# atributos pre definidos

## Links para documentação:

- <https://www.php.net/manual/pt_BR/reserved.attributes.php>

***Introdução aos Atributos Predefinidos***

Os atributos no PHP são uma forma de adicionar **metadados estruturados** e **legíveis por máquina** diretamente no código. Esses metadados podem ser aplicados a:

* **Classes**
* **Métodos**
* **Funções**
* **Parâmetros**
* **Propriedades**
* **Constantes de classes**

Esses atributos são inspecionáveis em **tempo de execução** através das APIs de reflexão, o que permite:

* Adicionar **configuração personalizada** ao código.
* Tornar o código mais **autoexplicativo** e **flexível**.

A abordagem com atributos pode ser comparada a **linguagens de configuração**, mas embutida diretamente no código, o que elimina a necessidade de arquivos separados para essa finalidade.

# Atributo Attribute

## Link para documentação:

- https://www.php.net/manual/pt\_BR/class.attribute.php

O atributo Attribute define as **características** de um atributo e controla onde ele pode ser aplicado.

***Resumo da Classe***

Texto

Descrição gerada automaticamenteA classe Attribute é uma classe **final**, o que significa que não pode ser estendida.

***Constantes Predefinidas***

As constantes definem os **alvos permitidos** para os atributos e outras características.

1. **Attribute::TARGET\_CLASS**  
   Permite que o atributo seja aplicado a **classes**.
2. **Attribute::TARGET\_FUNCTION**  
   Permite que o atributo seja aplicado a **funções**.
3. **Attribute::TARGET\_METHOD**  
   Permite que o atributo seja aplicado a **métodos**.
4. **Attribute::TARGET\_PROPERTY**  
   Permite que o atributo seja aplicado a **propriedades**.
5. **Attribute::TARGET\_CLASS\_CONSTANT**  
   Permite que o atributo seja aplicado a **constantes de classe**.
6. **Attribute::TARGET\_PARAMETER**  
   Permite que o atributo seja aplicado a **parâmetros de funções ou métodos**.
7. **Attribute::TARGET\_ALL**  
   É uma constante geral que permite que o atributo seja aplicado a **todas as declarações acima**.
8. **Attribute::IS\_REPEATABLE**  
   Define se o atributo pode ser repetido várias vezes no mesmo alvo.

***Propriedades***

A única propriedade pública da classe Attribute é:

* **$flags**:  
  Indica os alvos permitidos e se o atributo pode ser repetido. Seu valor padrão é Attribute::TARGET\_ALL.

***Métodos***

* **\_\_construct(int $flags = Attribute::TARGET\_ALL)**  
  Constrói uma nova instância da classe Attribute.
  + O parâmetro $flags define os alvos permitidos e se o atributo pode ser repetido.

***Exemplo Prático***

***Texto

Descrição gerada automaticamente***Aqui está um exemplo para demonstrar como usar o atributo Attribute:

**Explicação do exemplo**:

1. Definimos um atributo MyAttribute com o alvo restrito a **classes** e **métodos**.
2. Aplicamos o atributo à classe Example e ao método doSomething.

# Atributo AllowDynamicProperties

Link para a documentação:

- https://www.php.net/manual/pt\_BR/class.allowdynamicproperties.php

A partir do **PHP 8.2.0**, as **propriedades dinâmicas** (adicionar propriedades que não foram declaradas explicitamente em uma classe) foram **descontinuadas**. Isso significa que, sem o uso de **AllowDynamicProperties**, qualquer tentativa de criar propriedades dinâmicas em uma classe emitirá um **aviso de descontinuação**.

O atributo **AllowDynamicProperties** resolve esse problema, permitindo explicitamente que uma classe continue utilizando propriedades dinâmicas, mantendo a compatibilidade com versões anteriores do PHP.

Fundo preto com letras brancas

Descrição gerada automaticamenteA classe AllowDynamicProperties é definida como **final**, o que significa que não pode ser estendida.

***Método***

* **\_\_construct()**:  
  Construtor que cria uma nova instância do atributo AllowDynamicProperties. Ele não recebe parâmetros e é usado apenas como marcador para ativar o suporte a propriedades dinâmicas na classe.

