UNIPAC
Barbacena

Centro Universitário Presidente Antônio Carlos
Programação para Internet

Exceções

Felipe Roncalli de Paula Carneiro
felipecarneiro@unipac.br

O que vamos aprender nessa aula

- O que é uma Exceção?
- Porque tratar Exceções?
- Exemplo Estrutura Try-Catch
- Como capturar uma Exceção
- Personalizar Mensagens de Erro
- Criando Exceções Personalizadas
- Hierarquia e Tipos de Exceções em PHP

O que é uma Exceção?

Em programação, exceções são eventos anômalos ou situações imprevistas que ocorrem durante a execução de um programa e que interrompem o fluxo normal de execução. Essas situações podem incluir erros de runtime, condições inesperadas ou comportamentos indesejados que não podem ser tratados diretamente pelo código regular do programa.

O que é uma Exceção?

As exceções são usadas para lidar com essas condições excepcionais de maneira estruturada, permitindo que o programa reaja de maneira apropriada e, se possível, se recupere do erro sem encerrar abruptamente. Em vez de simplesmente interromper a execução do programa quando ocorre um problema, o código pode "lançar" uma exceção para indicar que algo deu errado.

Conceitos relacionados a exceções

Lançamento de Exceção (Throw): O ato de sinalizar explicitamente que ocorreu uma condição excepcional. Isso geralmente é feito pelo código quando uma situação imprevista é detectada.

Captura de Exceção (Catch): O bloco de código responsável por lidar com uma exceção lançada. Ele é projetado para tratar a exceção, fornecer feedback adequado ou realizar ações corretivas.

Conceitos relacionados a exceções

Bloco Try-Catch: Uma estrutura que envolve o código que pode gerar exceções. O bloco "try" contém o código suscetível a exceções, enquanto o bloco "catch" trata essas exceções se elas ocorrerem.

Hierarquia de Exceções: Muitas linguagens de programação organizam as exceções em uma hierarquia, onde classes de exceções mais específicas herdam de classes mais genéricas. Isso permite que o código capture exceções de maneira seletiva, lidando com diferentes tipos de erros de forma apropriada.

Porque tratar Exceções?

Ao usar exceções, os programadores podem criar código mais robusto, facilitando a detecção e o tratamento de condições excepcionais, melhorando a legibilidade do código e tornando-o mais fácil de manter. O tratamento adequado de exceções contribui para a resiliência e confiabilidade de um sistema de software.

Exemplos de Exceções

Divisão por zero

```php
<?
$numero = 10;
$divisor = 0;

// Isso gerará uma exceção de divisão por zero.
$resultado = $numero / $divisor;
?>
```

Exemplos de Exceções

Acesso a um Índice Inexistente em um Array:

```php
<?
$arr = [1, 2, 3];

// Isso gerará uma exceção de índice fora dos limites do array.
$valor = $arr[10];
?>
```

Exemplos de Exceções

Chamada de Método ou Função Inexistente:

```php
<?
class MinhaClasse {
    public function meuMetodo() {
        echo "Olá, mundo!";
    }
}

$objeto = new MinhaClasse();
// Isso gerará uma exceção de chamada para um método inexistente.
$objeto->metodoInexistente();

?>
```

Exemplos de Exceções

Erro de Conexão com Banco de Dados:

```php
<?
try {
    $conexao = new PDO("mysql:host=host_invalido;dbname=banco_de_dados", "usuario", "senha");
} catch (PDOException $e) {
    echo "Erro de conexão: " . $e->getMessage();
}
?>
```

Exemplos de Exceções

Manipulação de Arquivos Inexistentes:

```php
<?
$arquivo = 'arquivo_inexistente.txt';

try {
    $conteudo = file_get_contents($arquivo);
} catch (Exception $e) {
    echo "Erro ao ler o arquivo: " . $e->getMessage();
}
cho "Erro de conexão: " . $e->getMessage();
}
?>
```

Exemplos de Exceções

Validação de Entrada de Dados:

```php
<?
function dividir($num1, $num2) {
    if ($num2 == 0) {
        throw new Exception("Divisão por zero não é permitida.");
    }

    return $num1 / $num2;
}
try {
    $resultado = dividir(10, 0);
} catch (Exception $e) {
    echo "Erro: " . $e->getMessage();
}

?>
```

Exemplos de Exceções

Esses são apenas alguns exemplos e a natureza específica das exceções pode variar dependendo da linguagem de programação. O tratamento adequado dessas exceções é essencial para garantir que o programa seja capaz de lidar com condições excepcionais de maneira controlada e robusta.

