کتاب آموزش جامع

زبان برنامه نویسی C#

اسلام احمد زاده

هرگونه انتقاد و پیشنهاد در مورد مطالب کتاب را با شماره ۰۹۱۷۷۱۱۲۱۶۱ درمیان بگذارید تا در نسخه های بعدی اصلاحات لازم صورت گیرد. با تشکر

(من اسلام احمد زاده در زمان نوشتن این کتاب استاد آموزشکده فنی باهنر شیراز و موسسه غیر انتفاعی زند شیراز میباشم و دست تمام دانشجویانی را میبوسم که با رفتار خود به من بزرگ منشی را یاد داده اند و از تمام دانشجویانی که احساس میکنند در نمره دادن به آنها ظلم کرده ام عذر خواهی میکنم.)

فهرست مطالب

فصل یک

مقدمه‌ای بر .NET و C#

در این فصل یاد خواهید گرفت:

- **مروری بر چارچوب .NET: معماری و ویژگی‌ها**
- **CLR[1]: مروری بر کارهای قابل انجام توسط قسمت زمان اجرای چارچوب.**
- **کامپایلر JIT[2] .NET. (فقط در لحظه بارگذاری اسمبلی‌ها و تایید کد لازم است).**
- **CLS[3] و CTS[4]: قوانینی که سازگاری CLR و تعامل بین زبانی را مدیریت می‌کنند.**
- **اسمبلی‌ها: نگاهی کوتاه به ساختار یک اسمبلی، فلسفه‌ی پشت آن و تفاوت مابین اسمبلی‌های خصوصی و اشتراکی.**
- **FCL[5]: این کتابخانه صدها کلاس پایه‌ی گروه‌بندی شده در فضاهای نامی منطقی را فراهم می‌کند.**
- **ابزار توسعه: بوسیله‌ی .NET چندین ابزار با هدف توسعه‌ی کد فراهم شده است. این ابزار شامل ILdasm برای نمایش کد، WinCV برای مشاهده خصوصیات یک کلاس و ابزار دیگر پیکربندی چارچوب.**
- **کامپایل کردن و اجرای برنامه‌های C#: کاربرد کامپایلر C# از خط فرمان و گزینه‌هایی برای پیکربندی یک برنامه.**

استفاده‌ی کارای یک زبان، یادگیری گرامر و ویژگی‌های زبان را نیاز دارد. در حقیقت بخش اعظم منحنی یادگیری برای تکنولوژی جدید، به محیط برنامه‌نویسی مرتبط است. حرفه‌ای شدن در C# کافی نیست. معمار نرم‌افزار و توسعه دهنده‌ی موفق باید کتابخانه‌های اصیل کلاس و ابزار تولید آنها را بشناسد.

از منظر برنامه‌نویسی .NET Platform شامل یک محیط اجرایی پیوند خورده به یک کتابخانه کلاس پایه است، که در این فصل CLR، FCL و طرز کار با .NET Platform بررسی خواهد شد.

۱-۱ مروری بر ساختار .NET

ساختار .NET بصورت یک محیط مجتمع برای توسعه و اجرای برنامه‌های اینترنتی، برنامه‌های کاربردی ویندوز(میزکار) و حتی دستگاه‌های موبایل طراحی شده است. اهداف اصلی آن بصورت زیر است:

- **فراهم‌ساختن یک محیط شی‌گرایی مابین دامنه‌ای از کاربردها.**
- **با فراهم ساختن این محیط، تداخل نسخه‌های DLL را کم کرده و پروسه توزیع و نصب کد را ساده می کند.**

---

[1] Common Language RunTime

[2] Just In Time

[3] Common Language Specification

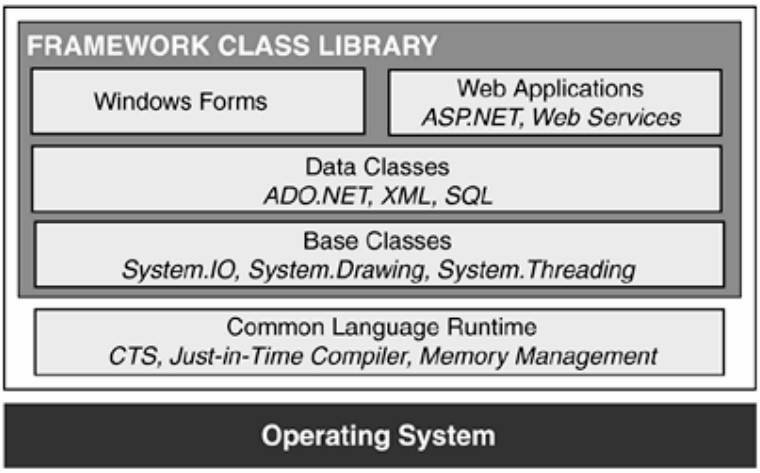
[4] Common Type System

[5] Framework Common Library

- یک محیط قابل‌حمل[۱] براساس استانداردهای تایید شده آماده می سازد تا بتوانند توسط هر سیستم عاملی میزبانی شوند. در حال حاضر C# و یک بخش اصلی از زمان اجرای .NET به نام CLI[۲] بوسیله‌ی ECMA[۳] استاندارد شده‌اند.
- فراهم ساختن یک محیط مدیریت شده، که اجرای امن را به سادگی تایید می‌کند.

طراحان چارچوب .NET برای رسیدن به این اهداف بزرگ روی یک معماری به توافق رسیدند که چارچوب را به دو بخش تجزیه می‌کند: CLR و .FCL شکل ۱-۱ آن را ارائه می‌کند.

