> TIP
>
> 为了简单起见，所有的路由**都省略了** `component` **属性**，只关注 `path` 值。

在参数中自定义正则

当定义像 `:userId` 这样的参数时，我们内部使用以下的正则 `([^/]+)` (至少有一个字符不是斜杠 `/`)来从 URL 中提取参数。这很好用，除非你需要根据参数的内容来区分两个路由。想象一下，两个路由 `/:orderId` 和 `/:productName`，两者会匹配完全相同的 URL，所以我们需要一种方法来区分它们。最简单的方法就是在路径中添加一个静态部分来区分它们：

```js
const routes = [
  // 匹配 /o/3549
  { path: '/o/:orderId' },
  // 匹配 /p/books
  { path: '/p/:productName' },
]
```

但在某些情况下，我们并不想添加静态的 `/o` `/p` 部分。由于，`orderId` 总是一个数字，而 `productName` 可以是任何东西，所以我们可以在括号中为参数指定一个自定义的正则：

```js
const routes = [
  // /:orderId -> 仅匹配数字
  { path: '/:orderId(\\d+)' },
  // /:productName -> 匹配其他任何内容
  { path: '/:productName' },
]
```

现在，转到 `/25` 将匹配 `/:orderId`，其他情况将会匹配 `/:productName`。`routes` 数组的顺序并不重要!

可重复的参数

如果你需要匹配具有多个部分的路由，如 `/first/second/third`，你应该用 `*`（0 个或多个）和 `+`（1 个或多个）将参数标记为可重复：

```js
const routes = [
  // /:chapters ->  匹配 /one, /one/two, /one/two/three, 等
  { path: '/:chapters+' },
  // /:chapters -> 匹配 /, /one, /one/two, /one/two/three, 等
  { path: '/:chapters*' },
]
```

这将为你提供一个参数数组，而不是一个字符串，并且在使用命名路由时也需要你传递一个数组：

```js
// 给定 { path: '/:chapters*', name: 'chapters' },
router.resolve({ name: 'chapters', params: { chapters: [] } }).href
// 产生 /
router.resolve({ name: 'chapters', params: { chapters: ['a', 'b'] } }).href
// 产生 /a/b

// 给定 { path: '/:chapters+', name: 'chapters' },
router.resolve({ name: 'chapters', params: { chapters: [] } }).href
// 抛出错误，因为 `chapters` 为空
```

这些也可以通过在**右括号后**添加它们与自定义正则结合使用：

```js
const routes = [
  // 仅匹配数字
  // 匹配 /1, /1/2, 等
  { path: '/:chapters(\\d+)+' },
  // 匹配 /, /1, /1/2, 等
  { path: '/:chapters(\\d+)*' },
]
```

可选参数

你也可以通过使用 `?` 修饰符(0 个或 1 个)将一个参数标记为可选：

```js
const routes = [
  // 匹配 /users 和 /users/posva
  { path: '/users/:userId?' },
  // 匹配 /users 和 /users/42
  { path: '/users/:userId(\\d+)?' },
]
```

请注意，`*` 在技术上也标志着一个参数是可选的，但 `?` 参数不能重复。