***Exemplo de Uso***

Texto

Descrição gerada automaticamenteAqui está um exemplo mostrando como usar o atributo AllowDynamicProperties:

1. A classe **DefaultBehaviour** não possui o atributo AllowDynamicProperties. Portanto, a tentativa de adicionar uma propriedade dinâmica (como $nonExistingProp) emitirá um aviso de descontinuação.
2. A classe **ClassAllowsDynamicProperties** possui o atributo AllowDynamicProperties, permitindo que propriedades dinâmicas sejam adicionadas sem problemas.

***Quando Usar AllowDynamicProperties***

1. **Manutenção de Código Legado**:
   * Caso você esteja trabalhando com código mais antigo que depende fortemente de propriedades dinâmicas.
2. **Compatibilidade Temporária**:
   * Se você precisa de tempo para refatorar seu código e remover o uso de propriedades dinâmicas.

***Boas Práticas***

Embora o atributo AllowDynamicProperties seja útil para compatibilidade, o uso de propriedades dinâmicas é **desencorajado**, pois:

1. **Afeta a legibilidade e previsibilidade** do código.
2. **Reduz o suporte a ferramentas de análise estática** (como o PHPStan e o Psalm).
3. **Pode levar a erros difíceis de depurar**, como erros de digitação ou propriedades inexistentes.

Com a descontinuação, o PHP está incentivando práticas de código mais **rígidas e previsíveis**. Dependendo do contexto do seu projeto, pode valer mais a pena **refatorar o código** para remover propriedades dinâmicas do que depender do atributo AllowDynamicProperties.

# Atributo Deprecated

Link para a documentação:

- <https://www.php.net/manual/pt_BR/class.deprecated.php>

O atributo **Deprecated** foi introduzido no **PHP 8.4.0**. Ele é usado para marcar **funcionalidades como descontinuadas**, incentivando os desenvolvedores a migrarem para alternativas mais seguras ou modernas.

Quando uma funcionalidade marcada com Deprecated é usada, o PHP emite um **erro de nível E\_USER\_DEPRECATED**, que serve como um aviso aos desenvolvedores.

Esse atributo é útil para:

1. **Indicar funcionalidades que não serão mantidas no futuro.**
2. **Orientar desenvolvedores sobre substituições adequadas.**
3. **Facilitar o processo de descontinuação em sistemas grandes.**

***Resumo da Classe***

Texto

Descrição gerada automaticamenteA classe Deprecated é definida como **final**, ou seja, não pode ser estendida.

***Propriedades***

1. **message** (opcional):  
   Uma string que fornece detalhes sobre o motivo da descontinuação ou a funcionalidade recomendada para substituição.
   * Exemplo: "Use 'safe\_replacement()' instead".
2. **since** (opcional):  
   Uma string que informa desde quando a funcionalidade foi marcada como descontinuada.
   * Pode conter números de versão, datas ou outras informações.
   * Exemplo: "8.4".

***Método***

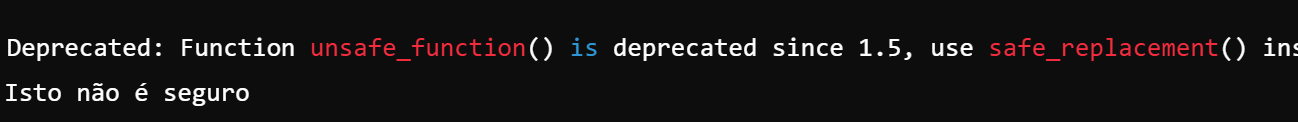
* **\_\_construct(?string $message = null, ?string $since = null)**:  
  Construtor que inicializa os valores das propriedades message e since.

***Exemplo de Uso***

Aqui está um exemplo de como o atributo Deprecated pode ser aplicado:

Texto

Descrição gerada automaticamente



***Explicação do Exemplo***

1. **Definição do atributo**:
   * A função unsafe\_function foi marcada como descontinuada usando o atributo Deprecated.
   * O atributo fornece:
     + Uma **mensagem** explicando a substituição recomendada.
     + Uma **versão** (since) indicando quando a funcionalidade foi marcada como descontinuada.
2. **Aviso emitido pelo PHP**:
   * Quando a função é chamada, o PHP emite um aviso com as informações fornecidas no atributo.