شکل ۱-۱ معماری .NET Framework



CLR(پیاده‌سازی استاندارد CLI توسط مایکروسافت) اجرای کد و همه کارهای تخصیص یافته به آن همچون کامپایل، مدیریت حافظه، امنیت، مدیریت ریسمان و ایمنی از نوع داده[۴]، کد اجرا شده تحت نظر CLR (که کد مدیریت شده می‌نامند)، کد مدیریت نشده توسط CLR همچون COM و API را اداره می‌کند.

FCLقطعه اصلی دیگر می‌باشد. یک کتابخانه کد قابل استفاده مجدد (شامل کلاس‌ها، ساختارها و غیره)، که برای برنامه‌های اجرا شده تحت .NET در دسترس است. همه زبان‌های .NET این کتابخانه کلاس مشترک را استفاده می‌کنند. پس این مفاهیم در همه زبان‌های .NET مشترک خواهد بود.

۱-۱-۱-.NET و استانداردهای CLI

یک توسعه‌دهنده قبل از صرف زمان برای یادگیری .NET و C# می‌پرسد: آیا این مهارت را می‌تواند به Platformهای دیگر تبدیل کند؟ آیا محصول .NET مایکروسافت فقط مختص سیستم‌عامل ویندوز است؟ یا آیا آن قابل حمل اجرایی است و آن برای حمل روی سیستم‌عامل‌های دیگر نیز پیاده‌سازی شده است؟ برای جواب دادن به این سئوال، فهمیدن رابطه‌ی مابین C#.NET و استانداردهای CLI ضروری است.

CLI یک محیط اجرایی مجازی مستقل از Platform را تعریف می‌کند. آن هیچ سیستم‌عاملی را تعیین نمی‌کند و برای لینوکس همانند ویندوز راحت است. بخش استاندارد مرکزی، تعریف یک CIL[۵] است که باید توسط کامپایلرهای مطیع CLI

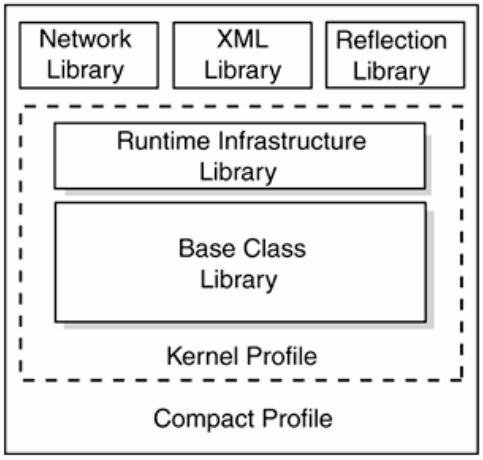
---

[۱] Portable
[۲] Common Language Infrastructure
[۳] European computer Manufactures Association
[۴] Type Safety
[۵] Common Intermediate Language

تولید شود. بخش دیگر سیستم نوع[1] است که همه انواع داده‌ای پشتیبان شده توسط هر زبان مطیع CLI را تعریف می‌کند. این کد میانی به زبان اصلی سیستم‌عامل میزبان کامپایل می‌شود.

CLI استانداردهایی را برای زبان C# در بر دارد که بوسیله مایکروسافت توسعه و ارتقاء یافته‌اند.(Fortran, Pascal, Pythen, Defacto)

چاچوب .NET ارائه شده در شکل۱ـ۱، پیاده‌سازی مایکروسافت از استانداردهای CLI است. این پیاده‌سازی ویژگی‌های زیادی دارد که بوسیله معماری CLI مشخص می‌شوند. برای فهم بیشتر، آن را با معماری استاندارد های CLI شکل ۱-۲ مقایسه کنید.



شکل ۱-۲ معماری تعریف شده بوسیله مشخصه CLI

این کتاب پیاده‌سازی مایکروسافت را تشریح می‌کند. فرض بر آن است که می‌توانیم از کدهای قبلی نوشته شده نیز استفاده کنیم و کدهای پیاده‌سازی شده توسط .NET به سیستم‌عامل دیگری منتقل نخواهد شد و محیط مجازی گفته شده نیز شفاف است.

۱-۲- *CLR*

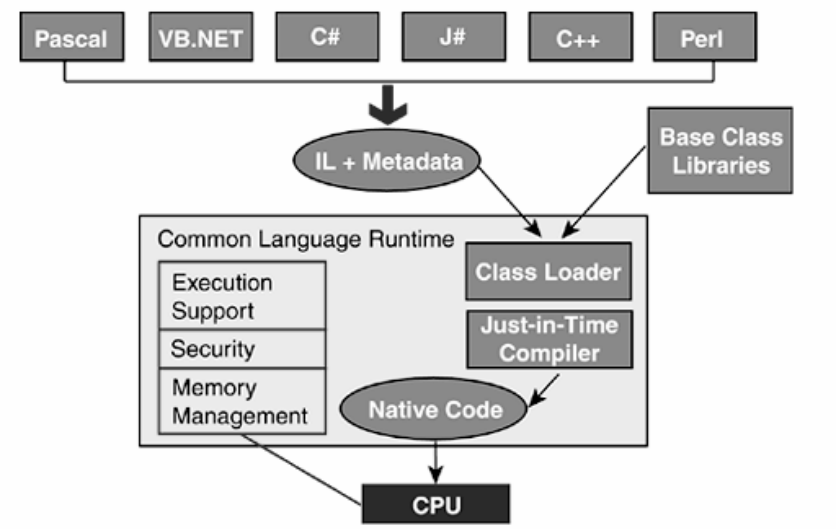
CLR تمام چرخه‌ی زندگی یک برنامه کاربردی را مدیریت می‌کند. آن کد را یافته و کامپایل می‌کند. کلاس های تخصیص یافته را بارگذاری می‌کند. اجرایش را مدیریت می‌کند و مدیریت اتوماتیک حافظه را مطمئن می‌سازد. آن ارتباط بین زبانی را پشتیبانی می‌کند، تا تعامل مابین کدهای نوشته شده در زبان‌های مختلف را مجاز دارد. این بخش کارکرد داخلی CLR را نشان می‌دهد. آن یک بحث عمقی نیست و فقط می‌خواهد شما را با اصطلاح آن آشنا سازد.

