Glibc库升级2.14

<https://baike.baidu.com/item/glibc/10058561?fr=aladdin>

glibc是[GNU](https://baike.baidu.com/item/GNU" \t "https://baike.baidu.com/item/glibc/_blank)发布的libc库，即c[运行库](https://baike.baidu.com/item/%E8%BF%90%E8%A1%8C%E5%BA%93/5587282" \t "https://baike.baidu.com/item/glibc/_blank)。glibc是[linux系统](https://baike.baidu.com/item/linux%E7%B3%BB%E7%BB%9F/1732935" \t "https://baike.baidu.com/item/glibc/_blank)中最底层的[api](https://baike.baidu.com/item/api/10154" \t "https://baike.baidu.com/item/glibc/_blank)，几乎其它任何运行库都会依赖于glibc。glibc除了封装[linux](https://baike.baidu.com/item/linux" \t "https://baike.baidu.com/item/glibc/_blank)操作系统所提供的[系统服务](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E6%9C%8D%E5%8A%A1/11027121" \t "https://baike.baidu.com/item/glibc/_blank)外，它本身也提供了许多其它一些必要功能服务的实现。由于 glibc 囊括了几乎所有的 [UNIX](https://baike.baidu.com/item/UNIX" \t "https://baike.baidu.com/item/glibc/_blank) 通行的标准，可以想见其内容包罗万象。而就像其他的 UNIX 系统一样，其内含的档案群分散于系统的树状[目录结构](https://baike.baidu.com/item/%E7%9B%AE%E5%BD%95%E7%BB%93%E6%9E%84/4348167" \t "https://baike.baidu.com/item/glibc/_blank)中，像一个支架一般撑起整个操作系统。在 GNU/Linux 系统中，其C函式库发展史点出了GNU/Linux 演进的几个重要里程碑，用 glibc 作为系统的C函式库，是GNU/Linux演进的一个重要里程碑。

glibc是一种按照[LGPL](https://baike.baidu.com/item/LGPL" \t "https://baike.baidu.com/item/glibc/_blank)许可协议发布的，自由的，公开[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81" \t "https://baike.baidu.com/item/glibc/_blank)的，方便从网络下载的C的编译程序。GNU C运行期库，是一种C函数库，是程序运行时使用到的一些[API](https://baike.baidu.com/item/API" \t "https://baike.baidu.com/item/glibc/_blank)集合，它们一般是已预先[编译](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91" \t "https://baike.baidu.com/item/glibc/_blank)好，以[二进制](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6" \t "https://baike.baidu.com/item/glibc/_blank)代码形式存在[Linux](https://baike.baidu.com/item/Linux" \t "https://baike.baidu.com/item/glibc/_blank)类系统中，GNU C运行期库通常作为GNU C编译程序的一个部分发布。

glibc最初是[自由软件基金会](https://baike.baidu.com/item/%E8%87%AA%E7%94%B1%E8%BD%AF%E4%BB%B6%E5%9F%BA%E9%87%91%E4%BC%9A" \t "https://baike.baidu.com/item/glibc/_blank)（FSF）为其[GNU](https://baike.baidu.com/item/GNU" \t "https://baike.baidu.com/item/glibc/_blank)操作系统所写，但当前最主要的应用是配合[Linux内核](https://baike.baidu.com/item/Linux%E5%86%85%E6%A0%B8" \t "https://baike.baidu.com/item/glibc/_blank)，成为GNU/Linux操作系统一个重要的组成部分。

## 功能实现

主要的如下：

（1）string，字符串处理

（2）signal，信号处理

（3）dlfcn，管理共享库的动态加载

（4）direct，[文件目录](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E7%9B%AE%E5%BD%95" \t "https://baike.baidu.com/item/glibc/_blank)操作

（5）elf，共享库的动态加载器，也即interpreter

（6）iconv，不同[字符集](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6%E9%9B%86" \t "https://baike.baidu.com/item/glibc/_blank)的编码转换

（7）inet，socket接口的实现

（8）intl，国际化，也即gettext的实现

（9）io

（10）[linux](https://baike.baidu.com/item/linux" \t "https://baike.baidu.com/item/glibc/_blank)threads

（11）locale，本地化

（12）login，[虚拟终端](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E7%BB%88%E7%AB%AF" \t "https://baike.baidu.com/item/glibc/_blank)设备的管理，及系统的安全访问

（13）malloc，动态内存的分配与管理

（14）nis

（15）stdlib，其它基本功能

字集转换模块与区域化资料库:

