.NET Core 开发实战

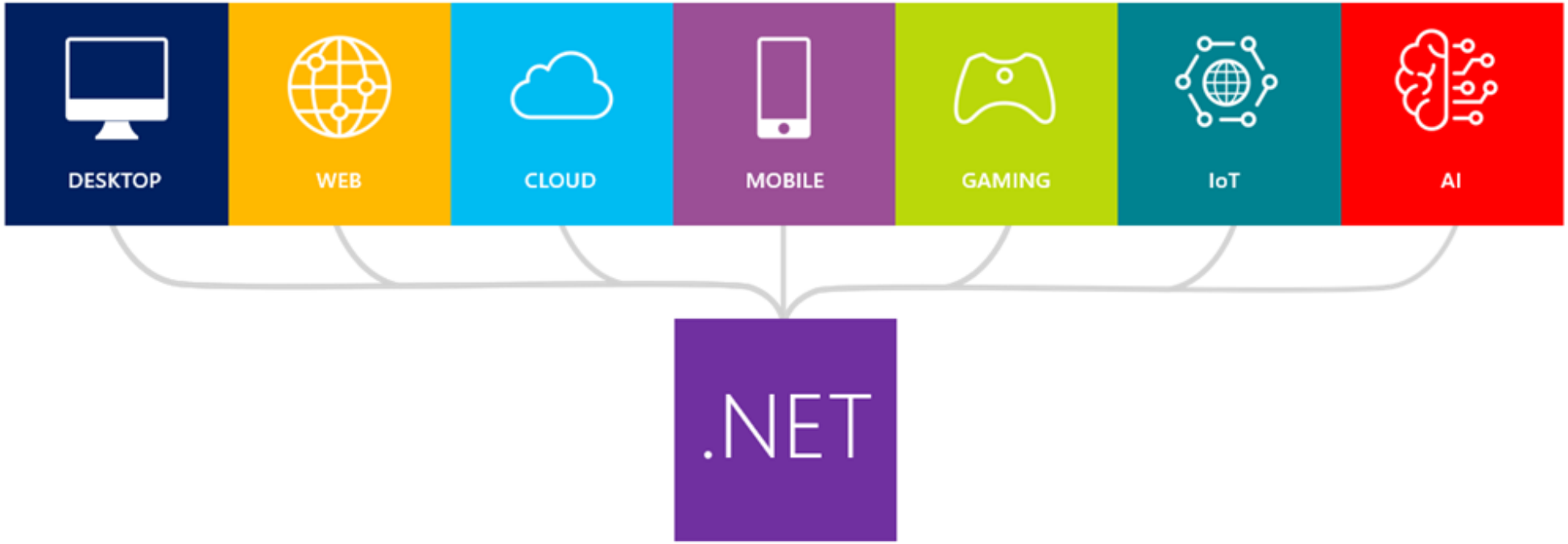
极客时间

扫码试看/订阅

《.NET Core 开发实战》视频课程

1.3 .NET Core 概述

.NET Core 能做什么



.NET Core 的版本历史

时间	版本
2016年2月	.NET Core 1.0 RC1
2016年5月	.NET Core 1.0 RC2
2016年6月	.NET Core 1.0
2017年3月	.NET Core 1.1
2017年8月	.NET Core 2.0
2018年5月	.NET Core 2.1(LTS)
2018年12月	.NET Core 2.2
2019年9月	.NET Core 3.0(Maintenance)
2019年12月	.NET Core 3.1(LTS)
2020年11月	.NET 5.0
2021年11月	.NET 6.0(LTS)
2022年11月	.NET 7.0
2023年11月	.NET 8.0(LTS)