۱-۲-۱-کامپایل کردن کد .NET

کامپایلرهای مطیع CLR کدی تولید می‌کنند که کد هدف زمان اجرا بوده و برای یک CPU خاص پیشنهاد شده است. این کد به نام CIL است. IL یا MSIL یک زبان از نوع اسمبلی است که در یک فایلDLL یا EXE بسته‌بندی می‌شود. توجه کنید که اسمبلی‌ها فایل‌هایی با استاندارد اجرایی نیستند. لازم است یک کامپایلر زمان اجرا به نامJIT , IL را به یک کد ماشین خاص تبدیل کند. (زمانی که برنامه واقعاً اجرا می‌شود). چون CLR مسئول مدیریت این IL است، این کد میانی یکی از کلیدهای رویارویی با اهداف اسمی چارچوب .NET از نظر سازگاری زبان است. همانطور که شکل ۱-۳ نشان می‌دهد، CLR

[1] Type System

نمی‌داند چه زبانی این برنامه را ایجاد کرده است. تعامل آن با IL مستقل از زبان می‌باشد. چون برنامه‌ها از طریق IL با هم ارتباط برقرار می‌کنند، پس خروجی یک کامپایلر می‌تواند با خروجی کامپایلر متفاوت دیگر مجتمع شود.



شکل ۳-۱ عملکرد CLR

هدف دیگر .NET، قابلیت حمل Platform است، که با محلی‌کردن[۱] ایجاد کد ماشین در کامپایلر JIT فراهم می‌شود. بدین معنی که IL تولید شده روی یک Platform می‌تواند روی Platform دیگری که چارچوب خاص خودش و یک کامپایلر JIT با کد ماشین خاص خودش را دارد، اجرا شود.

کامپایلرهایی که کد هدف آنها CLR است، برای هر ماژول علاوه بر تولید IL، باید فراداده[۲]ای را نیز صادر کنند. در فراداده‌ها مجموعه‌ای از جداول قرار می‌گیرند تا هر ماژول، کد خود- توصیف[۳] داشته باشد. در جداول علاوه بر توصیف کامل کد، اطلاعاتی درباره اسمبلی‌ها نیز وجود دارد. این اطلاعات شامل موارد دیگر نیز هستند: چه نوع داده‌هایی در دسترس هستند؟ نام هر نوع داده، اعضای نوع داده، دامنه یا میدان دید نوع داده و ویژگی‌های هر نوع داده دیگر. فراداده‌های دیگری که کاربردهای زیادی دارند:

- مهمترین کاربرد آن بوسیله کامپایلر JIT است، که اطلاعات همه‌ی نوع داده‌های مورد نیاز برای کامپایل کردن را مستقیماً از فراکد[۴] جمع‌آوری می‌کند. این اطلاعات را برای بررسی کد بکار می‌برد تا مطمئن شود برنامه عملیات را به درستی انجام می‌دهد. برای مثال، JIT از طریق مقایسه‌ی پارامترهای متد، فراخوانی صحیح را مطمئن می‌سازد.
- فراداده‌ها در پروسه جمع‌آوری زباله[۵] استفاده می‌شوند. جمع‌کننده زباله، برای شناسایی فیلدها و ارجاعات آنها از فراداده استفاده می‌کند و می‌تواند تعیین کند حافظه‌ی چه اشیایی می‌توانند آزاد شوند یا نه؟
- .NET یک مجموعه از کلاس‌ها برای خواندن فراداده‌های یک برنامه فراهم می‌کند. این توانایی به نام انعکاس[۶] شناخته می‌شود، که یک ویژگی قدرتمند است و اجازه می‌دهد یک برنامه در زمان اجرا، کد را مورد جستجو قرار دهد و براساس اطلاعات یافته شده تصمیم‌گیری کند. می‌توان صفات سفارشی را به فراداده اضافه کرد.

---

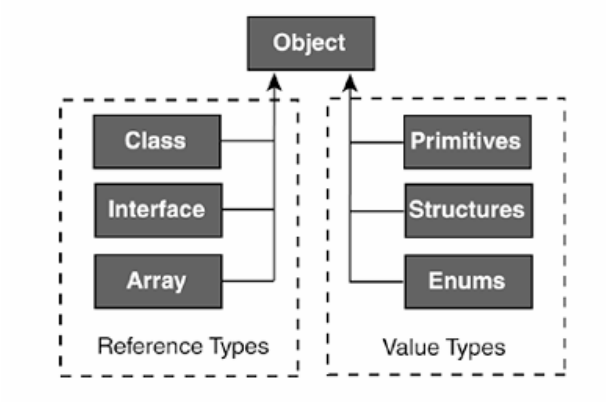
[۱] Localizing
[۲] Metadata
[۳] Self descriptive
[۴] Meta code
[۵] Garbage Collection
[۶] Reflection

IL و فراداده برای فراهم ساختن ارتباط بین زبانی بسیار مهم هستند. اما دنیای واقعی به همه کامپایلرهای .NET که یک مجموعه‌ی مشترک از انواع داده‌ای و توصیف زبان را پشتیبانی می‌کنند، منوط است. برای مثال، دو زبان در IL سازگار نیستند، اگر یکی عدد صحیح علامت‌دار ۳۲بیتی را پشتیبانی کند و دیگری آن را پشتیبانی نکند. آنها ممکن است گرامر متفاوتی داشته باشند، اما باید روی انواع داده‌ای پایه که پشتیبانی می‌کنند، توافق داشته باشند.

