IMERSÃO .NET EXPERT

Treinamento C# Avançado

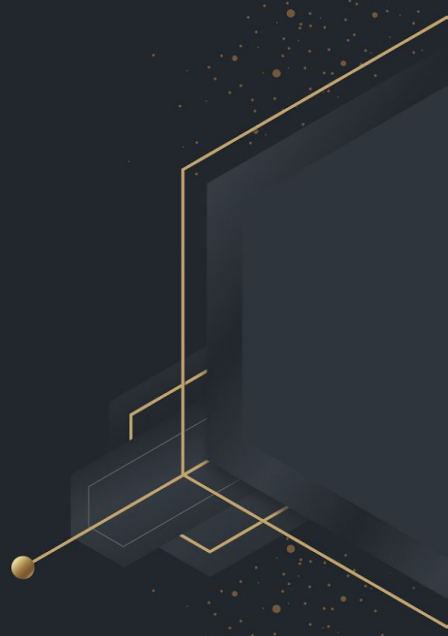
Luis Felipe

Desenvolvedor .NET Sênior,
3x Microsoft MVP

Cassiano Nunes

Arquiteto de Software

Sobre a Treinamento C# Avançado

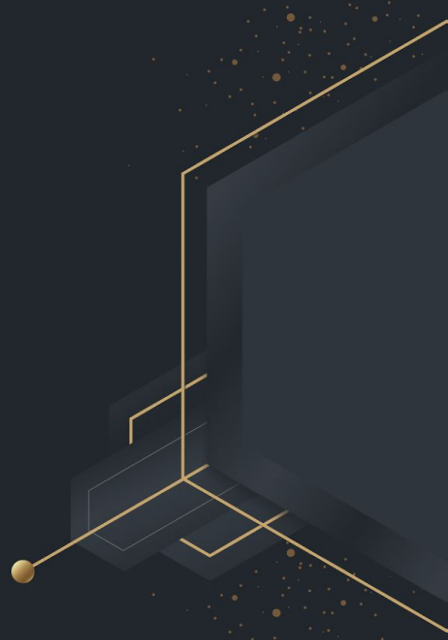


Sobre o Treinamento C# Avançado

O Treinamento C# Avançado é um treinamento para quem deseja se tornar referência técnica na linguagem.

- **Instrutor**
    - Cassiano Nunes, Arquiteto de Software, a 11 anos atuando como desenvolvedor de software.
- **Conteúdos**
    - Garbage Collector
    - Task Parallel Library (TPL)
    - Dynamic e ExpandoObject
    - Reflection e Attributes