***Usos Comuns do Atributo Deprecated***

1. **Descontinuação Gradual**:
   * Ajuda desenvolvedores a planejar a substituição de funcionalidades obsoletas.
2. **Comunicação com a Equipe**:
   * Permite que outros desenvolvedores saibam quais partes do código precisam ser evitadas.
3. **Depreciação em APIs**:
   * Facilita a migração para versões mais recentes de uma API sem quebrar o código imediatamente.

***Boas Práticas ao Usar Deprecated***

1. Sempre forneça uma **mensagem clara** com:
   * O motivo da descontinuação.
   * A funcionalidade de substituição recomendada.
2. Inclua a **versão ou data** na propriedade since:
   * Isso ajuda a rastrear quando a descontinuação foi introduzida.
3. Use o atributo apenas em **funcionalidades que você planeja remover no futuro**.
4. Monitore avisos de descontinuação durante os testes:
   * Certifique-se de que seu código atualizado está livre de dependências obsoletas.

Mas ai você se pergunta :

“ok, mas então pq eu iria colocar uma funcionalidade que eu sei que vai ser removida no futuro e ainda ponho um aviso pra isso ? não é mais fácil eu simplesmente buscar uma alternativa ???”

A resposta pra isso é muito simples...

***Transição Gradual em Sistemas Legados***

Se você está trabalhando em um **sistema grande ou legado**, simplesmente **remover uma funcionalidade obsoleta** pode:

* Quebrar o código de outros desenvolvedores ou equipes que ainda dependem dela.
* Impactar usuários finais no caso de bibliotecas ou APIs públicas, caso vc seja o criador dessas!

O uso do atributo Deprecated oferece uma **transição suave**, permitindo que:

* Os desenvolvedores sejam notificados para **migrar gradualmente para alternativas**.
* O sistema continue funcionando enquanto o código dependente é atualizado.

# Atributo Override

O atributo **Override** foi introduzido no PHP 8.3 para indicar explicitamente que um método:

1. **Sobrescreve** um método de uma classe pai (ou classe superior).
2. **Implementa** um método definido em uma interface.

Se não existir um método correspondente na classe superior ou na interface implementada:

* Um **erro fatal de compilação** será emitido.

Isso ajuda a evitar:

* **Erros acidentais de sobrescrita**, como erros de digitação no nome do método.
* Problemas em que o código parece estar sobrescrevendo algo, mas, na verdade, não está.

Texto

Descrição gerada automaticamente com confiança baixaA classe Override é definida como **final**, ou seja, não pode ser estendida.

***Método***

* **\_\_construct()**:  
  Construtor padrão que não recebe parâmetros. É usado apenas para marcar explicitamente métodos como sobrescritos.

***Exemplo de Uso***

#### **1. Sobrescrevendo Métodos de uma Classe Pai**

Texto

Descrição gerada automaticamenteAqui está um exemplo correto de uso do atributo Override:

Neste caso:

* O método foo() na classe Extended está sobrescrevendo corretamente o método foo() na classe Base.
* Nenhum erro será emitido.
* Se um método marcado com #[Override] não corresponder a nenhum método na classe superior ou interface implementada, o PHP emitirá um **erro fatal**.

***Por Que Usar o Atributo Override ?***

**1. Prevenção de Erros de Digitação**

Texto

Descrição gerada automaticamenteSem o atributo, é fácil cometer erros acidentais ao tentar sobrescrever métodos. Por exemplo:

Nesse caso, o PHP **não gera erro** sem #[Override], porque o método doSomethng() é tratado como um método novo.

Texto

Descrição gerada automaticamenteCom #[Override], o erro seria detectado em tempo de compilação:

**2. Facilitação de Refatoração**

Se um método na classe pai ou na interface for removido ou renomeado:

* Métodos filhos marcados com #[Override] gerarão um erro, indicando que algo precisa ser ajustado.

Isso é especialmente útil em sistemas grandes, onde mudanças em classes base podem passar despercebidas.

**3. Código Autoexplicativo**

Marcar métodos sobrescritos com #[Override] torna o código mais claro:

* Indica explicitamente que o método implementa ou sobrescreve algo.
* Ajuda outros desenvolvedores a entenderem a relação entre as classes ou interfaces.