.NET Core와 .NET Framework

IT개발운영1실 강지혜

목차

1. .NET Framework와 .NET Core의 발전 과정
2. .NET Core란?
3. .NET Framework란?
4. .NET Core와 .NET Framework 비교
5. .Net Core를 사용해야할 경우
6. .NET Framework를 사용해야할 경우
7. Garbage Collection
8. .NET Framework와 .NET Core의 발전 과정
   * **.NET Framework**는 2002년 MS에서 나온 소프트웨어 플랫폼(C#, F#, Visual Basic 지원)
   * 윈도우용 어플리케이션은 .NET Framework로 개발하면 생산성과 성능이 좋다.
   * 라이브러리도 풍부
   * 그러나 대부분 상용이고 윈도우 전용이라는 단점 존재했다.
   * 어플리케이션 생태계가 웹과 모바일 중심으로 가면서 리눅스 같은 오픈소스 소프트웨어 도입에 나서기 시작
   * 라이선스 비용이 많이 드는 윈도우 아니어도 수준 높은 웹 어플리케이션 서비스 제공 가능해졌다.
   * .NET Framework의 한계를 넘기 위해 등장한 것이 **.NET Core**
   * 크로스 플랫폼 지원하는 오픈소스 프레임워크
   * 마이크로소프트사의 비영리 오픈소스 단체가 만들었다.
   * .Net Core 출시와 동시에 .NET Standard도 발표
   * 2020년 .NET Framework 4.8을 끝으로 .NET Framework와 .NET Core로 나눠서 제공하지 않고, **.NET**이라는 이름으로 통일
9. .Net core란?
   * 마이크로소프트사의 .net framework의 최신 버전
   * 무료, 오픈소스, general 목적으로 쓰이는 프로그래밍 플랫폼
   * Cross-platform(다양한 운영체제, 언어, 컴퓨터 프로그램 작동 가능)
   * Windows, Mac OS, Linux환경에서도 작동가능
   * 다양한 앱을 만드는데 사용(모바일, 데스크탑, 온라인, IoT, 머신러닝, 마이크로서비스, 게임, 등등..)
   * .NET Core는 가볍고, 빠른 cross-platform 프래임워크다
   * 마이크로소프트사의 비영리 오픈소스 단체가 만들었다.
   * MIT license(MIT에서 개발한 소프트웨어 라이선스)는 .NET Core에서 가지고 있다.( C#과 C++로 개발되어 있다)
   * 첫번째 버전인 .net Core 1.0은 2016년에 제한된 기능으로 공개 되었다.
   * 제일 최신 버전인 3.0버전은 2022년 3월6일에 공개되었다.
10. .Net framework란?
    * Framework란? 쉬운 개발을 위해서, 라이브러리나 다앙한 스크립트 언어를 제공하는 소프트웨어 시스템을 위한 재사용가능한 디자인 플랫폼
    * 윈도우 어플리케이션을 빌드하고 실행하기 위한 소프트웨어 개발 플랫폼
    * 웹사이트, 웹 서비스, 게임을 만들기 위해서 사용된다.
    * 개발자들이 프로그램을 설계할 때 마이크로소프트 윈도우 플랫폼에서 실행가능하게 하기 위해서 디자인 되었다.
    * 2002년에 최초 버전이 공개되었다. (1.0 버전)
    * 그 이후로 엄청난 발전을 거쳐, 최신 버전 4.7.2는 2018년 4월 30일에 공개되었다.
11. .NET Core와 .NET Framework 비교

|  |  |  |
| --- | --- | --- |
|  | .NET Core | .NET Framework |
| 플랫폼 및 프레임워크 | ASP.NET Core같은 플랫폼 | 개발을 위한 UI, DB연결성, 서비스, api 등을 모두 제공할 수 있는 프레임워크 |
| 오픈소스 | 오픈소스 플랫폼 | 특정 오픈소스 요소들 포함(유료 소스 존재) |
| 크로스 플랫폼 | 다양한 운영시스템 호환가능(Window, Linux, Mac OS..) | Window OS에서만 호환 가능 |
| 어플리케이션 모델 | 데스크탑 어플리케이션 개발 미지원, 웹, 모바일 윈도우 개발에 초점 | 데스크탑, 웹 어플리케이션 개발 둘 다 가능, windows forms와 WPF어플리케이션도 지원 |
| 설치 | 크로스 플랫폼이라서 기본운영체제와 별개로 독립적인 패키지 설치 필요 | 윈도우환경을 위한 하나의 패키지 설치 필요 |
| 마이크로서비스의 지원 | 마이크로서비스 지원 가능 | 다양한 언어로 construction과 deployment 허용하지 않는다. |
| REST 서비스 지원 | Windows Communication Foundation 서비스 지원 없음, 항상 REST API 만들어서 사용해야 한다. | RESTful 서비스 사용 가능 |
| 성능 및 확장성 | .NET Framework에 비해 확장성과 성능이 뛰어남 | .Net Core에 비해 확장성과 성능이 떨어진다. |
| 보안성 | 코드 접근 보안 존재하지 않음 | 코드 접근 보안 존재 |
| 다양한 도메인 사용가능 유무 | 게임, 모바일, IoT, AI 같이 다양한 도메인을 가진 앱 개발 가능 | 윈도우 OS환경에 제한되어 있다. |
| 호환성 | Window, Linux, Mac OS같이 다양한 개발 환경과 호환 가능 | Window OS에서만 호환 가능 |
| 모바일 개발 가능성 | 몇 개의 모바일 지원 기능 존재, Xamarin 이나 오픈소스 플랫폼 존재 | 지원하지 않음 |
| CLI 도구 | 모든 플랫폼에 간단한 CLI 환경 제공 | CLI로 하기에는 환경이 너무 무겁다. IDE환경에서 처리하는 것이 낫다. |
| 배포 모델 | 새로운 버전이 설치되면 컴퓨터에서 업데이트, 기존 프로그램에 방해되지 않게 새로운 디렉토리와 폴더를 생성한다. | 업데이트 버전이 나오면 인터넷 배포서버에만 먼저 배치된다. |
| 패키징 및 배송(빌드, 테스트, 릴리즈, 배포) | Nuget 패키지의 모음으로 배송 | .NET Framework의 모든 라이브러리가 함께 패키징되고 배송된다. |
| 라이브러리 | .NET Framework가 만든 대부분의 프로젝트 생성 가능, 그러나, .NET Framework만 지원하는 라이브러리나 Nuget 지원하지 않음 | WPF, Windows Form, ASP.NET이 대표적인 라이브러리다. 위 라이브러리를 사용하려면 .NET Framework 를 사용해야 한다. |

