# attributes

Eles foram introduzidos no PHP 8 e são uma poderosa funcionalidade para adicionar metadados ao código.

## O que são Atributos no PHP?

* Atributos são metadados que você pode adicionar a **classes**, **métodos**, **propriedades**, **funções** e **parâmetros de funções.**

Esses metadados podem ser usados para adicionar informações adicionais sobre o comportamento ou intenção de determinado elemento, sem alterar diretamente o código funcional.

## Sintaxe Básica dos Atributos

* A **declaração de um atributo** usa o prefixo #[...] diretamente acima do elemento ao qual ele se aplica.
* Um **atributo** pode ter parâmetros, como em uma chamada de função.

**Exemplo:**

**Atributo Com Parâmetro**

Interface gráfica do usuário

Descrição gerada automaticamente

**Atributo Simples**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média

**Atributo Com Multiplas Linhas**

Texto

Descrição gerada automaticamente com confiança média

**Quando Usar Atributos?**

* **Definir Metadados**: Use atributos para adicionar informações sobre o comportamento de classes, métodos ou propriedades (por exemplo, validação, rotas, etc.).
* **Frameworks**: Muitos frameworks modernos (como Laravel e Symfony) utilizam atributos para simplificar configurações.
* **Automatização**: Reduzir código repetitivo e criar comportamentos dinâmicos.

# Metaprogramação

Antes de nos aprofundarmos, precisamos entender alguns conceitos, como a metaprogramação.

**O que é Metaprogramação?**

Metaprogramação é um paradigma em que o programa pode **analisar**, **modificar** ou **gerar código** dinamicamente. Em outras palavras, é quando o programa pode manipular a si mesmo como se fosse um "dado".

Devido a limitações nas linguagens de programação (entre outras ferramentas computacionais), muitas vezes nos vemos escrevendo as mesmas coisas de novo e de novo e de novo. Se não dá para escrever diretamente um programa que evite essas repetições, o que resta é escrever um programa diferente e então alterá-lo através de uma rotina automatizada (um "meta-programa").

* **Meta** significa "além", então metaprogramação é "programar sobre programação".
* Com metaprogramação, você pode acessar e manipular a estrutura do programa em tempo de execução, como:
  + **Classes**
  + **Métodos**
  + **Propriedades**
  + **Parâmetros**

**Exemplos de Metaprogramação no PHP**

* **Reflection API**: Permite inspecionar e manipular classes, métodos e propriedades dinamicamente.
* **Atributos**: São um exemplo de metaprogramação, pois permitem adicionar metadados que podem ser processados dinamicamente.

A metaprogramação é quando **um programa tem a capacidade de entender, modificar ou gerar o próprio código enquanto está sendo executado**. Em outras palavras, o programa pode tratar partes de si mesmo como **dados**, analisá-los e manipulá-los dinamicamente.

Imagine que você é um **escritor** trabalhando em um livro. Normalmente, você escreve e edita o livro diretamente. Agora, imagine que você tem uma ferramenta mágica que **escreve o livro para você**, baseada em certas regras e condições que você define.

* Essa ferramenta não apenas escreve, mas também pode **ler o que já está escrito** e **modificar o conteúdo** automaticamente quando você altera as regras.
* Isso é metaprogramação! Você está "programando" algo que programa ou modifica o próprio sistema.

# reflection

**Reflection**, no PHP, é uma funcionalidade que permite **inspecionar e manipular o código em tempo de execução**. Ele fornece ferramentas para acessar informações detalhadas sobre classes, métodos, propriedades, parâmetros de funções, atributos e muito mais.

A palavra "Reflection" vem do fato de que o programa pode "se refletir" e **ver a si mesmo** enquanto está sendo executado. Ou seja, ele consegue **analisar sua própria estrutura interna** e, em alguns casos, até modificá-la dinamicamente.

No PHP, a Reflection é implementada como um conjunto de classes e métodos conhecidos como **Reflection API**. Essas ferramentas permitem que você faça metaprogramação — analisar, criar ou alterar o comportamento do código dinamicamente.

Portanto, **Reflection** não é apenas uma classe, mas um **conjunto de classes** no PHP conhecido como **Reflection API**. Cada classe na Reflection API serve para inspecionar um elemento específico do código, como classes, métodos, propriedades, funções, parâmetros, e a própria estrutura de atributos.