همانطور که قبلاً بحث شده، CLI یک توصیف رسمی به نام CTS تعریف می‌کند که بخش مکمل CLR است. آن شرح می‌دهد که چگونه انواع داده‌ای تعریف می‌شوند و چگونه باید رفتار کنند تا بوسیله CLR پشتیبانی شوند.

CTS-۲-۲-۱

CTS یک مجموعه‌ی پایه از انواع داده‌ای برای زبان‌های تحت .NET فراهم می‌کند. علاوه براین نحوه اعلان و ایجاد انواع داده‌ای سفارشی و نحوه مدیریت مدت زمان عمر نمونه‌های این نوع داده‌ها را تعیین می‌کند. شکل ۱-۴ نشان می‌دهد چگونه .NET، CTS را سازمان‌دهی می‌کند.



شکل ۱-۴- انواع داده ای پایه تعریف شده در CTS

از این شکل دو چیز به دست می‌آید. واضح‌ترین مورد اینکه، انواع مقداری[۱] یا ارجاعی[۲] گروه‌بندی می‌شوند. این طبقه‌بندی بر اساس نحوه‌ی ذخیره و دسترسی در حافظه است. انواع ارجاعی در یک ناحیه‌ی خاص حافظه به نام Heap، از طریق اشاره‌گرها دستیابی می‌شوند، در حالیکه انواع مقداری مستقیماً در Stack برنامه قرار می‌گیرند. مورد دیگر اینکه، همه انواع داده‌ای اعم از سفارشی و انواع داده‌ای .NET از یک نوع داده‌ی پیش تعریف شده بنام System.Object ارث‌بری می‌کنند. پس مطمئناً همه انواع داده‌ای، یک مجموعه‌ی پایه از متدها و خصوصیات را ارث‌بری می‌کنند.

در.NET،"نوع داده"[۳] یک عبارت کلی است که به یک کلاس، ساختار، نوع شمارشی یا واسط یا نماینده اشاره می‌کند.

کامپایلری که مطیع مشخصه CTS است، تضمین می‌کند که انواع داده‌ای آن می‌توانند بوسیله‌ی CLR میزبانی شوند. این به تنهایی ضامن ارتباط یک زبان با زبان دیگر نیست. یک مجموعه‌ی محدود کننده از مشخصات به نام CLS وجود دارد که قوانینی را برای ارتباط بین زبانی تعریف می‌کنند. این مشخصات، ویژگی‌های حداقلی تعریف می‌کنند که یک کامپایلر با هدف CLR باید شامل باشد. جدول ۱-۱ بعضی از قوانین CLS را نشان می‌دهد:

جدول۱-۱ قوانین و ویژگی‌های CLS

---

[۱] ValueType

[۲] Refrence Type

[۳] Type

ویژگی	قانون
میدان دید[1]	این قوانین فقط به آن اعضایی از یک نوع داده اعمال می‌شوند که از بیرون اسمبلی تعریف کننده در دسترس است.
کاراکترها و حالت آنها	در دو متغیر متمایز، باید اختلاف آنها بیشتر از حالت کاراکترها در آنها باشد.
انواع اصلی	انواع داده‌ای اصلی مطیع CLS هستند. Byte,Int۱۶,Int۳۲,Int۶۴,Single,Double,Booleam,char,Decimal,Intptr,String
سازنده	یک سازنده قبل از دسترسی به هر داده از کتابخانه، باید سازنده کلاس پایه را فراخوانی کند.
محدوده‌های آرایه	همه ابعاد آرایه‌ها باید از اندیس صفر شروع شوند.
نوع شمارشی	نوع داده اصلی یک نوع شمارشی باید از نوع Byte و Int۱۶ Int۳۲, Int۶۴ باشد.
متد	انواع داده‌ی پارامترها و مقدار بازگشتی استفاده شده در متد باید مطیع CLS باشند.

این قوانین مشخص و واضح هستند. قطعه کدی از C#را در نظر بگیرید تا نحوه‌ی اعمال این قوانین را ببینیم:

```
public  Class  Conversion
}
     public double Metric (double inches)
       {
       return (۲.۵۴ * inches) ;
       }
     public double metric (double miles)
       {
       return (miles/ .۶۲);
       }
{
```

اگرچه با کد C# آشنا نیستید، ولی می‌توانید به راحتی ببینید که با قوانین CLS مغایرت دارد. چون دو متد Metric و metric قانون را نقض کرده‌اند. اگرچه در C# جواب می‌دهد، ولی درصورت تعامل با کدنویسی VB.NET با شکست مواجه می‌شود.

۱-۲-۳-اسمبلی‌ها

همه کدهای مدیریت شده که تحت .NET اجرا می‌شوند، باید در یک اسمبلی قرار گیرند. بطور منطقی اسمبلی یک فایل EXE یا DLL است. از نظر فیزیکی ممکن است شامل کلکسیونی از یک یا چند فایل باشد، که هر فایل می‌تواند شامل کد یا منابعی همچون تصاویر یا XML باشند.

زمانی که یک کامپایلر سازگار .NET یک فایل کد منبع را به یک DLL یا EXE تبدیل می‌کند، یک اسمبلی ایجاد می‌شود. همانطور که در شکل ۱-۵ مشاهده می‌کنید، یک اسمبلی شامل یک اظهارنامه[2]، فرا داده و IL است. این موارد را بیشتر بررسی می‌کنیم.

