ES6，import时如何正确使用花括号'{ }'

在 ES6 之前，社区制定了一些模块加载方案，最主要的有 CommonJS 和 AMD 两种。前者用于服务器，后者用于浏览器。ES6 在语言标准的层面上，实现了模块功能，而且实现得相当简单，完全可以取代 CommonJS 和 AMD 规范，成为浏览器和服务器通用的模块解决方案。

而我们这里要说的是在使用`import`语法引用模块时，如何正确使用`{}`。

假如有一个`B.js`，想要通过`import`语法引用模块`A.js`，那么可以这么写：

```
// B.js
import A from './A'
```

而上面的代码生效的前提是，只有在如下`A.js`中有**默认导出**的`export default`语法时才会生效。也就是：

```
// A.js
export default 42
```

在这种不使用`{}`来引用模块的情况下，`import`模块时的命名是随意的，即如下三种引用命名都是正确的：

```
// B.js
import A from './A'
import MyA from './A'
import Something from './A'
```

因为它总是会解析到`A.js`中默认的`export default`。

而下面是使用了花括号命名的方式`{A}`来导入`A.js`：

```
import { A } from './A'
```

上面代码生效的前提是，只有在模块`A.js`中有如下**命名导出**为`A`的`export name`的代码，也就是：

```
export const A = 42
```

而且，在明确声明了命名导出后，那么在另一个`js`中使用`{}`引用模块时，`import`时的模块命名是有意义的，如下：

```
// B.js
import { A } from './A'                 // 正确，因为A.js中有命名为A的export
import { myA } from './A'               // 错误！因为A.js中没有命名为myA的export
import { Something } from './A'         // 错误！因为A.js中没有命名为Something的export
```

要想上述代码正确执行，你需要明确声明每一个命名导出：

```
// A.js
export const A = 42
export const myA = 43
export const Something = 44
```

ps: 一个模块中只能有一个默认导出export default，但是却可以有任意命名导出（0个、1个、多个），你也可以如下，一次性将他们导入：

```
// B.js
import A, { myA, Something } from './A'
```

这里我们使用导入默认导出`A`，以及命名导出`myA`和`Something`。

我们甚至可以在导入的时候重命名导入：

```
import X, { myA as myX, Something as XSomething } from './A'
```

总结：模块的**默认导出**通常是用在你期望该从模块中获取到任何想要的内容；而**命名导出**则是用于一些有用的公共方法，但是这些方法并不总是必要的。