## Classes Principais de Reflection API

| **Classe** | **Descrição** |
| --- | --- |
| ReflectionClass | *Inspeciona classes e objetos.* |
| ReflectionMethod | *Inspeciona métodos dentro de classes.* |
| ReflectionProperty | *Inspeciona propriedades dentro de classes.* |
| ReflectionFunction | *Inspeciona funções.* |
| ReflectionParameter | *Inspeciona os parâmetros de funções ou métodos.* |
| ReflectionAttribute | *Inspeciona atributos aplicados a classes, métodos, propriedades, etc. (PHP 8).* |

Cada uma dessas classes tem seus próprios métodos, e claro, sempre podemos encontrar alguns métodos repetidos.

### 1. Métodos da classe Reflection\_class

Texto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamente

### 2. Métodos da classe Reflection\_Method

Texto

Descrição gerada automaticamente

### 3. Métodos da classe Reflection\_Property

Texto

Descrição gerada automaticamente

### 4. Métodos da classe Reflection\_Attribute

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

### 5. Métodos da classe Reflection\_function

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

### 6. Métodos da classe Reflection\_parameter

Texto

Descrição gerada automaticamente com confiança média

Exemplo em código:

Aqui iremos ter 2 código, 1 sem metaprogramação e outro com, o objetivo deles é validar o email.

Texto

Descrição gerada automaticamente**Sem metaprogramação**

**Com Metaprogramação (Usando Atributos e Reflection)**

Texto

Descrição gerada automaticamente

**1. O que o código faz?**

O objetivo do código é:

1. Adicionar um atributo personalizado chamado ValidarEmail a propriedades de uma classe.
2. Usar **Reflection** para identificar as propriedades que têm esse atributo.
3. Validar automaticamente essas propriedades como sendo e-mails válidos.
4. Exibir uma mensagem caso o valor da propriedade não seja um e-mail válido.

### **2. Divisão do Código**

Vamos analisar o código em partes.

#### **Parte 1: Criando o atributo**

Texto

Descrição gerada automaticamente com confiança média

**O que está acontecendo aqui?**

* A classe ValidarEmail é declarada como um **atributo**.
* A anotação #[Attribute] diz ao PHP que esta classe pode ser usada como metadado em outros elementos do código (como propriedades, métodos, etc.).
* Este atributo não tem lógica ou propriedades adicionais, é usado apenas como um **marcador** para identificar quais propriedades devem ser validadas como e-mails.

#### Parte 2: Criando a classe Usuario

Texto

Descrição gerada automaticamente com confiança média

**O que está acontecendo aqui?**

* A classe Usuario possui uma propriedade pública chamada $email e tipada como string.
* A anotação #[ValidarEmail] é aplicada à propriedade $email. Isso marca a propriedade como algo que deve ser validado como um e-mail.
* A anotação funciona como um "rótulo" que será detectado em tempo de execução.

#### Parte 3: Função validarCampos

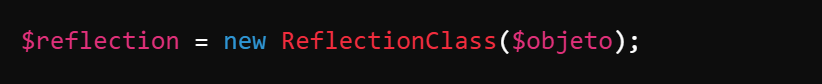
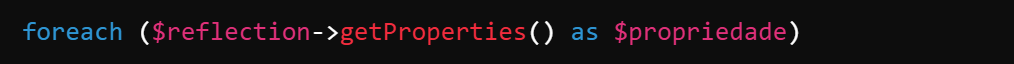
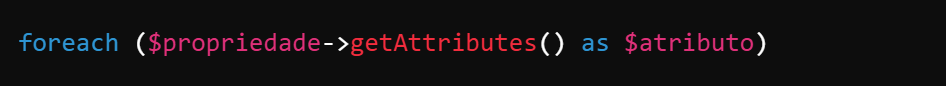
Texto

Descrição gerada automaticamente

**O que está acontecendo aqui?**

* A função validarCampos é usada para **validar automaticamente as propriedades de um objeto** que tenham atributos de validação.