اظهار نامه

[1] Visiblity(Scope)

[2] manifest

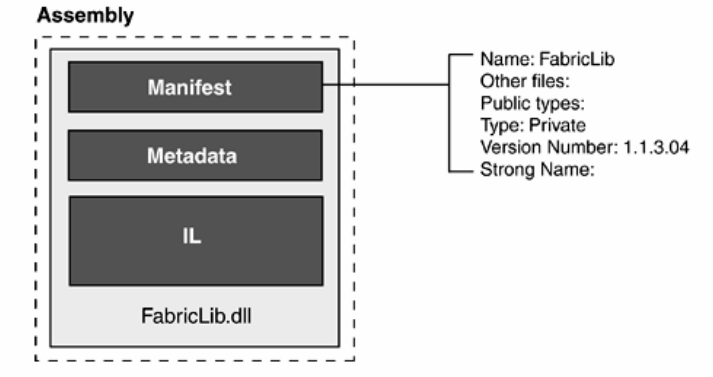
هر اسمبلی باید یک فایل برای در برداشتن اظهارنامه داشته باشد. اظهارنامه شامل جداولی است که در این جداول، اسامی همه فایل‌های موجود در اسمبلی، ارجاعات به اسمبلی‌های بیرونی و اطلاعاتی همچون نام و نسخه اسمبلی لیست می‌شوند. اسمبلی‌های نامگذاری شده، یک امضاء دیجیتالی[1] منحصر به فرد دارند. زمانی که یک اسمبلی بارگذاری می‌شود، ابتدا فایل manifest توسط CLR بارگذاری می‌شود تا بتواند اعضای اسمبلی را تشخیص دهد.

فرا داده

علاوه بر جداول manifest، کامپایلر C# جداول تعریف نوع داده و ارجاع را تولید می‌کند. جداول تعریف، یک توصیف کامل از انواع داده‌ای موجود در IL فراهم می‌کنند. برای مثال، جداولی برای تعریف انواع داده‌ای، متدها، فلیدها و خصوصیات وجود دارند. جداول ارجاع، اطلاعاتی را در مورد ارجاعات به انواع داده‌ای و اسمبلی‌های دیگر شامل هستند. کامپایلر JIT بر پایه‌ی این جداول، IL را به کد ماشین موردنظر تبدیل می‌کند.

IL

نقش ILدر حال حاضر بحث شده است. قبل از اینکه CLR بتواند IL را بکاربرد، باید در یک اسمبلی EXE یا DLL بسته‌بندی شود. این دو یکسان نیستند. یک اسمبلی EXE نقطه‌ی ورودی دارد که آن را قابل اجرا می‌سازد. یک اسمبلی DLL بصورت یک کتابخانه از تعاریف انواع داده طراحی می‌شود.



شکل ۱-۵ اسمبلی تک فایلی

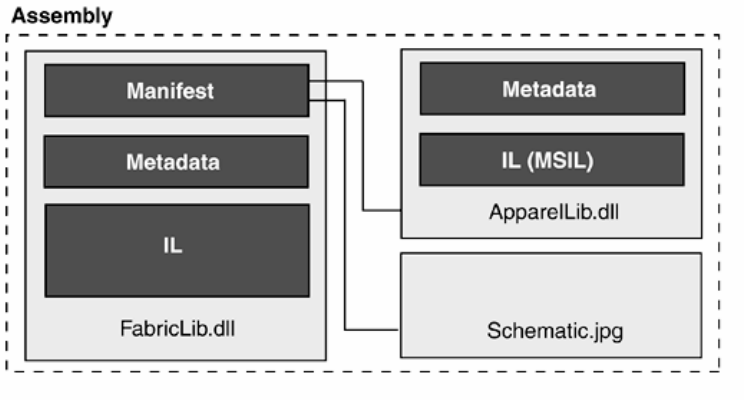
اسمبلی، چیزی بالاتر از یک روش منطقی برای بسته‌بندی کد قابل اجرا است. آن قلب مدل .NET را برای توسعه‌ی کد، کنترل نسخه و امنیت تشکیل می‌دهد.

- تمام کد مدیریت شده‌ی مربوط به یک برنامه مستقل، یک کنترل یا یک کتابخانه DLL شامل انواع داده‌ای قابل استفاده مجدد، در یک اسمبلی بسته‌بندی می‌شوند. آن تجریه‌ناپذیرترین واحد است که می‌تواند روی یک سیستم بکار گرفته شود. زمانی که یک برنامه آغاز می‌شود، باید فقط اسمبلی‌هایی که برای مقداردهی اولیه لازم هستند، در حافظه باشند. اسمبلی‌های دیگر براساس نیاز بارگذاری می‌شوند. یک توسعه‌دهنده این مزیت را برای تقسیم یک برنامه به چندین اسمبلی براساس فرکانس استفاده آنها بکار می‌برد.
- در.NET، اسمبلی یک محدوده‌ی نسخه تشکیل می‌دهد. فیلد نسخه در اظهارنامه روی تمام انواع داده و منابع اسمبلی اعمال می‌گردد. همه فایل‌های تشکیل دهنده اسمبلی بصورت یک واحد منفرد با نسخه‌ی یکسان در نظر گرفته می‌شوند. با جدا کردن بسته فیزیکی از منطقی، .NET می‌تواند یک صفت منطقی را مابین چندین فایل فیزیکی به اشتراک گذارد. این یک ویژگی پایه است که یک اسمبلی را از سیستم مبتنی بر DLL سنتی متمایز می‌کند.

[1] Digital signature

- اسمبلی یک محدوده امنیت روی جوازهای دسترسی تشکیل می‌دهد. C# معرف‌های دسترسی را برای کنترل نحوه دسترسی انواع داده و اعضای انواع داده در یک اسمبلی بکار می‌برد. دو مورد از کاربرد اسمبلی به عنوان محدوده بصورت زیر است:
  - public: دسترسی نامحدود به هر اسمبلی را مجاز می‌شمارد.
  - internal: دسترسی را به انواع داده و اعضای داخل آن اسمبلی محدود می‌کند.

