لوله‌کشی به‌روش جدید

شادی اسدی

علم کامپیوتر همواره المان‌های طراحی خود را از زندگی واقعی وام گرفته است و تلاش دارد خود را به زندگی واقعی نزدیک‌تر کند. خط لوله (Pipeline) نقش مهمی را در نرم‌افزار بازی می‌کند. به‌دلیل این‌که همه ما سیستم لوله‌کشی را می‌شناسیم و با آن آشنایی داریم، مفهوم نرم‌افزاری آن را بهتر درک خواهیم کرد.

همانطور که گفتیم لوله‌کشی در نرم‌افزار، کاربردهای زیادی دارد، به‌عنوان مثال از لوله‌کشی در مورد الگوریتم‌های طولانی علمی و ریاضی که هر مرحله باید پشت مرحله بعدی انجام شود، خواندن و نوشتن از روی دیسک، استفاده از بافر ورودی/خروجی، و حتی پردازش‌های گرافیکی می‌توان استفاده کرد.

خط لوله، مجموعه‌ای از تکه لوله‌ها است که به‌روش‌های مختلفی می‌توانند به‌یکدیگر متصل شوند تا یک وظیفه خاص را انجام دهند. در ساده‌ترین شکل ممکن، یک خط لوله، یک سر و یک دم دارد که می‌توانند یکی باشند. (هر چند اگر فقط یک لوله داشته باشیم، هدف استفاده از خط لوله را زیر سوال برده‌ایم.)

در شماره گذشته، در مورد آماده‌سازی برنامه‌های شی‌گرا، برای کار با خط فرمان صحبت کردیم. یکی از قوی‌ترین سیستم‌های عامل دنیا (Unix)، محیط خط فرمان قوی‌ای دارد و همه امور را به‌سادگی با خط فرمان انجام می‌دهد.

این سیستم عامل، که سیستم‌های خط لوله را به‌خوبی پیاده کرده است، مثال خوبی است برای به‌کارگیری خط لوله در خط فرمان و آماده سازی برنامه‌ها برای کار با خط فرمان.

در یونیکس، خروجی هر دستور (دم لوله) به‌عنوان ورودی برای یک دستور یا یک برنامه دیگر به‌کار می‌رود (سر لوله بعدی) و به‌همین ترتیب می‌توان چندین دستور و یا چندین برنامه را یک‌جا با هم اجرا و نتیجه را به‌طور واحد دریافت کرد.

اگر شماره قبلی را به‌خاطر بیاورید، از کلاس خاصی به‌نام فایل استفاده شد که امور مرتبط با کپی و انتقال فایل را به‌آن واگذار کردیم تا براساس اصل استفاده مجدد در شی‌گرایی، بتوانیم درخواست‌های مختلف را با آن پاسخ بدهیم.

در روش یونیکسی آن، که در این شماره می‌خواهیم در موردش صحبت کنیم، خروجی یک دستور را به ورودی یک دستور دیگر می‌دهیم. بگذارید با یک مثال بیشتر در این مورد بحث کنیم:

فرض کنید که قرار است در هنگام اجرای دستور، نیاز به کپی‌کردن فایل باشد ما دیگر به کلاسی برای کپی کردن فایل نیاز نداریم بلکه دستور Copy را فراخوانی می‌کنیم، و متدهای CopyFile و MoveFile که قرار بود در کلاس ثالث دیگری به اسم File بگذاریم، به‌صورت خصوصی در کلاس مربوط به کلاس Copy قرار می‌دهیم.