**Passo a passo do que a função faz**:

1. Usa ReflectionClass para criar uma representação da classe do objeto passado como parâmetro.
2. Obtém todas as propriedades da classe usando o método getProperties() :
   * Neste exemplo, apenas a propriedade $email será encontrada.
3. Para cada propriedade, verifica se há **atributos associados** com getAttributes():
   * Aqui, será detectado que a propriedade $email possui o atributo ValidarEmail.
4. Verifica se o nome do atributo é ValidarEmail:  
   * Isso garante que apenas propriedades marcadas com o atributo ValidarEmail sejam validadas.
5. Obtém o valor atual da propriedade com getValue():
   * Para a propriedade $email, o valor será 'teste@exemplo'.
6. Usa a função filter\_var para validar o valor como e-mail:  
   * Texto

     Descrição gerada automaticamenteNeste exemplo, o e-mail 'teste@exemplo' é inválido porque falta o domínio completo (.com, .org, etc.), então a mensagem será exibida.

#### Parte 4: Criando o objeto e chamando a função

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**O que está acontecendo aqui?**

1. Um objeto da classe Usuario é criado.
2. O valor da propriedade $email é atribuído como 'teste@exemplo'.
3. A função validarCampos é chamada, passando o objeto $usuario como parâmetro.
4. A função detecta que $email possui o atributo ValidarEmail e tenta validar o valor como um e-mail válido.
5. Como 'teste@exemplo' não é um e-mail válido, a mensagem **"E-mail inválido!"** é exibida.

# usando atributos

Para usarmos atributos nas classes PHP, temos sempre que colocar #{Attribute}

Esse #{Attribute} é especial e diferente pois informa para o PHP que está é uma classe de atributo.

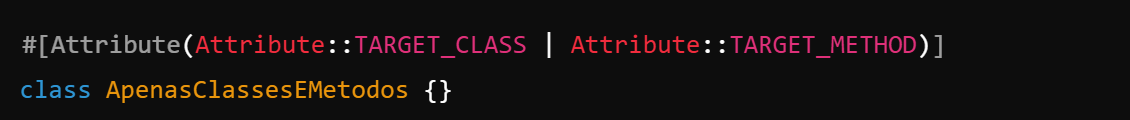
* **É uma declaração obrigatória para atributos personalizados**:
  + No PHP, **somente classes marcadas com #[Attribute] podem ser usadas como atributos**. Se você tentar usar uma classe que não tenha essa anotação como um atributo, o PHP lançará um erro.
* **Faz parte do núcleo da linguagem PHP**:
  + A anotação #[Attribute] é parte do sistema de metaprogramação introduzido no **PHP 8** e está intimamente integrada com a **Reflection API**.
* **Define a classe como um "Atributo"**:
  + Ao adicionar #[Attribute], você está declarando que a classe é usada exclusivamente para fornecer **metadados** e será processada dinamicamente por meio de **Reflection**.
  + Sem #[Attribute], a classe será tratada como uma classe regular.
* **Classe Associada**:
* #[Attribute] é atrelado à classe interna do PHP chamada **Attribute**, que reside no namespace global.
* Texto

  Descrição gerada automaticamenteA classe **Attribute** tem constantes que configuram o comportamento dos atributos, como alvos (TARGET\_\*) e repetibilidade (IS\_REPEATABLE).

Como faz parte de uma classe, também tem suas próprias propriedades, as propriedades da classe Attribute no escopo global, são constantes.

Ela não possui métodos, ela é apenas uma classe final, que contém algumas constantes públicas que servem para definir onde e como o atributo pode ser usado.

## Interface gráfica do usuário, Texto Descrição gerada automaticamenteconstantes de attribute

Essas constantes são passadas como argumento para #[Attribute] para definir onde e como o atributo pode ser usado.

**O que acontece aqui?**

* ApenasClassesEMetodos pode ser aplicado apenas a **classes** ou **métodos**.
* Tentar aplicá-lo a uma propriedade, por exemplo, resultará em erro.
* Você usa :: para acessar suas constantes porque elas são **estáticas**.

**Exclusivo para aqueles que estão embaixo dele**:

* O #[Attribute] **não se propaga** ou afeta outras coisas no mesmo arquivo ou projeto.
* Cada classe etc, que você deseja usar como atributo deve ser anotada individualmente.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**Resultado**:

* O atributo MeuAtributo é válido apenas porque a classe foi anotada com #[Attribute].

**Se Não Colocar #[Attribute]**

Se você tentar usar uma classe sem a anotação #[Attribute] como um atributo, o PHP lançará um erro:

Aplicativo

Descrição gerada automaticamente com confiança baixa

Texto

Descrição gerada automaticamente**Outras classes não são afetadas:**