**c++11 vector中push\_back和emplace\_back的区别**

区别

在引入右值引用，转移构造函数，转移复制运算符之前，通常使用push\_back()向容器中加入一个右值元素（临时对象）的时候，首先会调用构造函数构造这个临时对象，然后需要调用拷贝构造函数将这个临时对象放入容器中。原来的临时变量释放。这样造成的问题是临时变量申请的资源就浪费。

引入了右值引用，转移构造函数（请看这里）后，push\_back()右值时就会调用构造函数和转移构造函数。在这上面有进一步优化的空间就是使用emplace\_back，在容器尾部添加一个元素，这个元素原地构造，不需要触发拷贝构造和转移构造。而且调用形式更加简洁，直接根据参数初始化临时对象的成员。

总结：

push\_back()右值时就会调用构造函数和转移构造函数

emplace\_back()函数向容器中中加入临时对象， 临时对象原地构造，没有赋值或移动的操作。

emplace\_back()函数要比push\_back()函数要快一倍

**C++ 如何快速清空vector以及释放vector内存？**

1、为什么需要主动释放vector内存

vector其中一个特点：内存空间只会增长，不会减小，援引C++ Primer：为了支持快速的随机访问，vector容器的元素以连续方式存放，每一个元素都紧挨着前一个元素存储。设想一下，当vector添加一个元素时，为了满足连续存放这个特性，都需要重新分配空间、拷贝元素、撤销旧空间，这样性能难以接受。因此STL实现者在对vector进行内存分配时，其实际分配的容量要比当前所需的空间多一些。就是说，vector容器预留了一些额外的存储区，用于存放新添加的元素，这样就不必为每个新元素重新分配整个容器的内存空间。

在调用push\_back时，每次执行push\_back操作，相当于底层的数组实现要重新分配大小；这种实现体现到vector实现就是每当push\_back一个元素,都要重新分配一个大一个元素的存储，然后将原来的元素拷贝到新的存储，之后在拷贝push\_back的元素，最后要析构原有的vector并释放原有的内存。

2、怎么释放vector的内存

A、对于数据量不大的vector，没有必要自己主动释放vector，一切都交给操作系统。

B、但是对于大量数据的vector，在vector里面的数据被删除后，主动去释放vector的内存就变得很有必要了！

由于vector的内存占用空间只增不减，比如你首先分配了10000个字节，然后erase掉后面9999个，留下一个有效元素，但是内存占用仍为10000个。所有内存空间是在vector析构时候才能被系统回收。empty()用来检测容器是否为空的，clear()可以清空所有元素。但是即使clear()，vector所占用的内存空间依然如故，无法保证内存的回收。如果需要空间动态缩小，可以考虑使用deque。如果vector，可以用swap()来帮助你释放内存。

**C++ list, vector, map, set 区别与用法比较**

List封装了链表,Vector封装了数组, list和vector得最主要的区别在于vector使用连续内存存储的，他支持[]运算符，而list是以链表形式实现的，不支持[]。

Vector对于随机访问的速度很快，但是对于插入尤其是在头部插入元素速度很慢，在尾部插入速度很快。List对于随机访问速度慢得多，因为可能要遍历整个链表才能做到，但是对于插入就快的多了，不需要拷贝和移动数据，只需要改变指针的指向就可以了。另外对于新添加的元素，Vector有一套算法，而List可以任意加入。

Map,Set属于标准关联容器，使用了非常高效的平衡检索二叉树：红黑树，他的插入删除效率比其他序列容器高是因为不需要做内存拷贝和内存移动，而直接替换指向节点的指针即可。

Set和Vector的区别在于Set不包含重复的数据。Set和Map的区别在于Set只含有Key，而Map有一个Key和Key所对应的Value两个元素。

Map和Hash\_Map的区别是Hash\_Map使用了Hash算法来加快查找过程，但是需要更多的内存来存放这些Hash桶元素，因此可以算得上是采用空间来换取时间策略。

用两个3\*3代替5\*5卷积的优点

1、对于5\*5和两个3\*3的计算量，我们可以比较一下。

2、我们假设输入图像大小是5\*5\*1,最终都需要将其变成1\*1\*1.