.NET Core 开发工具介绍

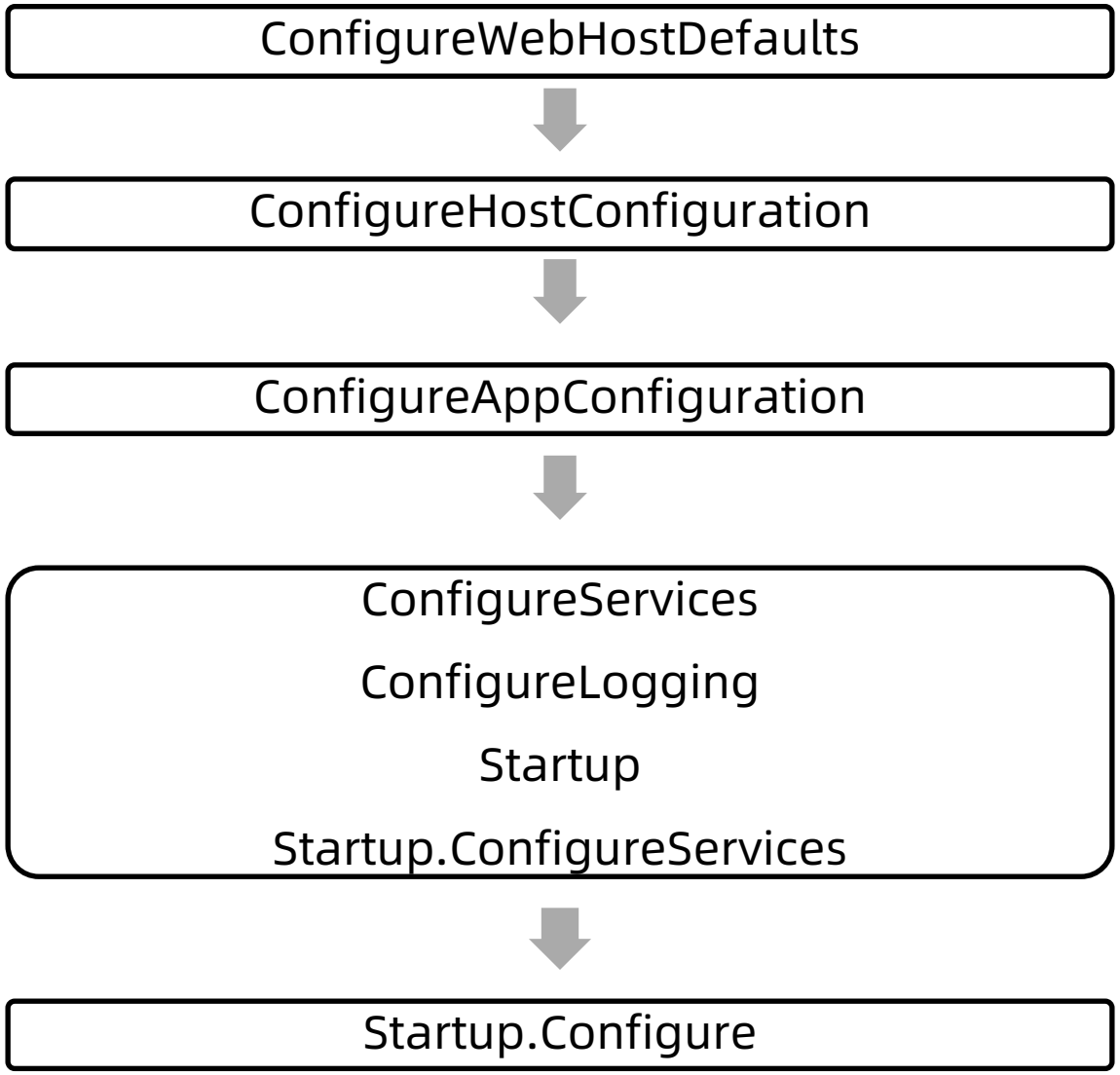
- Visual Studio（Community，Professional，Enterprise）
- Visual Studio for Mac
- Visual Studio Code

1.4 Startup：理解程序启动的过程

内容概要

- IHostBuilder 介绍
- Startup 介绍
- 启动过程执行顺序介绍

启动执行顺序


ConfigureWebHostDefaults
ConfigureHostConfiguration
ConfigureAppConfiguration
ConfigureServices
ConfigureLogging
Startup
Startup.ConfigureServices
Startup.Configure