Estrutura Try-Catch

A estrutura básica do bloco Try-Catch em PHP é projetada para lidar com exceções durante a execução de um bloco de código. O bloco Try-Catch é utilizado para envolver um código suscetível a lançar exceções, permitindo que você capture e lide com essas exceções de maneira controlada.

Estrutura Try-Catch

```php
<?
try {
    // Código que pode gerar exceções
} catch (TipoDeExcecao $excecao) {
    // Código para lidar com a exceção
} catch (OutroTipoDeExcecao $outraExcecao) {
    // Código para lidar com outro tipo de exceção, se necessário
} finally {
    // Código a ser executado sempre, independentemente
    //de ocorrer ou não uma exceção (opcional)
}

?>
```

Estrutura Try-Catch

try: O bloco try envolve o código que pode gerar exceções. Durante a execução deste bloco, se uma exceção ocorrer, o controle é transferido imediatamente para o bloco catch.

catch: O bloco catch é onde você trata a exceção. Ele especifica o tipo de exceção que o bloco irá manipular. Se uma exceção do tipo especificado for lançada no bloco try, o bloco catch será executado. Pode haver vários blocos catch para diferentes tipos de exceções.

Tipo de Exceção: O tipo de exceção indica qual tipo específico de exceção o bloco catch irá lidar. Por exemplo, PDOException para exceções relacionadas ao banco de dados.

Estrutura Try-Catch

excecao: A variável $excecao (ou o nome que você escolher) contém uma instância da exceção capturada. Você pode usar esta variável para acessar informações sobre a exceção, como mensagens de erro.

finally (opcional): O bloco finally é opcional e contém código que será executado independentemente de ocorrer ou não uma exceção. Isso é útil para ações que devem ser realizadas, independentemente de ocorrer uma exceção ou não.

Exemplo Estrutura Try-Catch

```php
<?
try {
    // Código que pode gerar exceções
    $resultado = 10 / 0; // Isso lançará uma exceção de divisão por zero
} catch (Exception $excecao) {
    // Código para lidar com a exceção
    echo "Exceção capturada: " . $excecao->getMessage();
} finally {
    // Código a ser executado sempre, independentemente de ocorrer ou não uma exceção
    echo "Bloco finally executado.";
}
?>
```

Como capturar uma Exceção

O bloco catch é responsável por capturar e lidar com exceções que são lançadas dentro do bloco try. Vamos explorar como o bloco catch funciona com um exemplo prático em PHP.

Suponhamos que temos uma função que realiza uma operação de divisão e queremos lidar com a exceção de divisão por zero:

Como capturar uma Exceção

```php
<?php
function dividir($num1, $num2) {
    try {
        if ($num2 == 0) {
            throw new Exception("Divisão por zero não é permitida.");
        }

        $resultado = $num1 / $num2;
        echo "Resultado: $resultado";
    } catch (Exception $excecao) {
        // Código para lidar com a exceção
        echo "Exceção capturada: " . $excecao->getMessage();
    }
}
// Testando a função com diferentes valores
dividir(10, 2);  // Resultado: 5
dividir(8, 0);   // Exceção capturada: Divisão por zero não é permitida.
?>
```

Como capturar uma Exceção

- A função dividir é definida com um bloco try que contém a operação de divisão.
- Se $num2 for igual a zero, uma exceção do tipo Exception é lançada com uma mensagem específica.
- O bloco catch captura essa exceção, exibindo uma mensagem de erro usando o método getMessage() da instância da exceção ($excecao).