3、那么对于5\*5的核（暂时用1个），我们的总参数是25，总的乘法计算数为1\*5\*5\*1=25;

4、而对于3\*3的核（用1个），我们总参数是2\*3\*3=18.总的乘法计算数：

4.1 5\*5\*1->3\*3\*1： 乘法计算数目为 1\*3\*3\*3\*3\*1=81次。

4.2 3\*3\*1->1\*1\*1： 乘法计算数目为 1\*3\*3\*1\*1\*1=9次

4.3 总共是90次（超多）



但是，计算机读内存的速度比计算乘法的速度慢多了，所以我们宁愿多算几次，也不要多读一点内存数据。

因此，虽然3\*3的卷积核计算量较大，但是参数数目较5\*5少很多，在用3\*3卷积核参与卷积运算时计算机的处理速度会快很多。该优化方法在早期的VGG网络中很常见。而且，使用2个3\*3替换一个5\*5使得网络的深度（层数）增加，非线性表达特征的能力就会增强。

**1\*1卷积的主要作用有以下几点：**

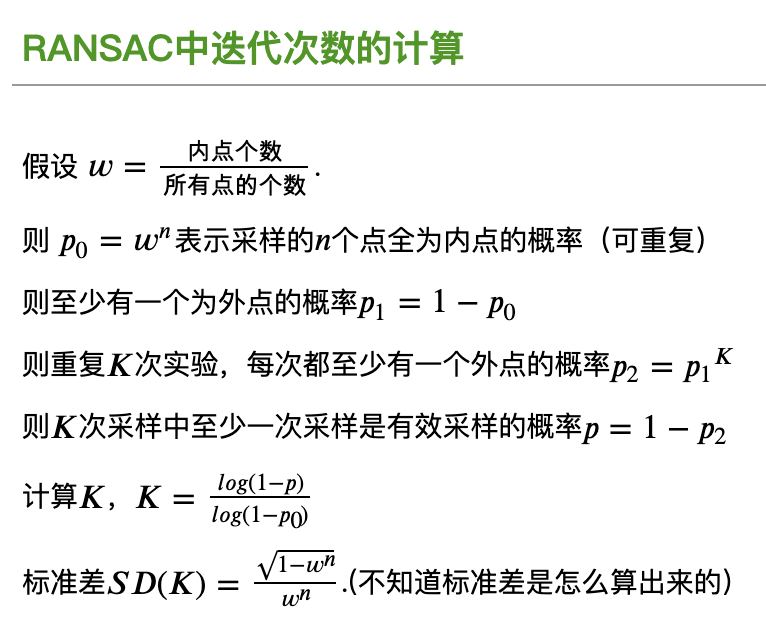
1、降维（ dimension reductionality ）（减少参数）。比如，一张500 \* 500且厚度depth为100 的图片在20个filter上做1\*1的卷积，那么结果的大小为500\*500\*20。

2、升维（用最少的参数拓宽网络channal）

3、加入非线性。卷积层之后经过激励层，1\*1的卷积在前一层的学习表示上添加了非线性激励（ non-linear activation ），提升网络的表达能力；

4、跨通道信息交互（channal 的变换）例子：使用1\*1卷积核，实现降维和升维的操作其实就是channel间信息的线性组合变化，3\*3，64channels的卷积核后面添加一个1\*1，28channels的卷积核，就变成了3\*3，28channels的卷积核，原来的64个channels就可以理解为跨通道线性组合变成了28channels，这就是通道间的信息交互。注意：只是在channel维度上做线性组合，W和H上是共享权值的sliding window

5、从fully-connected layers的角度来理解1\*1卷积核，将其看成全连接层



**vector存储是连续的吗**

https://www.codenong.com/19876069/