1.5 依赖注入框架：管理服务的依赖与生命周期

内容介绍

- 为什么要使用依赖注入框架
- 组件包
- 核心接口
- 三种生命周期的行为介绍

为什么要使用依赖注入框架

- 借助依赖注入框架，我们可以轻松管理类之间的依赖，帮助我们在构建应用时遵循设计原则，确保代码的可维护性和可扩展性。

- ASP.NET Core 的整个架构中，依赖注入框架提供了对象创建和生命周期管理的核心能力，各个组件相互协作，也是由依赖注入框架的能力来实现的。

组件包

- Microsoft.Extensions.DependencyInjection.Abstractions
- Microsoft.Extensions.DependencyInjection

核心类型

- IServiceCollection
- ServiceDescriptor
- IServiceProvider
- IServiceScope

生命周期：ServiceLifetime

- 单例 Singleton
- 作用域 Scoped
- 瞬时（暂时） Transient

注意点

- 避免通过静态属性的方式访问容器对象
- 避免在服务内部使用 GetService 方式来获取实例
- 避免使用静态属性存储单例，应该使用容器管理单例对象
- 避免在服务中实例化依赖对象，应该使用依赖注入来获得依赖对象
- 避免向单例的类型注入范围的类型

1.6 依赖注入：理解作用域与对象释放行为

作用域

- IServiceScope

实现 IDisposable 接口类型的释放

- DI 只负责释放由其创建的对象实例
- DI 在容器或子容器释放时，释放由其创建的对象实例

建议

- 避免在根容器获取实现了 IDisposable 接口的瞬时服务
- 避免手动创建实现了 IDisposable 对象，应该使用容器来管理其生命周期

1.7 依赖注入：使用 Autofac 增强容器能力

什么情况下需要引入第三方容器组件

- 基于名称的注入
- 属性注入
- 子容器
- 基于动态代理的 AOP

核心扩展点

- public interface IServiceProviderFactory<TContainerBuilder>

1.8 配置框架：让服务无缝适应各种环境

核心组件包

- Microsoft.Extensions.Configuration. Abstractions
- Microsoft.Extensions.Configuration

配置框架

- 以 key-value 字符串键值对的方式抽象了配置
- 支持从各种不同的数据源读取配置

配置框架核心类型

- IConfiguration
- IConfigurationRoot
- IConfigurationSection
- IConfigurationBuilder

配置框架扩展点

- IConfigurationSource
- IConfigurationProvider

1.9 配置框架：使用命令行配置提供程序接收命令行参数

支持的命令格式

- 无前缀的 key=value 模式
- 双中横线模式 --key=value 或 --key value
- 正斜杠模式 /key=value 或 /key value
- 备注： 等号分隔符和空格分隔符不能混用

命令替换模式

- 必须以单划线 (-) 或双划线 (--) 开头
- 映射字典不能包含重复 Key

1.10 配置框架：使用环境变量配置提供程序接收环境变量

适用场景

- 在 Docker 中运行时
- 在 Kubernetes 中运行时
- 需要设置 ASP.NET Core 的一些内置特殊配置时

特性

- 对于配置的分层键，支持用双下横线 “__” 代替 “:”
- 支持根据前缀加载

1.11 配置框架：使用文件配置提供程序读取配置文件

文件配置提供程序

- Microsoft.Extensions.Configuration.Ini
- Microsoft.Extensions.Configuration.Json
- Microsoft.Extensions.Configuration.NewtonsoftJson
- Microsoft.Extensions.Configuration.Xml
- Microsoft.Extensions.Configuration.UserSecrets

特性

- 指定文件可选、必选
- 指定是否监视文件的变更

1.12 配置框架：跟踪配置变更实现配置热更新

场景

- 需要记录配置源的变更时
- 需要在配置数据变更时触发特定操作时

关键方法

- IChangeToken IConfiguration. GetReloadToken()

