该方法与pytorch可以联合使用，其使用字符串（equation）的形式来表达对张量的改变。使得维度转换等更加容易理解，安装使用如下代码进行：

conda install einops

同时在pytorch中集成了爱因斯坦求和 （einsum）的API，其可以实现向量内积，向量外积，矩阵乘法，转置和张量收缩（tensor contraction）等张量操作。equation包含两部分：自由索引（Free indices）和求和索引（Summation indices）。

* 自由索引，出现在箭头右边的索引；
* 求和索引，只出现在箭头左边的索引，表示中间计算结果需要这个维度上求和之后才能得到输出。

**其共有三条基本规则：**

* 规则一，equation 箭头左边，在不同输入之间重复出现的索引表示，把输入张量沿着该维度做乘法操作；
* 规则二，只出现在 equation 箭头左边的索引，表示中间计算结果需要在这个维度上求和，也就是上面提到的求和索引；
* 规则三，equation 箭头右边的索引顺序可以是任意的，用户只需要定义好索引的顺序，转置操作会在 einsum 内部完成。

**特殊规则有两条：**

* equation 可以不写包括箭头在内的右边部分，那么在这种情况下，输出张量的维度会根据默认规则推导。就是把输入中只出现一次的索引取出来，然后按字母表顺序排列；
* equation 中支持 "..." 省略号，用于表示用户并不关心的索引（einops中也可以这样使用）。

einops与einsum的使用案例如下：

* **import** torch
* **from** einops **import** rearrange, reduce, repeat
* x = torch.ones([3, 4, 6, 6])
* y = torch.randn([5, 4, 6, 7])
* a = torch.arange(5)
* # @转置
* out2 = rearrange(x, 'n c h w -> n c w h')
* # @变形
* out2 = rearrange(x, 'n c h w -> (n c) w h')
* out2 = rearrange(out2, '(n c) h w -> n c w h', n=3)
* # @Channel shuffle
* out1 = rearrange(x, 'b (c1 c2) h w -> b (c2 c1) h w', c1=groups)
* # @堆叠张量
* tensor\_list = [x, x, x]
* out1 = rearrange(tensor\_list, 'a n c h w ->a n c h w')
* # @扩维
* out1 = rearrange(x, 'n c h w -> n c h w 1')
* # @复制
* out2 = repeat(out1, 'n c h w 1 -> n c h w 3')
* out2 = repeat(out2, 'n c h w a -> n (2 c) h w a')
* # @求均值、求和、最大最小、连乘
* out1 = reduce(x, 'n c h w -> n c', 'mean')  # mean, min, max, sum, prod
* out2 = reduce(x, 'n c h w -> n c 1 1', 'max')
* # @image2patch
* out1 = rearrange(x, 'n c (h1 p1) (w1 p2) -> n (h1 w1) (p1 p2 c)', p1=3, p2=3)
* # @对角线求和(trace)
* out1 = torch.einsum('ncii -> nc', x)
* # @矩阵转置
* out1 = torch.einsum('n...j -> j...n', x)
* # @求和
* out1 = torch.einsum('nchw -> nc', x)  # 不能保持原有维度
* # @矩阵向量乘法
* out1 = torch.einsum('ncwh, NchW -> nNwW', x, y)
* # @矩阵元素对应相乘并求和
* out1 = torch.einsum('ncwh, ncwh -> ', x, x)
* # @向量外积(列向量\*行向量)
* out1 = torch.einsum('i,j -> ij', a, a)