Como capturar uma Exceção

- Na **primeira chamada** (dividir(10, 2)), não há exceção e o resultado é exibido normalmente.
- Na **segunda chamada** (dividir(8, 0)), uma exceção é lançada devido à divisão por zero. O bloco catch é executado, exibindo a mensagem de erro adequada.
- Isso ilustra como o bloco **catch** é usado para capturar exceções específicas e executar um código personalizado para lidar com essas situações excepcionais. Essa abordagem ajuda a manter a robustez do código, proporcionando um controle mais refinado sobre o tratamento de erros durante a execução do programa

Como capturar uma Exceção

- Na **primeira chamada** (dividir(10, 2)), não há exceção e o resultado é exibido normalmente.
- Na **segunda chamada** (dividir(8, 0)), uma exceção é lançada devido à divisão por zero. O bloco catch é executado, exibindo a mensagem de erro adequada.
- Isso ilustra como o bloco **catch** é usado para capturar exceções específicas e executar um código personalizado para lidar com essas situações excepcionais. Essa abordagem ajuda a manter a robustez do código, proporcionando um controle mais refinado sobre o tratamento de erros durante a execução do programa

Personalizar Mensagens de Erro

Personalizar mensagens de erro em exceções é uma prática importante para tornar o código mais legível e facilitar a depuração. Ao fornecer mensagens claras e informativas, você ajuda os desenvolvedores a entenderem o contexto do erro e a tomarem medidas adequadas para corrigi-lo. Vamos ver como personalizar mensagens de erro em exceções em PHP:

Personalizar Mensagens de Erro

Personalizar mensagens de erro em exceções é uma prática importante para tornar o código mais legível e facilitar a depuração. Ao fornecer mensagens claras e informativas, você ajuda os desenvolvedores a entenderem o contexto do erro e a tomarem medidas adequadas para corrigi-lo. Vamos ver como personalizar mensagens de erro em exceções em PHP:

Personalizar Mensagens de Erro

```php
<?php
function dividir($num1, $num2) {
    try {
        if ($num2 == 0) {
            throw new Exception("Não é possível realizar a divisão. O divisor não pode ser zero.");
        }

        $resultado = $num1 / $num2;
        echo "Resultado: $resultado";
    } catch (Exception $excecao) {
        // Código para lidar com a exceção
        echo "Exceção capturada: " . $excecao->getMessage();
    }
}

// Testando a função com diferentes valores
dividir(10, 2);  // Resultado: 5
dividir(8, 0);   // Exceção capturada:
//Não é possível realizar a divisão. O divisor não pode ser zero.
?>
```

Personalizar Mensagens de Erro

Neste exemplo, a mensagem de erro na exceção foi personalizada para incluir informações mais específicas sobre a natureza do problema. Aqui estão algumas práticas recomendadas ao personalizar mensagens de erro em exceções:

Personalizar Mensagens de Erro

1. Seja Descritivo: A mensagem de erro deve ser descritiva o suficiente para indicar claramente o motivo da exceção. Evite mensagens genéricas que não ofereçam insights úteis.
2. Forneça Contexto: Inclua informações relevantes no contexto da mensagem. No exemplo, a mensagem indica que a divisão não pode ser realizada porque o divisor não pode ser zero.
3. Seja Consciente da Segurança: Evite incluir informações sensíveis ou detalhes internos do sistema nas mensagens de erro, para evitar possíveis riscos de segurança.
4. Consistência: Mantenha uma abordagem consistente ao personalizar mensagens de erro em seu código para facilitar a manutenção e a compreensão global.

Criando Exceções Personalizadas

Vamos criar um exemplo de exceção personalizada para validação de entrada em PHP. Suponha que você esteja desenvolvendo uma função que realiza uma operação específica e deseja garantir que os parâmetros de entrada atendam a certos critérios. Neste caso, podemos criar uma exceção personalizada para lidar com a validação de entrada.

