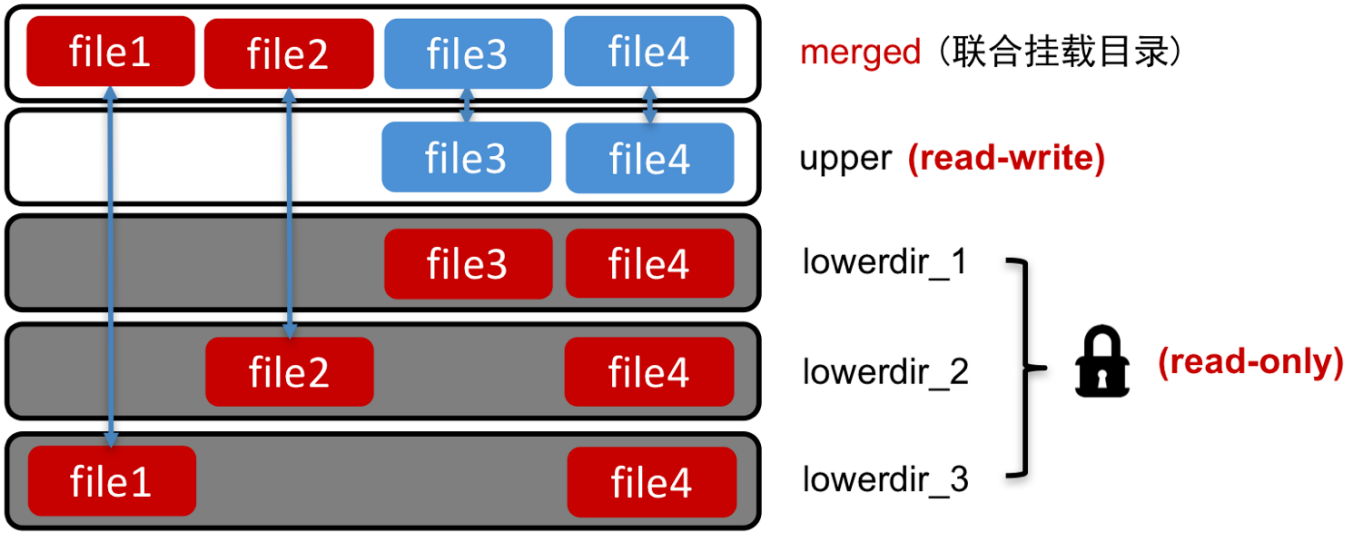
Docker 入门教程：Docker 基础技术 Union File System

我们在前面 《Docker 入门教程：镜像分层》 文章中介绍了 Docker 为什么构建速度非常快，其原因就是采用了镜像分层，镜像分层底层采用的技术就是本文要介绍的 Union File System。Linux 支持多种 Union File System，比如 aufs、OverlayFS、ZFS 等。



如果想及时了
解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众帐号：iteblog_hadoop

aufs & OverlayFS

下面我们简单介绍一下比较常见的两种 Union File System：aufs 和 OverlayFS。

aufs

aufs 的全称是 Advanced Multi-layered unification filesystem，实现了 Linux 文件系统的 union mount 功能，就是把不同物理位置的目录合并挂载到同一个目录中。aufs 是由 Junjiro Okajima 在 2006 年开发的，他完全把 UnionFS 重写了，主要目的就是提升可靠性和性能，并且引入了一些新的概念，比如可写分支的负载均衡。不过 aufs 并没有合并到 Linux
内核，因为它的代码被批评为“密集”、“不可读”以及“未注释”（dense, unreadable, and uncommented）。经过几次努力作者想把 aufs 合并到 Linux
内核，但是都没有成功，最后放弃了。不过 aufs 被 Debian "jessie"、 Ubuntu 16.04 以及 Gentoo Linux LiveDVD 合并进去了。（参见 维基百科 Aufs）。