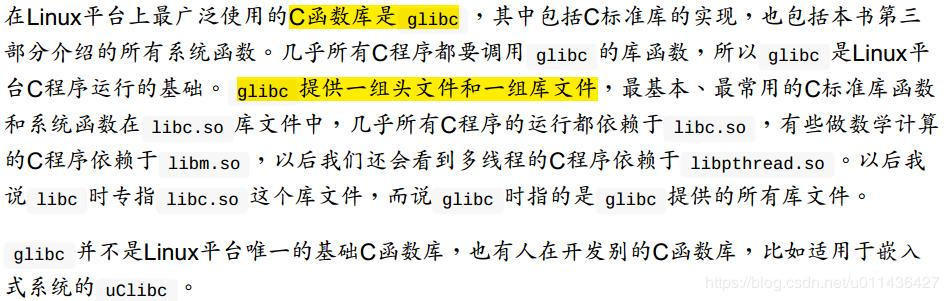
这些是与程式国际化与本土化相关的部分，主要可分成四大块: /usr/lib/gconv/ 内含大量的字集转换模块，大部分是各种字集及编码方式与系统的基底字集之间的 转换。第二块是 /usr/lib/locale，内含以系统基底字集写成的区域化资料库 (locale)，像是 LC\_CTYPE、LC\_TIME .... 等等。第三块是

/usr/share/locale/，内含可跨平台使用的区域化资料，主要是各应用程式的信息翻译部分。而最后一块是 /usr/share/i18n/，其内容是各区域化资料库的原始码，以及系统支持的[内码](https://baike.baidu.com/item/%E5%86%85%E7%A0%81" \t "https://baike.baidu.com/item/glibc/_blank)对应表 .... 等等。

分享函式库群:

这是 glibc 的主体，分布 /lib 与 /usr/lib 中，包括 libc 标准 C 函式库、libm 数学函式库、libcrypt 加密与编码函式库、libdb 资料库函式库、libpthread 行程[多执行绪](https://baike.baidu.com/item/%E5%A4%9A%E6%89%A7%E8%A1%8C%E7%BB%AA" \t "https://baike.baidu.com/item/glibc/_blank)函式库、libnss 网路服务函式库 .... 等等。这些都是可分享函式库，档名都以 .so 做结尾。其中，/lib/ld\*.so 是程式与函式库连结的工具。有的用于程式编译时将程式与函式库内的函式物件连结，在只支持[静态连结](https://baike.baidu.com/item/%E9%9D%99%E6%80%81%E8%BF%9E%E7%BB%93" \t "https://baike.baidu.com/item/glibc/_blank)的系统中，此连结方式就是直接将所需的物件自函式库中抽出来与程式的可[执行档](https://baike.baidu.com/item/%E6%89%A7%E8%A1%8C%E6%A1%A3" \t "https://baike.baidu.com/item/glibc/_blank)相连，而在支持可分享函式库的系统中，在程式编译时期的连结只是在执行档中纪录了那些函式物件是存在那个函式库档案中，等该程式开始执行时，则由另一个负责动态连结的 ld\*.so 将所需的函式库连结好并执行。

要在一个平台上支持C语言，不仅要实现C编译器，还要实现C标准库，但是很多单片机的C语言开发工具中只有C编译器而没有完整的C标准库。



现在centos6.8-x64系统里的c标准库已经成了glibc，glibc取代了libc,c标准库的位置在/lib64/libc.so.6，glibc是linux下面c标准库的实现，即GNU C Library。glibc本身是GNU旗下的C标准库，后来逐渐成为了Linux的标准c库，而Linux下原来的标准c库Linux libc逐渐不再被维护。Linux下面的标准c库不仅有这一个，如uclibc（https://www.uclibc.org/）、klibc，以及上面被提到的Linux libc，但是glibc无疑是用得最多的。glibc在/lib目录下的.so文件为libc.so.6。

 GNU C 函数库是一种类似于第三方插件的东西。由于 Linux 是用 C 语言写的，所以 Linux 的一些操作是用 C 语言实现的，因此，GUN 组织开发了一个 C 语言的库 以便让我们更好的利用 C 语言开发基于 Linux 操作系统的程序。不过现在的不同的 Linux 的发行版本对这两个函数库有不同的处理方法，有的可能已经集成在同一个库里了。

默认的Centos6.5 glibc版本最高为2.12, 而在进行开发时项目所依赖的包往往需要更高版本的glibc库支持, 因此在不升级系统的前提下, 需要主动更新系统glibc库. 一般遇到错误libc.so.6: version GLIBC\_2.14 not found时表示需要对glibc进行升级了.