همانطور که شرح داده شد، ممکن است یک اسمبلی چندین فایل را شامل شود که این فایل‌ها محدود به ماژول‌های کد نیستند. ممکن است فایل‌های منبع همچون تصاویر گرافیکی و فایل‌های متنی باشند. یک کاربرد عمومی این فایل‌ها، مجاز داشتن منابعی است که یک واسط برای کشور یا زبان یک کاربر فراهم می‌کند. هیچ محدودیتی روی تعداد فایل‌ها در اسمبلی نیست. شکل ۱-۶ طرح یک اسمبلی چند فایلی را در دیاگرام اسمبلی چند فایلی نشان می‌دهد. توجه داشته باشید که اظهارنامه‌ی اسمبلی شامل اطلاعاتی است که همه فایل‌های اسمبلی را تعیین می‌کند.



شکل ۱-۶-اسمبلی چند فایلی

اگرچه بیشتر اسمبلی‌ها، یک فایل منفرد را شامل هستند، ولی در چندین حالت، اسمبلی‌های چند فایلی مزایایی دارند.

- آنها ترکیب ماژول‌های تولید شده در زبان‌های برنامه‌سازی مختلف را مجاز می‌دارند. اگر کدهایی در VB.NET و کدهایی در C# نوشته شده باشند، این دو کد می‌توانند در یک فایل اسمبلی .NET باهم تعامل داشته باشند.
- برای بهینه کردن نحوه‌ی بارگذاری به CLR، ماژول‌های کد می‌توانند تقسیم‌بندی شوند. بایستی کد پر استفاده و مرتبط بهم در یک ماژول قرار گیرند. CLR ماژول‌ها را در صورت نیاز بارگذاری می‌کند. هنگام ایجاد یک کتابخانه‌ی کلاس، لازم است قطعات کد براساس چرخه زندگی و نسخه و امنیت مشترک در اسمبلی‌های مجزایی گروه‌بندی شوند.
- فایل‌های منبع می‌توانند در ماژول‌های مجزایی از ماژول‌های IL قرار بگیرند، تا چندین برنامه منابع مشترک خود را به راحتی به اشتراک گذارند.

اسمبلی‌های چند فایلی می‌توانند با اجرای کامپایلر C# از خط فرمان یا برنامه سودمند AL.exe (Assembly Linker) ایجاد شوند. یک مثال کاربرد کامپایلر C# از خط فرمان در بخش‌های بعدی آمده است. توجه داشته باشید که VS.NET ۲۰۰۵ ایجاد اسمبلی‌های چند فایلی را پشتیبانی نمی‌کند.

۱-۲-۴-اسمبلی‌های خصوصی[۱] و اشتراکی[۲]

[۱] Private

[۲] Shared

اسمبلی‌ها به دو روش ممکن (خصوصی یا سراسری) ایجاد می‌شوند. اسمبلی‌هایی که در فهرست اصلی برنامه یا در یک زیرفهرست آن قرار می‌گیرند، اسمبلی‌های خصوصی خوانده می‌شوند. نصب و بهنگام سازی یک اسمبلی ساده است. فقط لازم است اسمبلی به یک فهرست به نامAppBase در آن برنامه کپی شود. هیچ تنظیم رجیستری لازم نیست. علاوه بر این برای overrideکردن تنظیمات پیکربندی برنامه می‌توان یک اظهارنامه به برنامه کاربردی اضافه کرد و اجازه داد یک فایل اسمبلی به فهرست AppBase منتقل شود.

یک اسمبلی اشتراکی در یک موقعیت سراسری نصب می‌شود، که GAC[1]نامیده می‌شود و بوسیله چندین برنامه قابل دستیابی است. مهمترین ویژگی GAC مجاز داشتن اجرای چندین نسخه از اسمبلی در کنار همدیگر می‌باشند. .NET برای پشتیبانی از این امر، مشکل تداخل اسمی را با استفاده از چهار صفت جهت شناختن یک اسمبلی رفع می‌کند. نام فایل، مشخصه فرهنگ ، شماره نسخه، نشانه کلید عمومی.

معمولاً اسمبلی‌های عمومی در زیرفهرست Assembly از فهرست سیستم عامل(winnt) قرار می‌گیرند. همانطور که در شکل ۱-۷ نشان داده شده است، اسمبلی‌ها در یک قالب خاص لیست می‌شوند که چهار صفت آن نشان داده می‌شود. چارچوب .NET یک فایل DLL دارد کهWindows Explorer را قادر می‌سازد محتوای GAC را نمایش دهد.

نگاهی سریع به این چهار خصوصیت داریم:

- **نام اسمبلی: این همان نام فایل اسمبلی بدون پسوند آن است.**
- **نسخه : هر اسمبلی یک شماره نسخه دارد که به همه فایل‌های اسمبلی اعمال می‌گردد. آن شامل چهار عدد به قالب زیر است :**

```
<major number>.<minor number>.<build>.<revision>
```

معمولا شماره های اصلی و فرعی نسخه برای تغییرات بروز می‌شوند، چون سازگاری را تحت تاثیر قرار می‌دهند. شماره نسخه بوسیله یک صفت به نام AssemblyVersionدر کد منبع اسمبلی به آن تخصیص داده می‌شود.