OverlayFS

OverlayFS 也实现了 Linux 文件系统的 union mount 功能，最初由 Miklos Szeredi 于 2010

年提出，并于 2014 合并到 Linux 3.18 内核中。（参见 维基百科 OverlayFS）。

实战

好了，前面扯了这么久，很多人肯定还是不知道 union mount
到底是什么东西，下面我们来实战一把，这样印象会更加深入。网上有很多 aufs union mount
的例子，所以这篇文章我们来介绍 OverlayFS union mount
使用。在介绍之前，我们先在电脑里面创建如下的目录结构：

```
[root@iteblog.com ~/test]$ tree
.
├── iteblog_1
│   ├── a
│   ├── b
│   ├── c
│   └── sub_iteblog
│       ├── d
│       └── f
├── iteblog_2
│   ├── c
│   ├── e
│   └── sub_iteblog
│       ├── d
│       └── g
├── merge
└── work

6 directories, 9 files
```

上面 a、b、d、f、c、e、d、g 都属于文本文件，其他都属于目录。现在我们使用 OverlayFS
mount 把 iteblog_1 和 iteblog_2 目录合并挂载到 merge 目录中，如下：

```
[root@iteblog.com ~/test]$ sudo mount -t overlay overlay -o lowerdir=iteblog_1,upperdir=itebl
og_2,workdir=work merge
```

在 OverlayFS 中，存在 Lower 和 Upper 的概念，overlay
其实是“覆盖...上面”的意思，表示一个文件系统覆盖在另一个文件系统上面，也就是将 upperdir
参数指定的目录覆盖到 lowerdir 参数指定的目录之上，并且挂载到 merge 目录里面，workdir
类似于工作目录，其中 upperdir 和 workdir 参数是可选的。lowerdir

参数支持指定多个目录，目录之前使用 : 分割。

注意：如果 lowerdir
参数指定了多个目录，那么目录的覆盖顺序是从左到右进行的，比如下面命令：

```
# mount -t overlay overlay -o lowerdir=/lower1:/lower2:/lower3,upperdir=/upper,workdir=/work /merged
```

那么覆盖的顺序如下：

/upper
/lower1
/lower2
/lower3

执行完，我们使用 df -h 命令可以看到 overlay 文件系统已经挂载到 /root/test/merge 目录

```
[root@iteblog.com ~/test]$ df -h
文件系统        容量  已用  可用 已用% 挂载点
/dev/vda1        50G   31G   17G   65% /
devtmpfs        911M     0  911M    0% /dev
tmpfs           920M     0  920M    0% /dev/shm
tmpfs           920M  388K  920M    1% /run
tmpfs           920M     0  920M    0% /sys/fs/cgroup
tmpfs           184M     0  184M    0% /run/user/0
overlay          50G   31G   17G   65% /root/test/merge
```

我们再来看看 merge 目录的结构：

```
[root@iteblog.com ~/test]$ tree merge
merge
├── a
├── b
├── c
├── e
└── sub_iteblog
```

```
├── d
├── f
└── g
```

1 directory, 7 files

overlay 文件和目录合并策略

从上面可以看出 merge 目录里面的东西就是 iteblog_1 和 iteblog_2
两个目录合并的结果，那么合并的规则是怎么样的呢？

- 文件名及目录不相同，则 iteblog_1 及 iteblog_2 目录中的文件及目录按原结构都融入到 merge 目录中；
- 文件名相同，只显示 iteblog_2 层的文件。如上图在 iteblog_1 和 iteblog_2 目录下都存在名为 c 的文件，合并到 merge 目录时，则只显示 iteblog_2 的，而 iteblog_1 中的文件隐藏；
- 目录名相同，则目录里面的内容会进行合并。如上图在iteblog_1 和 iteblog_2 目录下都有 sub_iteblog 目录，将目录及目录下的所有文件合并到 merge 的 sub_iteblog 目录，目录内如有文件名相同，则同样只显示 iteblog_2 的。

overlay 文件和目录修改策略

如果我们对合并后的文件进行修改会发生什么呢？比如我们修改 merge/c
文件，那么最终修改的是 iteblog_1/c 文件还是 iteblog_2/c 文件？答案是修改 iteblog_2/c 文件：

```
[root@iteblog.com ~/test]$ echo merge >> merge/c
[root@iteblog.com ~/test]$ cat iteblog_2/c
merge
[root@iteblog.com ~/test]$ cat iteblog_1/c
iteblog_1
```

