Templates em C++

Especificação de Exceção Dinâmica em C++

Introdução

A especificação de exceção dinâmica em C++ foi uma forma de declarar quais exceções uma função poderia lançar. No entanto, essa característica foi descontinuada no C++11 e removida no C++17. Em vez disso, o especificador noexcept é utilizado para indicar que uma função não lança exceções.

1. Definição e Sintaxe

- Definição: A especificação de exceção dinâmica declarava as exceções que uma função poderia lançar usando a sintaxe throw(T1, T2, ...). No C++11 e posterior, o uso de noexcept é recomendado.
- Sintaxe:

```
void func() throw(T1, T2);
```

2. Uso Básico da Especificação de Exceção Dinâmica

- Definição: Declarar as exceções que uma função pode lançar usando throw.
- Exemplo:

```
void func() throw(std::runtime_error, std::logic_error) {
    // Código que pode lançar std::runtime_error ou std::logic_error
}
```

```
int main() {
    try {
        func();
    } catch (const std::runtime_error& e) {
        std::cout << "Capturou std::runtime_error: " << e.what() << std::endl;
    } catch (const std::logic_error& e) {
        std::cout << "Capturou std::logic_error: " << e.what() << std::endl;
    }
    return 0;
}
```

3. Especificação de Exceção Vazia

- Definição: Declarar que uma função não lança exceções usando throw().
- Exemplo:

```
void func() throw() {
    // Código que não lança exceções
}

int main() {
    try {
        func();
    } catch (...) {
        std::cout << "Isso nunca será impresso" << std::endl;
```

```
    }
    return 0;
}
```

4. Especificação de Exceção em Funções Membro

- Definição: Aplicar a especificação de exceção dinâmica a funções membro, incluindo construtores e destrutores.
- Exemplo:

```
class Exemplo {
public:
    Exemplo() throw(std::runtime_error) {
        // Construtor que pode lançar std::runtime_error
    }

    ~Exemplo() throw() {
        // Destrutor que não lança exceções
    }

    void metodo() throw(std::logic_error) {
        // Método que pode lançar std::logic_error
    }
};

int main() {
```

```
    try {
        Exemplo e;
        e.metodo();
    } catch (const std::runtime_error& e) {
        std::cout << "Capturou std::runtime_error: " << e.what() << std::endl;
    } catch (const std::logic_error& e) {
        std::cout << "Capturou std::logic_error: " << e.what() << std::endl;
    }
    return 0;
}
```

5. Funções Virtuais e Especificação de Exceção

- Definição: Especificar exceções para funções virtuais.
- Exemplo:

```
class Base {
public:
    virtual void func() throw(std::runtime_error) {
        // Função virtual que pode lançar std::runtime_error
    }
};

class Derivada : public Base {
public:
    void func() throw(std::runtime_error) override {
```

```
        // Função sobrescrita que pode lançar std::runtime_error
    }
};

int main() {
    Base* b = new Derivada();
    try {
        b->func();
    } catch (const std::runtime_error& e) {
        std::cout << "Capturou std::runtime_error: " << e.what() << std::endl;
    }
    delete b;
    return 0;
}
```

6. Depreciação e Remoção da Especificação de Exceção Dinâmica

- Definição: A especificação de exceção dinâmica foi descontinuada no C++11 e removida no C++17. É recomendável usar noexcept em vez disso.
- Exemplo:

```
void func() noexcept {
    // Função que não lança exceções
}

int main() {
```

```
    try {
        func();
    } catch (...) {
        std::cout << "Isso nunca será impresso" << std::endl;
    }
    return 0;
}
```

7. Conversão Automática para noexcept(false)

- Definição: No C++11 e posterior, funções com especificação de exceção dinâmica são tratadas como noexcept(false).
- Exemplo:

```
void func() throw() {
    // Tratada como noexcept(true)
}

void outraFunc() throw(std::runtime_error) {
    // Tratada como noexcept(false)
}

int main() {
    std::cout << std::boolalpha;
    std::cout << noexcept(func()) << std::endl;        // Saída: true
    std::cout << noexcept(outraFunc()) << std::endl;   // Saída: false
```

```
    return 0;
}
```

Dicas de Boas Práticas

- Evite Especificação de Exceção Dinâmica: Use noexcept em vez de especificação de exceção dinâmica para funções que não lançam exceções.
- Clareza e Manutenção: Mantenha o uso de noexcept claro e bem documentado para facilitar a leitura e a manutenção do código.
- Verificação de noexcept: Use a expressão noexcept para verificar se uma função é noexcept, garantindo a correção do programa.

Esta seção abrange os conceitos sobre a especificação de exceção dinâmica em C++. Para mais detalhes, consulte a documentação oficial: https://en.cppreference.com/w/cpp/language/except_spec