BLOG / FIELD NOTES

Astro 双语内容路由：把语言、内容与 URL 对齐

从内容集合到静态路由，梳理双语站点中内容标识、列表过滤、语言切换与缺失译文回退的实现。

2026-10-19　Astro / i18n / 内容集合

给一个静态站点增加中英文页面，看起来只是给 URL 加上 `/zh/` 和 `/en/`。真正动手以后，问题很快会扩散到内容文件、详情页参数、导航链接和语言切换：同一篇内容怎样配对？没有译文时跳到哪里？构建阶段怎样保证不会生成跨语言的错误页面？

这次改造采用了一个简单约定：**目录表达语言，文件名表达内容身份**。

```
src/content/blog/
├── zh/mqtt-reconnect-demo.mdx
└── en/mqtt-reconnect-demo.mdx
```

两个文件共享 `mqtt-reconnect-demo` 这个内容键，同时各自声明 `locale`。这样既能从文件结构判断语言，也能让内容集合在读取 frontmatter 时校验语言字段。

先拆开内容 ID

Astro 内容集合返回的 `id` 包含相对路径，例如：

```
zh/mqtt-reconnect-demo
```

如果直接把这个 ID 当作 slug，最终 URL 会重复携带语言信息。路由层需要先把它拆成两个概念：

```ts
export function contentLocale(id) {
  return id.split('/', 1)[0];
}

export function contentKey(id) {
  return id.split(/\/(.+)/)[1] ?? '';
}
```

`contentLocale()` 用于识别来源语言，`contentKey()` 用于生成稳定的详情页地址。这个拆分也让后续的译文配对不依赖标题或文件内容。

让静态路由由内容本身驱动

详情页使用 `[locale]/blog/[slug].astro`。构建路径时，语言来自 frontmatter，slug 来自内容键：

```ts
export async function getStaticPaths() {
  const articles = await getCollection('blog', ({ data }) => !data.draft);

  return articles.map((article) => ({
    params: {
      locale: article.data.locale,
      slug: contentKey(article.id),
    },
    props: { article },
  }));
}
```

这一步有两个边界值得固定下来：草稿不能进入构建结果，URL 中的语言必须与内容语言一致。文章列表还要再按当前语言过滤，否则中文列表可能混入英文卡片：

```ts
const articles = await getCollection(
  'blog',
  ({ data }) => !data.draft && data.locale === locale,
);
```

语言切换需要知道译文是否存在

普通页面可以把 `/zh/about/` 机械地换成 `/en/about/`，内容详情页却不能假定译文一定存在。如果目标语言下没有同名内容，直接切换会得到一个没有生成的 URL。

当前实现先收集所有已发布内容的 ID，再判断目标语言和内容键组成的 ID 是否存在：

```
export function counterpartPath(locale, type, id, translatedIds) {
  const key = contentKey(id);

  return translatedIds.has(`${locale}/${key}`)
    ? localizedPath(locale, `${type}/${key}`)
    : localizedPath(locale, type);
}
```

有译文时进入对应详情页；没有译文时回到目标语言的文章列表。这个回退保留了用户切换语言的意图，也避免制造 404。

根路径只负责选择语言

站点根路径没有具体内容，它只根据已保存的选择和浏览器语言进入对应首页。优先级可以写得很明确：

使用用户之前主动选择的语言。

没有记录时，读取浏览器语言。

只有明确的英文环境进入英文站，其余情况使用中文。

语言开关点击后把选择写入 `localStorage`，下次访问根路径时继续使用。即使存储不可用，也应提供中文首页作为可预测的回退。

用小函数锁住路由规则

多语言改造最容易出现的错误往往不是页面渲染，而是斜杠、前缀和回退路径的组合。把这些规则集中到纯函数以后，可以用少量测试覆盖关键行为：

```
assert.equal(localizedPath('en', '/blog/'), '/en/blog/');
assert.equal(contentKey('zh/mqtt-reconnect-demo'), 'mqtt-reconnect-demo');
assert.equal(
  counterpartPath('en', 'blog', 'zh/mqtt-reconnect-demo', new Set()),
  '/en/blog/',
);
```

测试关注的是路由契约，而不是组件内部写法。页面和导航可以继续调整，只要这些契约不变，双语路径就不会悄悄漂移。

复盘

双语站点的核心不是翻译按钮，而是建立稳定的内容身份。把语言和内容键分开以后，列表过滤、静态路径生成、译文配对和缺失译文回退都能围绕同一组规则实现。以后增加第三种语言时，需要扩展语言配置和文案，但不必重新设计内容 URL。