查看系统glibc库版本可使用如下命令:

第一、查看系统glibc版本库

strings /lib64/libc.so.6 |grep GLIBC\_

[root@218 ~]# strings /lib64/libc.so.6 |grep GLIBC\_

GLIBC\_2.2.5

GLIBC\_2.2.6

GLIBC\_2.3

GLIBC\_2.3.2

GLIBC\_2.3.3

GLIBC\_2.3.4

GLIBC\_2.4

GLIBC\_2.5

GLIBC\_2.6

GLIBC\_2.7

GLIBC\_2.8

GLIBC\_2.9

GLIBC\_2.10

GLIBC\_2.11

GLIBC\_2.12

GLIBC\_PRIVATE

第二、下载和安装glibc-2.14

wget http://ftp.gnu.org/gnu/glibc/glibc-2.14.tar.gz

tar -xzvf glibc-2.14.tar.gz

cd glibc-2.14

mkdir build

cd build

../configure --prefix=/opt/glibc-2.14

make && make install

第三、创建软连接：

ln -sf /opt/glibc-2.15/lib/libc-2.15.so /lib64/libc.so.6(f为强制替换)

注：网上很多方法都是先删除 /lib64/libc.so.6,再创建软连接ln -s，但是删除libc.so.6后，ln -s创建软连接命令就会失败，连最简单的命令都使用不了，报 command not found，此处是个大坑，删除需谨慎，补救办法是：LD\_PRELOAD=/opt/glibc-2.14/lib/libc-2.14.so ln -s /opt/glibc-2.14/lib/libc-2.14.so /lib64/libc.so.6；或者还原成旧版本。

升级失败导致命令无法使用（如：Segmentation Fault），使用如下命令回滚

LD\_PRELOAD=/lib64/libc-2.12.so ln -s /lib64/libc-2.12.so /lib64/libc.so.6

第四、查看当前新的glibc版本库

[root@218 ~]# strings /lib64/libc.so.6 |grep GLIBC\_

GLIBC\_2.2.5

GLIBC\_2.2.6

GLIBC\_2.3

GLIBC\_2.3.2

GLIBC\_2.3.3

GLIBC\_2.3.4

GLIBC\_2.4

GLIBC\_2.5

GLIBC\_2.6

GLIBC\_2.7

GLIBC\_2.8

GLIBC\_2.9

GLIBC\_2.10

GLIBC\_2.11

GLIBC\_2.12

GLIBC\_2.13

GLIBC\_2.14

GLIBC\_PRIVATE

升级后问题：

中文显示乱码：

locale: Cannot **set** LC\_CTYPE **to** **default** locale: **No** such **file** **or** **directory**

locale: Cannot **set** LC\_ALL **to** **default** locale: **No** such **file** **or** **directory**

以上错误不仅出现在手动运行locale命令时，而且每次ssh远程登录进shell时都会出现。

原因：我们重新安装了一个新的glibc库，而不是覆盖升级，也就是说系统中目前至少有两个locale命令，查看新安装的glibc2.14的locale

方法一：  
*/opt/glibc-2.14/bin/locale -a*

C

POSIX

找到locale读取的文件位置：

[\*\*\*\*@\*\*\*\*\*\* bin]**# strings /opt/glibc-2.14/bin/locale|grep locale**

setlocale

/locale.

locale (GNU %s) %s

/opt/glibc-2.14/lib/locale

/opt/glibc-2.14/share/locale

all-locales

/opt/glibc-2.14/lib/locale/locale-archive

locale: %-15.15s archive: /opt/glibc-2.14/lib/locale/locale-archive

locale: %-15.15s directory: %.\*s

Cannot set LC\_CTYPE to **default** locale

Cannot set LC\_MESSAGES to **default** locale

Cannot set LC\_COLLATE to **default** locale

Cannot set LC\_ALL to **default** locale

Write names of available locales

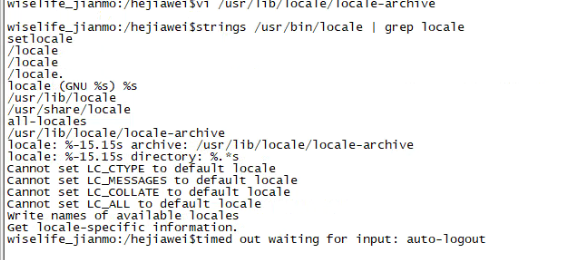
Get locale-specific information.

果然上面出现了一个路径/opt/glibc-2.14/lib/locale/locale-archive，通过名字很容易猜出这个很可能就是locale命令去读取所有被支持的locale目录（/或归档文件）位置。

这个locale只支持C和POSIX，而没有en\_US.utf8，所以如果我们将locale配置成en\_US.utf8，自然会报No such file or directory错误。

根据上述定位我们得到locale的配置文件为： /usr/local/glibc-2.15/lib/locale/locale-archive，不出所料这个文档并不存在，正是问题关键所在，所以我们只需要设定好这个文件就行啦，怎么搞？方法就是再次使用 strings 命令定位出 /usr/bin/locale 的配置文档，然后复制过去就OK啦：

**strings /usr/bin/locale | grep locale**



路径在/usr/lib/locale/locale-archive

1. mkdir -pv /usr/local/glibc-2.15/lib/locale/
2. cp /usr/lib/locale/locale-archive /usr/local/glibc-2.15/lib/locale/
3. 方法二：其实安装包INSTALL文档说的很清楚，自己去翻翻吧，进入编译安装包，一条命令搞定：
4. cd /hejiawei/glibc-2.14/build
5. make localedata/install-locales