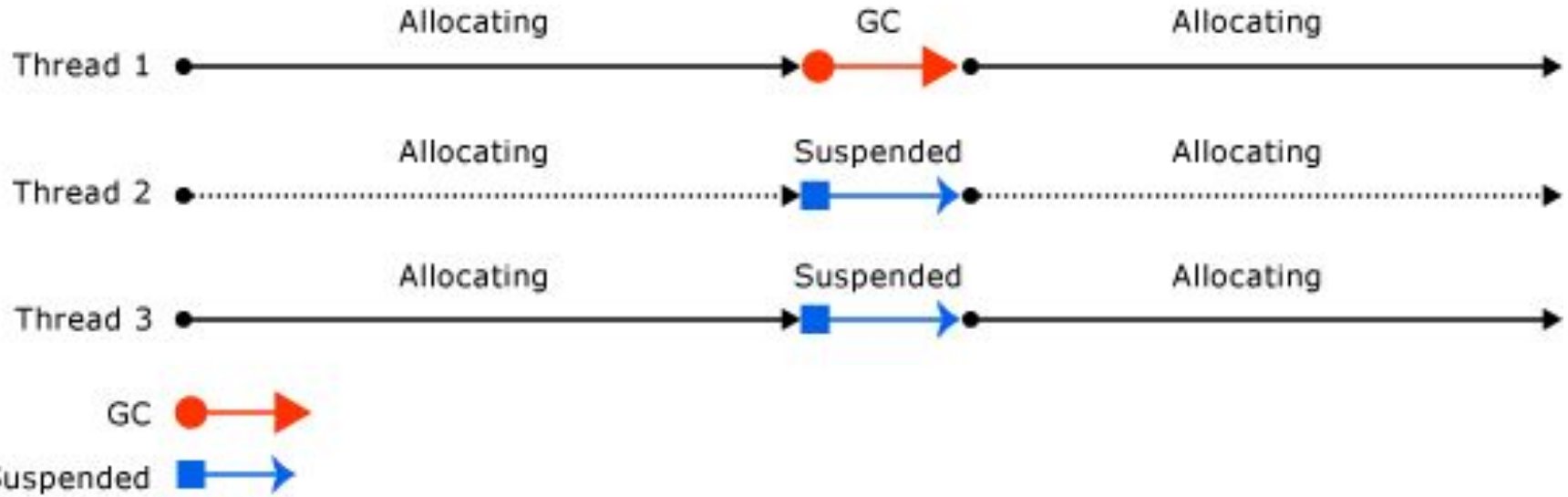
Garbage Collector



Garbage Collector

- Funciona como um gerenciador de memória automático no Common Language Runtime (CLR).
- O desenvolvedor trabalha apenas com espaço de endereço virtual e nunca manipula a memória física diretamente.
- Heap vs Stack.
- Aloca objetos no heap gerenciado com eficiência.
- Recupera objetos que não estão mais sendo usados.

Evite uso desnecessário do Garbage Collector



Quando o Garbage Collector é acionado?

- Não existe total clareza, mas existem condições:
- O sistema tem pouca memória física.
- A memória usada por objetos alocados no heap gerenciado ultrapassa um limite aceitável.
- Quando é feita solicitação de alocação no heap gerenciado.
- O método GC.Collect é chamado (CUIDADO).

Gerações do Garbage Collector:

A coleta de lixo ocorre principalmente com a recuperação de objetos de vida curta

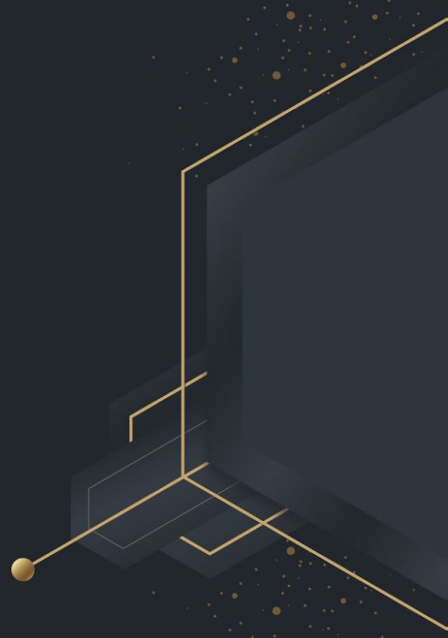
- Geração 0 : Esta geração é a mais jovem e contém objetos de vida curta.
- Geração 1 : Esta geração contém objetos de vida curta e serve como um buffer entre objetos de vida curta e objetos de vida longa.
- Geração 2 : Esta geração contém objetos de vida longa.
- LOH (também pode ser chamado de Geração 3): Outro tipo de memória, utilizado quando necessário uma alocação de memória muito grande (maior que 85000 bytes).

Os objetos que não são recuperados em uma coleta de lixo são conhecidos como sobreviventes e são promovidos para a próxima geração.

IDisposable/Dispose

- Objetos que implementam System.IDisposable ou System.IAsyncDisposable devem ser sempre descartados corretamente, independentemente do escopo de variável.
- Os tipos que definem um finalizador para liberar recursos não gerenciados geralmente chamam GC.SuppressFinalize da implementação Dispose.
- A chamada SuppressFinalize indica ao GC que o finalizador já foi executado e que o objeto não deve ser promovido para finalização.