1.13 配置框架：使用强类型对象承载配置数据

要点

- 支持将配置值绑定到已有对象
- 支持将配置值绑定到私有属性上

1.14 配置框架：自定义配置数据源与配置中心方案

扩展步骤

- 实现 IConfigurationSource
- 实现 IConfigurationProvider
- 实现 AddXXX 扩展方法

1.15 选项框架：使用选项框架解耦服务与配置

特性

- 支持单例模式读取配置
- 支持快照
- 支持配置变更通知
- 支持运行时动态修改选项值

设计原则

- 接口分离原则（ISP），我们的类不应该依赖它不使用的配置
- 关注点分离 (SoC)，不同组件、服务、类之间的配置不应相互依赖或耦合

建议

- 为我们的服务设计 XXXOptions
- 使用 IOptions<XXXOptions> 、 IOptionsSnapshot <XXXOptions> 、IOptionsMonitor <XXXOptions> 作为服务构造函数的参数

1.16 选项框架：选项数据的热更新

关键类型

- IOptionsMonitor<out TOptions>
- IOptionsSnapshot<out TOptions>

场景

- 范围作用域类型使用 IOptionsSnapshot
- 单例服务使用 IOptionsMonitor

通过代码更新选项

- IPostConfigureOptions<TOptions>

1.17 选项框架：为选项数据添加验证

三种验证方法

- 直接注册验证函数
- 实现 IValidateOptions<TOptions>
- 使用 Microsoft.Extensions.Options.DataAnnotations