- **تنظیم فرهنگ : ممکن است محتوای یک اسمبلی به یک زبان و فرهنگ خاصی تخصیص داده شود، که با صفت AssemblyCulture در کد منبع اسمبلی مشخص می‌گردد.**

```
["assembly: AssmblyCulture ("fr-CA]
```

- **نشانه کلید عمومی: برای اطمینان از اینکه یک اسمبلی اشتراکی، منحصر بفرد و تصدیق شده است، در .NET باید ایجاد کننده اسمبلی آن را با یک نام قوی نشانه‌گذاری کند. این پروسه را امضاء کردن[2] گویند که جفت کلید عمومی/خصوصی را نیاز دارد. در زمان کامپایل اسمبلی، کلید خصوصی برای تولید یک نام قوی بکار می‌رود، کلید عمومی برای نشانه بزرگ است، پس با عمل درهم‌سازی کلیدعمومی، ۸ بایت آخر آن را انتخاب می‌کنند. این نشانه در اظهارنامه‌ی هر اسمبلی سرویس‌گیرنده که به یک اسمبلی اشتراکی ارجاع دارد جای می‌گیرد و برای تشخیص اسمبلی در حین اجرا بکار می‌رود.**

---

[1] Global Assembly Cache

[2] Signing

Assembly Name	Version	Culture	Public Key Token
Accessibility	2.0.3600.0		b03f5f7f11d50a3a
ADODB	7.0.3300.0		b03f5f7f11d50a3a
apphost	2.0.3600.0		b03f5f7f11d50a3a
AspNetMMCExt	2.0.3600.0		b03f5f7f11d50a3a
CRVsPackageLib	1.0.0.0		692fbea5521e1304
CrystalDecisions.CrystalReports.Engine	9.1.3300.0		692fbea5521e1304

شکل ۱-۷- بخشی از فهرست اسمبلی سراسری

۱-۲-۴-از پیش کامپایل کردن یک اسمبلی

بعد از بارگذاری یک اسمبلی، بایستی IL آن به کد ماشین جاری کامپایل شود. اگر شما با فایل‌های قابل اجرا در فرمت کد ماشین کار می‌کنید، سئوالاتی در مورد بهره‌وری و اینکه آیا ایجاد فایل‌های قابل اجرای معادل در .NET امکان‌پذیر است، پیش می‌آید. جواب سئوال قسمت دوم بله است. .NET یک روش برای از پیش کامپایل کردن یک اسمبلی فراهم می‌کند.

چار چوب.NET ابزاری به نام(Native Image Generator (Ngen دارد، که برای کامپایل یک اسمبلی به یک "تصویر محلی[۱]" بکار گرفته می‌شود، که در کش تصویر محلی(یک فضای رزرو شده ازGAC) ذخیره می‌شود. هر زمانی که CLR یک اسمبلی را بارگذاری می‌کند، کش را برای وجود یک تصویر محلی از آن اسمبلی بررسی می‌کند، اگر باشد آن کد از پیش کامپایل شده را بارگذاری می‌کند. ظاهراً، این یک ایده‌ی خوب برای بهبود کارایی به نظر می‌رسد. اما چندین ایراد دارد.

Ngen یک تصویر برای معماری ماشین فرضی اجرا کننده ایجاد می‌کند. برای مثال روی هر ماشین سازگار با پردازنده .X۸۶، زمانی که JITدر .NETاجرا می‌گردد، آن از نوع ماشین آگاه بوده و می‌تواند نکات بهره‌وری را در نظر بگیرد. نتیجه اینکه اغلب اوقات خروجی آن خارج از عملکرد اسمبلی از پیش کامپایل شده است. ایراد دیگر کاربرد یک تصویر محلی این است که تغییرات پیکربندی سخت افزار یا سیستم‌عامل یک سیستم، اغلب اوقات اسمبلی از پیش تعریف شده را نامعتبر می‌کند.

تایید کد[۲]

به عنوان بخشی از پروسه‌ی کامپایل JIT، CLRدو نوع تایید انجام می‌دهد. تایید IL و ارزیابی فراداده. هدف آن اطمینان از قابل قبول بودن کد نوع امن است. در عمل، بدین معنی است که پارامترهای موجود در یک فراخوانی و متد فراخوانی شده، همنوع هستند یا نوع مقدار بازگشتی یک متد همان نوع برگشتی در اعلان است. خلاصه اینکه CLR از طریق IL و فراداده، سازگاری نوع داده‌ها را مطمئن می‌سازد. اگر به غیر از این باشد، یک خطا رخ می‌دهد.

مزیت کد تایید شده این است که CLR یقین دارد، کد از طریق دسترسی به حافظه‌ی خارج از محدوده‌ی مجاز خود نمی‌تواند برنامه‌های دیگر را تحت تاثیر قرار دهد. بدین ترتیب CLR برای اجرای امن چندین برنامه در یک پروسه یا فضای آدرس واحدی است و کارایی و کاهش استفاده از منابع سیستم عامل را بهبود می‌بخشد.

FCL -۳-۱

FCL کلکسیونی از کلاس‌ها و انواع داده‌ی دیگر (نوع شمارشی، ساختارها، واسط‌ها) است که برای تمام کدهای مدیریت شده‌ی نوشته شده در هر زبانی با کد هدف CLR در دسترس هستند. این بسیار مهم است، بدین معنی که این کتابخانه‌ها مختص کامپایلرهای خاصی نیستند. به عنوان یک توسعه‌دهنده، شما می‌توانید با انواع داده موجود درکتابخانه‌ها آشنا شوید، که این دانش در هر زبان .NET برای شما قابل استفاده است.