Task Parallel Library (TPL)



Daniel Abreu Dantas - sistemas1@2flex.com.br - IP: 177.200.160.22

Task Parallel Library (TPL)

- Parte importante do .NET Framework (e do .NET Core / .NET 5+), permitindo a execução paralela de tarefas para melhorar o desempenho de aplicativos.
- A TPL, (em português: Biblioteca de tarefas paralelas) é fornecido pelos conjuntos de tipos públicos em System.Threading e namespaces do System.Threading.Tasks.
- É considerado uma evolução de Thread / ThreadPool.

Task Parallel Library (TPL)

- Benefícios relacionados TPL
  - Possui o objetivo de diminuir a complexidade de trabalhar com paralelismo, aumentando a produtividade do desenvolvedor, podendo ele concentrar-se mais na regra de negócio.
  - Aproximadamente 35% mais performático que a Thread.

Task Parallel Library (TPL)

- Cuidados que devemos ter com TPL
  - Uso inteligente: Seu uso não deve ser feito em cenários onde o bloco de código por si só já é rápido, do contrário, será mais provável que o código venha a ficar lento por conta do gerenciamento das threads.
  - Deadlocks: Evite situações de deadlock, onde várias tarefas estão aguardando umas pelas outras, bloqueando o progresso do programa. Isso pode ocorrer quando há bloqueios mútuos ou dependências circulares entre as tarefas.
  - Depuração: A depuração de problemas em cenários paralelos pode ser complexa. Use ferramentas de depuração, logs e técnicas de profiling para identificar problemas e entender o comportamento das tarefas.

Task Parallel Library (TPL)

- O que é thread safe:
  - Refere a um estado ou condição em que um programa ou sistema pode ser executado simultaneamente por várias threads sem causar comportamentos inesperados ou resultados incorretos
  - Em código "thread-safe", as operações em dados compartilhados são coordenadas de tal maneira que não ocorrem conflitos de acesso e manipulação.

Task Parallel Library (TPL)

- Cuidados em thread safe e Atômicos:
  - Evite Deadlocks: Certifique-se de que os bloqueios sejam adquiridos sempre na mesma ordem para evitar possíveis deadlocks, onde as threads ficam presas esperando por recursos que nunca são liberados.
  - Interlocked: Use métodos e operações "Interlocked" ou outras construções atômicas fornecidas pela linguagem para operações simples que precisam ser executadas de maneira indivisível. Isso evita problemas de condições de corrida.
  - Lock: Assim como Interlocked, ele trata solicitações de forma atômica, porém ele oferece uma flexibilidade maior, uma vez que podemos fazer uso de tipos complexos e até blocos de código serem tratados desta forma. Porém seu cuidado deverá ser mais minucioso, pois uma vez que ele oferece uma flexibilidade maior, também demandará mais recursos durante o processamento.

Task Parallel Library (TPL)

- Um pouco mais sobre Atômicos e "Interlocked":
  - Uma operação atômica é uma operação que é realizada em uma única unidade indivisível, sem ser interrompida por outras operações. Isso significa que, quando uma operação atômica está ocorrendo, nenhuma outra operação pode ocorrer simultaneamente.
  - Interlocked é uma boa opção para variáveis de tipos primitivos.
  - Interlocked.Add: Adiciona um valor a uma variável de forma atômica. Por exemplo, Interlocked.Add(ref counter, 5) adicionaria 5 à variável counter.
  - Interlocked.Increment e Interlocked.Decrement: Incrementa e decrementa uma variável em uma unidade, respectivamente, de forma atômica.