1.18 日志框架：聊聊记日志的最佳姿势

总结

- 日志级别定义
- 讲解日志对象获取
- 讲解日志过滤的配置逻辑
- 讲解日志记录的方法
- 避免记录敏感信息，如密码、密钥

1.19 日志作用域：解决不同请求之间的日志干扰

作用域的场景

- 一个事务包含多条操作时
- 复杂流程的日志关联时
- 调用链追踪与请求处理过程对应时

1.20 结构化日志组件 Serilog：记录对查询分析友好的日志

结构化日志的好处

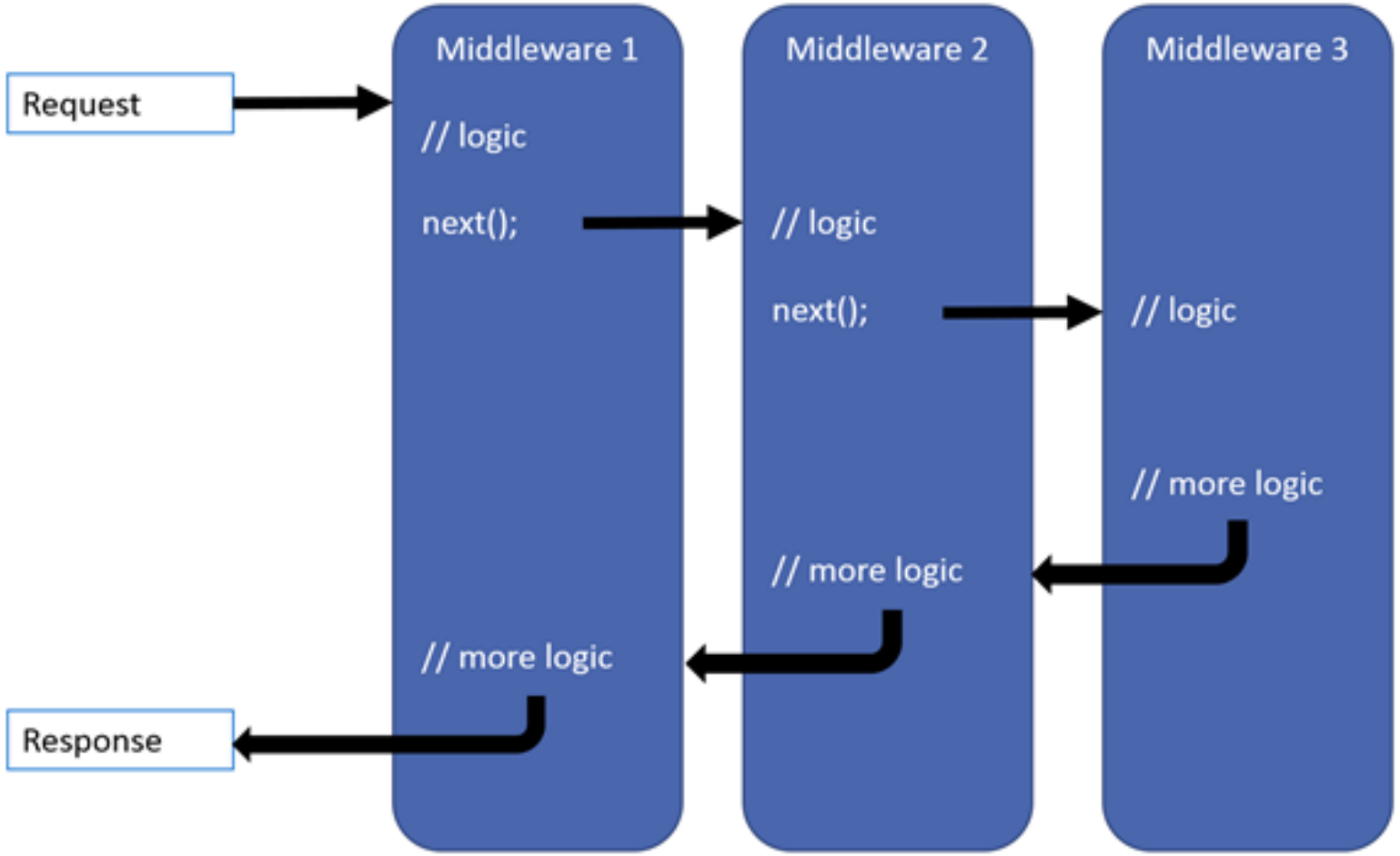
- 易于检索
- 易于分析统计

场景举例

- 实现日志告警
- 实现上下文的关联
- 实现与追踪系统集成

1.21 中间件：掌控请求处理过程的关键

中间件的工作原理


Middleware 1
Middleware 2
Middleware 3
Request
// logic
next();
// logic
next();
// logic
// more logic
// more logic
// more logic
Response

核心对象

- IApplicationBuilder
- RequestDelegate

1.22 异常处理中间件：区分真异常与逻辑异常

处理异常的方式

- 异常处理页
- 异常处理匿名委托方法
- IExceptionFilter
- ExceptionFilterAttribute

异常处理技巧

- 用特定的异常类或接口表示业务逻辑异常
- 为业务逻辑异常定义全局错误码
- 为未知异常定义特定的输出信息和错误码
- 对于已知业务逻辑异常响应 HTTP 200 (监控系统友好)
- 对于未预见的异常响应 HTTP 500
- 为所有的异常记录详细的日志

1.23 静态文件中间件：前后端分离开发合并部署骚操作

静态文件中间件的能力

- 支持指定相对路径
- 支持目录浏览
- 支持设置默认文档
- 支持多目录映射

小任务：实现前端 HTML5 History 路由模式支持

1.24 文件提供程序：让你可以将文件放在任何地方

文件提供程序核心类型

- IFileProvider
- IFileInfo
- IDirectoryContents

内置文件提供程序

- PhysicalFileProvider
- EmbeddedFileProvider
- CompositeFileProvider

1.25 路由与终结点：如何规划好你的 WebAPI

路由注册方式

- 路由模板的方式
- RouteAttribute 方式

路由约束

- 类型约束
- 范围约束
- 正则表达式
- 是否必选
- 自定义 IRouteConstraint

URL 生成

- LinkGenerator
- IUrlHelper

WebAPI 定义

- Restful 不是必须的
- 约定好 API 的表达契约
- 将 API 约束在特定目录下，如 /api/
- 使用 ObsoleteAttribute 标记即将废弃的 API

极客时间

扫码试看/订阅

《.NET Core 开发实战》视频课程