Criando Exceções Personalizadas

```php
<?php
class EntradaInvalidaException extends Exception {
    public function __construct($mensagem = "Entrada inválida.", $codigo = 0, Exception $anterior = null) {
        parent::__construct($mensagem, $codigo, $anterior);
    }

    // Personalize métodos conforme necessário
    public function mensagemPersonalizada() {
        return 'Erro: ' . $this->getMessage();
    }
}
function realizarOperacao($valor) {
    try {
        if (!is_numeric($valor) || $valor <= 0) {
            throw new EntradaInvalidaException("O valor deve ser um número positivo.");
        }

        // Lógica da operação aqui...

        echo "Operação realizada com sucesso!";
    } catch (EntradaInvalidaException $excecao) {
        // Código para lidar com a exceção personalizada
        echo $excecao->mensagemPersonalizada();
    }
}
// Testando a função com diferentes valores
realizarOperacao(5);     // Operação realizada com sucesso!
realizarOperacao(-2);    // Erro: O valor deve ser um número positivo.
realizarOperacao("texto"); // Erro: O valor deve ser um número positivo.
?>
```

Criando Exceções Personalizadas

- EntradaInvalidaException é uma classe que estende a classe base Exception. Ela pode ser personalizada conforme necessário, adicionando métodos ou propriedades específicos.
- O construtor da exceção é personalizado para aceitar uma mensagem padrão (definida como "Entrada inválida."), um código opcional e uma exceção anterior.

Criando Exceções Personalizadas

- A função realizarOperacao verifica se o valor passado como parâmetro é um número positivo. Se a validação falhar, ela lança a exceção EntradaInvalidaException com uma mensagem específica.
- O bloco catch captura essa exceção personalizada e utiliza o método mensagemPersonalizada() para exibir uma mensagem mais amigável.

Ao criar exceções personalizadas, você torna o código mais modular e legível, permitindo uma manipulação mais específica e eficaz de situações excepcionais relacionadas à validação de entrada.

Hierarquia e Tipos de Exceções em PHP

Em PHP, as exceções são organizadas em uma hierarquia, o que significa que existem classes de exceção mais genéricas e classes mais específicas que herdam delas. Isso permite um tratamento mais refinado de diferentes tipos de erros. Aqui estão algumas das classes de exceção mais comuns na hierarquia de exceções em PHP:

Hierarquia e Tipos de Exceções em PHP

Exception:

- A classe base para todas as exceções em PHP. Pode ser usada diretamente, mas geralmente é mais eficaz criar exceções mais específicas para diferentes situações.

ErrorException:

- Representa erros lançados por funções internas do PHP. Herda de Exception.

LogicException:

- Representa exceções de lógica que ocorrem durante a execução do programa. Exemplos incluem tentativas de chamar um método em uma classe que não existe. Herda de Exception.

Hierarquia e Tipos de Exceções em PHP

RuntimeException:

- Representa exceções de tempo de execução mais gerais. Herda de Exception.

InvalidArgumentException:

- Indica que um argumento fornecido para uma função ou método não é do tipo esperado. Herda de LogicException.

OutOfBoundsException:

- Indica que uma tentativa foi feita para acessar um índice de matriz, caractere de string ou posição em uma estrutura de dados que não existe.

Hierarquia e Tipos de Exceções em PHP

DivisionByZeroError:

- Indica uma tentativa de dividir um número por zero. Herda de ArithmeticError.

ParseError:

- Representa erros de análise (parse) de código fonte. Herda de Error.

TypeError:

- Indica que uma operação foi realizada com um tipo de dado incompatível. Herda de Error.

Hierarquia e Tipos de Exceções em PHP

Essa é apenas uma pequena amostra da hierarquia de exceções em PHP. Ao criar exceções personalizadas, você pode estender as classes existentes para fornecer detalhes específicos sobre o tipo de erro que ocorreu em sua aplicação. Isso ajuda a estruturar o código de tratamento de exceções de forma mais granular e a fornecer mensagens de erro mais informativas durante a depuração.

Hierarquia e Tipos de Exceções em PHP

As exceções em PHP podem ser classificadas em duas categorias principais: exceções padrão (built-in) e exceções personalizadas, definidas pelo usuário.

Considerações Gerais

Equilíbrio:

- Geralmente, uma boa prática é usar exceções padrão para casos gerais e exceções personalizadas para casos específicos. Isso proporciona um equilíbrio entre a reutilização de exceções já definidas e a capacidade de personalizar o tratamento de erros conforme necessário.

Documentação Adequada:

- Independentemente do tipo de exceção, é crucial fornecer uma documentação adequada que explique quando e por que uma exceção é lançada, bem como a maneira correta de tratá-la.

Dúvidas??