به این ترتیب کدهای نوشته شده کم خواهد شد و دیگر برای انجام عمل کپی نیاز به فراخوانی متدهای استاتیک کلاس File نداریم و فقط هر بار خواستیم که فایل را کپی یا حتی جابه‌جا کنیم دستور مربوط به Copy را فراخوانی می‌کنیم، حال کدهای نوشته شده در مقاله هفته پیش را با استفاده از این روش باز نویسی می‌کنیم، بدین ترتیب کلاس Copy ما به شکل زیر در می‌آید:

بسیار خوب حالا فرض می‌کنیم که به فراخوانی دستوری برای کپی کردن دایرکتوری نیاز داریم (این دستور در سیستم عامل ویندوز با XCopy شناخته می‌شود و در سیستم عامل لینوکس با استفاده از دستور cp انجام می‌شود. برای توضیحات بیشتر این دستور می‌توانید به لینک زیر مراجعه کنید:

http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/xcopy.mspx?mfr=true

فرض کنید ما می‌خواهیم این دستور را در برنامه خودمان داشته باشیم، باتوجه به روش قبلی که در مقاله هفته پیش نوشته شد، باید یک کلاس مربوط برای Directory تعریف کنیم در این کلاس باید متدی برای کپی کردن Directory بنویسیم و این متد، متد File:Copy را برای همه فایل‌های درون دایرکتوری فراخوانی می‌کند، خوب اگر با استفاده از روش ذکر شده در این مقاله بخواهیم همچین کاری را انجام دهیم باید یک کلاس مربوط به دستور dir را نیز تعریف کنیم، در این کلاس دستور dir را با فرمان s/ اجرا می‌کنیم و سپس خروجی را از دستور dir می‌گیریم. این خروجی شامل آدرس فایل‌ها و زیردایرکتوری‌ها می‌شود.

```
class Copy : public baseCommand{
public:
Copy(int argc, char* argv[]) :
baseCommand(argc,argv){}
void execute(){
if(this->contain("/X") || this-
>contain("/x")){
this->copyFile(this->argv[1],
this->argv[2]);
}
else
this->moveFile(this->argv[1],this-
>argv[2]);
}
private:
void copyFile(const char*
srcFile,const char* dstFile){}
void moveFile(const char*
srcFile,const char* dstFile){}
};
```

فایل‌ها را با استفاده از دستور Copy در جایی که باید کپی شوند، کپی می‌کنیم و دایرکتوری‌ها را نیز با استفاده از دستور mkdir در محل مشخص ایجاد می‌کنیم و این عمل تا زمانی انجام می‌شود که آدرس فایل یا دایرکتوری در خروجی دستور dir موجود است. اگر بخواهیم مطالب گفته شده را بصورت شبه کد بنویسیم به این صورت خواهد بود:

اینجا ما در کلاس xCopy یک شی از کلاس Copy ساخته‌ایم و آنرا اجرا کردیم خوب بگذارید کد را کمی زیباتر کنیم که بحث لوله‌کشی را هم به‌صورت کامل در آن پیاده کرده باشیم. به‌جای استفاده از کلاس Copy می‌توانیم از کلاس Helper که در مقاله هفته پیش توضیح داده بودیم، استفاده کنیم.

همچنین به‌جای ساخت شی از روی کلاس Copy، می‌توانیم بدون ایجاد شی و با اجرای خود برنامه، دستور کپی را اجرا کنیم. به‌عبارت دیگر دقیقا دستور را اجرا کردیم و به برنامه خود اعلام کرده‌ایم که دستور کپی را اجرا کن، نه‌این‌که متد execute کلاس Copy را اجرا کنیم.

بحثی که ارائه کردیم، فقط گوشه‌ای از مفهوم خط لوله بود. بیایید یکی از کاربردهای اصلی خط لوله را (که در لینوکس و یونیکس استفاده می‌شود) با هم بررسی کنیم:

در خط فرمان لینوکس، کاراکتر پایپ (|) برای استفاده چند دستور پشت هم به‌کار می‌رود. به‌عنوان مثال، خط دستور زیر را در نظر بگیرید:

```
$ ps  -ax | grep click
```

این دستور ابتدا دستور ps را اجرا می‌کند، و بعد خروجی خود را به grep می‌دهد تا تنها رکوردهایی را برگرداند که عبارت کلیک داخل آنها وجود دارد.

```
class xCopy : public baseCommand{
public:
void execute(){
dir _dir;
_dir.execute();
int index = 0;
for(index ; index < _dir.fileCount ;
index++){copy.execute();}}}
```