Task Parallel Library (TPL)

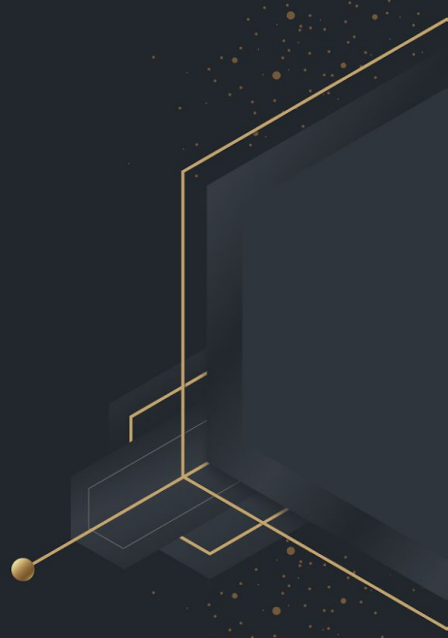
- Um pouco mais sobre Lock
  - Proteção de Recursos Compartilhados: O lock é útil quando você precisa acessar ou modificar dados compartilhados, como objetos ou variáveis, de várias threads.
  - Simplicidade: A construção lock é relativamente simples de usar, pois lida automaticamente com a aquisição e a liberação do bloqueio.
  - Deadlocks: Cuidado ao usar vários bloqueios em sequência, pois isso pode levar a deadlocks, onde as threads ficam bloqueadas indefinidamente esperando por recursos que nunca são liberados.
  - Código na Região Crítica: Mantenha o código dentro da região crítica (lock) o mais simples e rápido possível. Operações que levam muito tempo para executar podem aumentar o risco de bloqueios.

Task Parallel Library (TPL)

- Métodos mais usados em TPL
  - Parallel.For: Executa um loop for no qual as iterações podem ser executadas em paralelo.
  - Parallel.Foreach: Executa uma operação foreach no qual as iterações podem ser executadas em paralelo.
  - Task.WhenAll: Cria uma tarefa (task) que será concluída quando todas as tarefas fornecidas forem concluídas.
  - Task.Delay: Cria uma tarefa que será concluída após um atraso. (alternativa a Thread.Sleep).

Show me the code!

Dynamic



Dynamic

- O dynamic em .NET é um tipo de dados especial que permite que as verificações de tipo sejam realizadas em tempo de execução, em vez de em tempo de compilação. Isso significa que, ao usar dynamic, você não precisa especificar o tipo de dados explicitamente ao escrever código. Em vez disso, o tipo é resolvido somente durante a execução do programa.
- Útil em cenários em que o tipo exato de um objeto não é conhecido até o tempo de execução.

Dynamic

- Benefícios relacionados a dynamic:
  - Flexibilidade para Tratar Dados Dinâmicos: A utilização da palavra-chave dynamic oferece a capacidade de lidar com tipos de dados desconhecidos ou que podem mudar em tempo de execução.
  - Agilidade na Programação e Prototipagem: Em situações onde a velocidade e a prototipagem são essenciais, a utilização do dynamic pode possibilitar uma programação mais ágil. Isso é especialmente útil quando se está explorando dados ou criando protótipos de forma rápida.
  - Redução de Código Repetitivo: Em cenários específicos, a utilização do dynamic pode diminuir a quantidade de código repetitivo necessário para acessar e manipular dados dinâmicos.

Dynamic

- Cuidados que devemos ter:
  - Desempenho: O uso excessivo de dynamic pode afetar o desempenho, pois as resoluções de tipo em tempo de execução são mais lentas do que as verificações de tipo em tempo de compilação.
  - Cuidado com Operações: Certas operações podem falhar em tempo de execução devido a tipos incompatíveis. Verifique e valide cuidadosamente as operações antes de executá-las.
  - Refatoração com Cuidado: Se você tiver código com muitos usos de dynamic e decidir refatora-lo para tipos estáticos, esteja ciente de que isso pode exigir alterações significativas.

