用。

3、全局后置钩子

你也可以注册全局后置钩子，然而和守卫不同的是，这些钩子不会接受 `next` 函数也不会改变导航本身：

```
router.afterEach((to, from) => {
  // ...
})
```

4、路由独享的守卫

你可以在路由配置上直接定义 `beforeEnter` 守卫：

```
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from, next) => {
        // ...
      }
    }
  ]
})
```

这些守卫与全局前置守卫的方法参数是一样的。

5、组件内的守卫

最后，你可以在路由组件内直接定义以下路由导航守卫：

- `beforeRouteEnter`
- `beforeRouteUpdate` (2.2 新增)
- `beforeRouteLeave`

```
const Foo = {
  template: `...`,
  beforeRouteEnter (to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不！能！获取组件实例 `this`
    // 因为当守卫执行前，组件实例还没被创建
  },
  //不过，你可以通过传一个回调给 next来访问组件实例。
  //在导航被确认的时候执行回调，并且把组件实例作为回调方法的参数。
  beforeRouteEnter (to, from, next) {
    next(vm => {
      // 通过 `vm` 访问组件实例
    })
  },
  beforeRouteUpdate (to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2
之间跳转的时候，
    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这
个情况下被调用。
    // 可以访问组件实例 `this`
  },
  beforeRouteLeave (to, from, next) {
    // 导航离开该组件的对应路由时调用
    // 可以访问组件实例 `this`
  }
}
```

注意：`beforeRouteEnter` **是支持给**`next` **传递回调的唯一守卫。**对于 `beforeRouteUpdate` 和 `beforeRouteLeave` 来说，`this` 已经可用了，所以不支持传递回调，因为没有必要了：

```
beforeRouteUpdate (to, from, next) {
  // just use `this`
  this.name = to.params.name
  next()
}
```

离开守卫beforeRouteLeave：通常用来禁止用户在还未保存修改前突然离开。该导航可以通过 next(false) 来取消：

```
beforeRouteLeave (to, from , next) {
  const answer = window.confirm('Do you really want to leave? you have unsaved changes!')
  if (answer) {
    next()
  } else {
    next(false)
  }
}
```