---

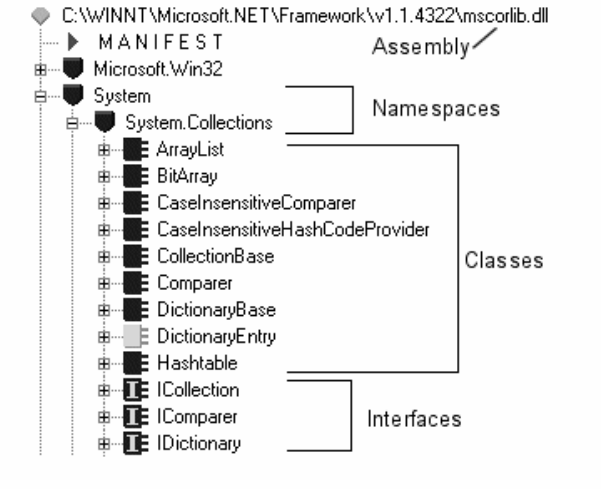
[۱] Native Image

[۲] Code verification

منابع بوسیله FCL از طریق گروه‌بندی‌های منطقی به نام فضای نامی سازمان‌دهی می‌شوند. این گروه‌بندی‌ها براساس محدوده عملکرد می‌باشند. برای مثال، انواع داده‌ای مورد استفاده برای عملیات گرافیکی در فضاهای نامی System.Drawing و System.Drawing.Drawing۲D گروه‌بندی می‌شوند. انواع داده‌ای مورد نیاز برای ورود و خروج فایل‌ها، اعضایی از فضای نامی System.IO هستند. فضاهای نامی، یک مفهوم منطقی نه فیزیکی هستند.

FCL صدها اسمبلی DLL را در برمی‌گیرد. هر اسمبلی ممکن است چند فضای نامی را شامل شود. به علاوه، ممکن است یک فضای نامی چندین اسمبلی را بهم ببافد[1]. برای ارائه این مطلب به داخل یک اسمبلی FCL نگاه کنید.

شکل ۱-۸ بخشی از خروجی تولید شده با برنامه ILdasm.exe جهت کنترل محتوای اسمبلی mscorlib را نمایش می‌دهد. اگرچه یک لیست ناقص است، شما می‌توانید ببینید که فضای نامی mscorlib System را شامل است و انواع داده‌ای پایه‌ی .NET در آن قرار دارند و فضای نامی System.Collections را نیز در بر می‌گیرد که کلاس‌ها و واسط‌های مورد استفاده برای دستکاری کلکسیون‌های داده را شامل می‌شود.



جدول ۱-۲ بعضی از مهمترین فضاهای نامی .NET را لیست می‌کند.

جدول۱-۲ تعدادی از فضاهای نامی متداول

فضای اسمی	کاربرد
System	شامل انواع داده‌ی پایه است که بوسیله همه برنامه‌ها استفاده می‌شود. آن کلاس‌های استثناء، خصوصیاتی از پیش تعریف شده، کتابخانه‌ی Math و کلاس‌های مدیریت محیط برنامه را نیز شامل است.
System.Collections System.Collections.Specialized System.Collections.Generic	واسط‌ها و کلاس‌های استفاده شده جهت مدیریت کلکسیون‌هایی از اشیا. این کلکسیون‌ها شامل ArrayList , Stack,HashTable و ... هستند.

[1] Span

System.Data System.Data.OracleClient System.Data.OleDB System.Data.Odbc	کلاس‌های مورد استفاده برای عملیات پایگاه داده(ADO.NET). فضاهای نامی سرویس گیرنده‌ی Oracle و SQLServer را پشتیبانی می‌کنند و Oledb,Odbc اتصال داده مورد استفاده را تعریف می‌کنند.
System.Diagnostics	کلاس‌هایی را شامل است که می‌توانند اجرای برنامه، اشکال‌یابی، کار با log های سیستم و شمارنده‌های بهره‌وری را پیگیری کنند.
System.Drawing System.Drawing.Drawing۲D System.Drawing.Printing System.Drawing.Text	عملکردهای گرافیکی را برای GDI+ فراهم می‌کند. این فضاهای نامی یک کلاس ترسیم به خوبی fonts,pens,geometric shapes, brushes را در بر دارند.
System.Globalization	کلاس‌هایی در ارتباط با اطلاعات مرتبط با فرهنگ دارد که روش مقداردهی تاریخ‌ها، واحد پول و سمبل‌های نمایشی را تحت تاثیر قرار می‌دهد.
System.IO	عملیات ورود و خروج فایل و جریان داده را فراهم می‌کند. این کلاس‌ها یک روش برای دسترسی به سیستم‌های فایل سیستم عامل میزبان فراهم می‌کنند.
System.Net	کلاس‌هایی که عملیات و پروتکل‌های شبکه را پشتیبانی می‌کنند. برای مثال WebRequest و WebResponse که یک صفحه وب را درخواست و واکشی می‌کنند.
System.Reflection System.Reflection.Emit	انواع داده‌ای که تغییر فراداده را در زمان اجرا مجاز می‌دارند، شامل است. فضای نامی Emit به یک کامپایلر یا ابزار، تولید پویای ILو فراداده را اجازه می‌دهد.
System.Runtime.InterOpServices	ارتباط داخلی مابین کد مدیریت شده وکد مدیریت نشده همچون DLL یا COM را فراهم می‌سازد .
System.Security System.Securing.Permission System.Securing.CryptoGraphy	کلاس‌های استفاده شده برای مدیریت امنیت NET. کلاس‌هایی تعریف می‌کنند که دسترسی به عملیات و منابع را کنترل می‌کنند.
System.Text.ReqularExperssions	کلاس‌هایی که موتور عبارت منظم NET. را پشتیبانی می‌کنند.
System.Threading System.Threading.Thread	فعالیت‌های برنامه‌نویسی ریسمان یعنی ایجاد ریسمان، همگام‌سازی و دسترسی به استخر ریسمان را مدیریت می‌کنند.
System.Web System.Web.Services	کلاس‌های مرتبط با اینترنت که به ASP.NET معروف هستند. آنها نیازهای ارتباط با سرور، دستکاری کوکی‌ها را مدیریت می‌کنند.
System.Web.UI System.Web.UI.WebControls System.WebSecurity	Web.UIدارای کلاس‌ها و واسطهایی است که برای ایجاد کنترل‌ها و صفحات مرتبط با فرم‌های وب استفاده می شوند.