如果我们修改 merge/e 这个文件会发生什么呢？

```
[root@iteblog.com ~/test]$ echo merge_e >> merge/e
[root@iteblog.com ~/test]$ cat iteblog_2/e
merge_e
[root@iteblog.com ~/test]$ cat iteblog_1/e
cat: iteblog_1/e: 没有那个文件或目录
```

可以看出，直接修改了 iteblog_2/e 文件了。那如果我们修改 merge/a 会发生什么？

```
[root@iteblog.com ~/test]$ echo merge_a >> merge/a
[root@iteblog.com ~/test]$ cat iteblog_1/a
[root@iteblog.com ~/test]$ cat iteblog_2/a
merge_a
[root@iteblog.com ~/test]$ ll iteblog_2/
总用量 12
-rw-r--r-- 1 root root    8 2月   5 23:19 a
c--------- 1 root root 0, 0 2月   5 23:10 b
-rw-r--r-- 1 root root    6 2月   5 23:14 c
-rw-r--r-- 1 root root    8 2月   5 23:18 e
c--------- 1 root root 0, 0 2月   5 23:10 sub_iteblog
```

可以看出，a 本来是在 iteblog_1/a 目录的，但是执行完上面命令 a 文件居然出现在 iteblog_2 目录下，而且我们修改了 iteblog_2/a 文件。这是因为对只在 iteblog_1 中存在的文件，则会做一个 copy_up 的操作，先从 iteblog_1 将文件拷贝一份到 iteblog_2，然后写拷贝到 iteblog_2 的文件，而 iteblog_1 还是原文件，并不发生修改。

在 overlay 中，upperdir 参数指定的文件系统通常是可写的；lowerdir 参数指定的文件系统通常是只读。也就是说，当我们对 overlay 文件系统做任何的变更，都只会修改 upperdir 文件系统中的文件，lowerdir 参数指定的文件不会变化，具体如下：

写

- 如果需要写的文件只在 upperdir 参数指定的文件里面，则直接写 upperdir 参数文件夹对应的文件；
- 如果需要写的文件同时在 lowerdir 和 upperdir 参数指定的文件夹里面，则直接写 upperdir 参数文件夹对应的文件；
- 如果需要写的文件只在 lowerdir 参数指定的文件里面，则会做一个 copy_up 的操作，先从 lowerdir 将文件拷贝一份到 upperdir，然后写拷贝到 upperdir 的文件，而 lowerdir 还是原文件，并不发生修改。
- 如果是创建文件/目录，效果等同于在 upperdir 层直接进行创建。

读

- 需要读取的文件只存在 lowerdir 层时，直接从 lowerdir 读；
- 需要读取的文件只存在 upperdir 层时，直接从 upperdir 读；
- 需要读取的文件同时存在 upperdir 和 lowerdir 层时，直接从 upperdir 读；

删除

- 需要删除的文件只存在 upperdir 层时，直接从 upperdir 删除；
- 需要删除的文件只存在 lowerdir 层时，则在 upperdir 层创建同名的字符设备（使用 ls -l 命令列出的文件以 c 字母开头），同时 lowerdir 层的文件不做修改；
- 需要删除的文件同时存在 lowerdir 和 upperdir 层时，则 upperdir 层的文件被删除，同时在 upperdir 层创建同名的字符设备（使用 ls -l 命令列出的文件以 c 字母开头），而 lowerdir 层的文件不做修改；

从上面的描述可以看出，因为 lowerdir 是只读层，所有的操作都是对在 upperdir 层做的。理解这个以及前面介绍的镜像分层机制，对我们理解 Docker 的工作原理很重要的。好了，今天就到这里了，我们下一篇文章来介绍 Union File System 和 Docker 的关系。