5. .Net Core를 사용해야할 경우

* Cross-platform이 필요할 때
* 마이크로서비스나 도커 컨테이너를 사용하려고 할 때
* 높은 성능과 확장성을 필요로 할 때
* 각각의 어플리케이션에 다른 .NET 버전을 사용할 때

6. .NET Framework를 사용해야할 경우

* 윈도우 환경에서만 사용할 때
* 이미 .NET Framework로 실행되고 있을 때
* .NET Core를 사용할 수 없는 .NET Framework 기술을 사용하고 있을 때

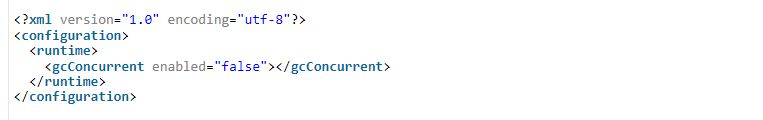
7. Garbage Collection

메모리를 관리하고 더 이상 사용하지 않는 메모리를 해제하는 역할

어플리케이션의 메모리를 주기적으로 스캔해서 사용중인 객체와 그렇지 않는 객체를 결정한다.

더 이상 사용하지 않는 객체는 가비지 컬렉터에 의해 자동으로 해제된다.

* 1. .NET Framework GC
  + .NET Framework의 모든 객체는 힙 영역에 할당
  + 연속적으로 배치되다보니, 메모리가 부족한 시점이 온다.
  + 이 것을 방지하기 위해서 메모리 할당이 특정 크기에 도달하거나 메모리가 부족할 때, gc가 동작한다.
  + GC은 3단계(marketing 단계, relocation 단계, compacting 단계)로 구성된다.
  + 서로 다른 개체의 수명을 더 잘 관리하기 위해 ‘세대’개념 도입
  + 새로 생성된 개체는 0세대에 배치, 가비지 수집을 실행했을 때 해제되지 않으면 1세대로 이동한다. 그 다음 수집에도 해제되지 않으면 2세대로 이동하게 된다.
  + 이렇게 처리하면 대부분의 gc는 0세대만 처리해서 프로세스를 최적화하는데에 사용될 수 있다. (만약 0세대에서 처리가 끝나지 않으면 다음 세대로 가서 메모리를 해제한다.)
  + 2가지 타입의 GC(Workstation GC, Server GC)가 있다
  + Workstation GC는 최신 메모리 할당으로 메모리 임계값을 전달하거나 GC.Collect를 호출하여 일반 우선순위 스레드에서 실행된다.
  + Server GC는 각각의 논리 CPU를 위해 독립적으로 관리되는 힙(heap) 영역을 만들고 대응하는 GC 스레드를 만든다. 스레드는 높은 우선순위를 가지고 작동한다. 이 방식은 GC를 더 빠르게 만들지만 리소스를 더 많이 사용하게 된다.
  + 주로 Workstation GC가 쓰이지만 application configuration XML파일세팅에 따라 바뀔 수 있다.



* + 단일 CPU를 사용하는 컴퓨터라면 파일세팅에 상관없이 Workstaion GC방식이 사용된다.
  1. .NET Core GC
* .NET Framework와 같이 Workstaion GC, Server GC 옵션이 존재한다.
* 그러나, .NET Core는 application configuration XML파일이 없고 runtime configuration JSON file(AssemblyName.runtimeconfig.json)파일에서 세팅을 바꿀 수 있다.



* 대부분 이 파일은 수동으로 편집되지 않아서 MSBuild project(.csproj)파일을 기반으로 내용이 생성된다.
* 가비지 컬렉터를 설정하려면 프로젝트 파일에 밑의 이미지와 같은 property를 추가해야 한다.