Dynamic

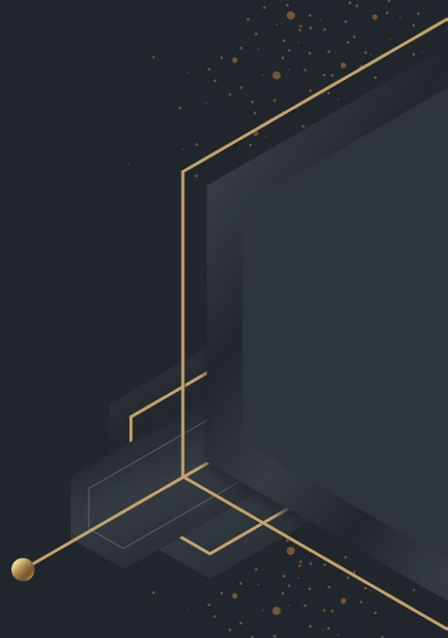
- Principais casos de uso
  - Consulta de Dados Dinâmicos: Em casos em que você está consultando bancos de dados ou fontes de dados que têm esquemas dinâmicos, o dynamic pode ser útil para lidar com diferentes tipos de resultados.
  - UI Dinâmica: Se você estiver construindo interfaces de usuário (UI) dinâmicas ou personalizáveis, o dynamic pode permitir a manipulação de controles e comportamentos sem a necessidade de um conhecimento prévio do layout.
  - Interação com APIs Dinâmicas: Quando você está trabalhando com APIs externas ou bibliotecas COM que expõem objetos dinâmicos, o uso de dynamic pode simplificar a interação, já que os tipos podem ser desconhecidos em tempo de compilação.

Dynamic

- ExpandoObject
  - Usado quando existe a necessidade de criar a instância de variáveis dynamic.
  - Permite adicionar e excluir membros de suas instâncias em tempo de execução e também definir e obter valores desses membros.
  - Oferece suporte a controle de estado através da propriedade PropertyChanged que atua como um delegate.

Show me the code!

Reflections



Reflections

- Permite que um programa examine e interaja com informações sobre tipos, objetos e membros em tempo de execução.
- Descobre detalhes sobre as estruturas de classes, como propriedades, métodos e campos, mesmo que não conheça esses detalhes durante a compilação.
- OBS: o Entity framework faz uso de Reflections para setar as propriedades.

Reflections

- Benefícios de utilizar Reflections
  - Flexibilidade Dinâmica: A Reflection proporciona uma flexibilidade excepcional ao permitir que um programa análise e interaja com informações sobre tipos e membros em tempo de execução.
  - Extensibilidade Aprimorada e Suporte a Plugins: Reflection é fundamental para construir sistemas extensíveis e compatíveis com plugins. Ao permitir que o código principal descubra e interaja com tipos de plugins desconhecidos (ex: Anotações existentes e personalizadas)
  - Criação de Frameworks e Ferramentas de Desenvolvimento: Reflection é amplamente utilizada na criação de frameworks e ferramentas de desenvolvimento que precisam analisar e interagir com código de terceiros.

Reflections

- Cuidados ao utilizar Reflections
  - Segurança: A Reflection pode abrir brechas de segurança se não for usada com cuidado. Dado que ela permite acessar e modificar informações e objetos em tempo de execução
  - Manutenção e Legibilidade: O código que faz uso intensivo de Reflection pode ser mais difícil de entender, manter e depurar.
  - Validação de Dados Dinâmicos: Ao interagir com dados dinâmicos usando Reflection, é crucial realizar uma validação completa desses dados antes de qualquer operação.

Reflections

- Principais usos de Reflections
  - FieldInfo e PropertyInfo: para descobrir informações como o nome, os modificadores de acesso (como public ou private) e detalhes de implementação (como static) de um campo e para obter ou definir os valores de campo.
  - CustomAttributeData: para descobrir informações sobre atributos personalizados ao trabalhar no contexto de somente reflexão de um domínio do aplicativo.

Reflections

- Mas o que é CustomAttributeData?
  - Os atributos fornecem uma maneira de associar informações ao código de forma declarativa.
  - Eles também podem fornecer um elemento reutilizável que pode ser aplicado a vários destinos.
  - são usados para fornecer informações adicionais ao compilador e ao tempo de execução sobre o código

Onde CustomAttributeData é suportado

- Assembly
- Class
- Constructor
- Delegate
- Enum
- Event
- Field
- GenericParameter
- Interface
- Method
- Module
- Parameter
- Property
- ReturnValue
- Struct

Show me the code!