另一个体现不重复存储的地方是，有时候可能有多个逻辑上不同的版本标记，都指向了同一个物理上的制品版本。比如，初始版本用 123456 的构建顺序号来标记，后面被选中送去测试时，又被标记为 22-05-02 版，即 2022 年第 5 个迭代的第二次送测的版本。如果其通过测试准备发布，则又被标记为 22-05，也就是 2022 年第 5 个迭代的发布版本。尽管 123456、22-05-02 和 22-05 是不同的三个版本名称，但其实它们都指向同一个物理上的实体，故不需要存储三份。

2）控制制品的尺寸

制品的尺寸不宜过大，如果太大，会产生几方面的不利影响。

（1）传输制品的时间较长。

（2）在制品库存储制品的空间耗费较多。

（3）制品运行时耗费的内存等资源较多。

（4）在生产环境中，随着用户使用量的增加，可能只是一个制品中的部分功能需要扩容，但是不得不把此制品作为整体扩容，造成浪费。

从本质上说，降低制品的尺寸还是要做好软件架构工作，要追求细粒度、低耦合、可复用的软件架构，尽量采用微服务而不是大型单体应用的方式。

另外，把一个制品拆为部署在同一台机器上的若干个制品，对减少传输制品时长、降低存储空间耗费、降低运行时的内存耗费都有帮助。一是因为每次更新时可能只需要更新部分上层制品；二是因为部分底层制品可能供若干个上层制品复用，其中，典型的是以动态库形式存在的底层制品。Docker 镜像的内建分层机制在本质上也具有类似作用。

此外，还可以排查构建、打包、制作制品时有没有混入无用的内容，如果有尽量想办法去掉。比如，在构建程序时，作为输入的静态库包含很多方法和函数，它们在程序运行时肯定不会全部被调用。那么，如何避免把不被调用的方法和函数放进制品呢？

3）提高存取速度

制品单体文件体积大和并发下载量高是导致制品库性能差的两个因素，磁盘 IO 和网络 IO 的瓶颈制约着我们在开发过程中流畅地使用不同类型的制品，这一问题在 Docker 镜像得到充分应用的开发场景下被无限放大，为了提高制品下载和上传的速度，可以考虑如下措施。

（1）更好的硬件，特别是更快的网络传输速度。

（2）在构建服务器上构建依赖内容的缓存。

（3）在运行服务器上构建部署内容的缓存，这在生产环境版本回滚时特别有用。