Программирование на языке C++
Лекция 10

Гарантии безопасности исключений

Александр Смаль

Гарантии безопасности исключений

Гарантия отсутствия исключений

“Ни при каких обстоятельствах функция не будет генерировать исключения”.

Гарантии безопасности исключений

Гарантия отсутствия исключений

“Ни при каких обстоятельствах функция
не будет генерировать исключения”.

Базовая гарантия

“При возникновении любого исключения
состояние программы останется согласованным”.

Гарантии безопасности исключений

Гарантия отсутствия исключений

“Ни при каких обстоятельствах функция
не будет генерировать исключения”.

Базовая гарантия

“При возникновении любого исключения
состояние программы останется согласованным”.

Строгая гарантия

“Если при выполнении операции возникнет исключение,
то программа останется том же в состоянии,
которое было до начала выполнения операции”.

Строгая гарантия безопасности исключений

Строгая гарантия безопасности исключений

- В каком случае мы не можем обеспечить строгую гарантию безопасности исключений?

Строгая гарантия безопасности исключений

- В каком случае мы не можем обеспечить строгую гарантию безопасности исключений?

  При наличии взаимодействия со внешним окружением.

Строгая гарантия безопасности исключений

- В каком случае мы не можем обеспечить строгую гарантию безопасности исключений?

  При наличии взаимодействия со внешним окружением.

- Как обеспечить строгую гарантию безопасности исключений в остальных случаях?

Строгая гарантия безопасности исключений

- В каком случае мы не можем обеспечить строгую гарантию безопасности исключений?

  При наличии взаимодействия со внешним окружением.

- Как обеспечить строгую гарантию безопасности исключений в остальных случаях?

  Выполнять операцию над копией состояния программы.
  Если операция прошла успешно, заменить состояние на копию.

Строгая гарантия безопасности исключений

- В каком случае мы не можем обеспечить строгую гарантию безопасности исключений?

  При наличии взаимодействия со внешним окружением.

- Как обеспечить строгую гарантию безопасности исключений в остальных случаях?

  Выполнять операцию над копией состояния программы.
  Если операция прошла успешно, заменить состояние на копию.

- Когда можно обеспечить строгую гарантию *эффективно*?

Строгая гарантия безопасности исключений

- В каком случае мы не можем обеспечить строгую гарантию безопасности исключений?

  При наличии взаимодействия со внешним окружением.

- Как обеспечить строгую гарантию безопасности исключений в остальных случаях?

  Выполнять операцию над копией состояния программы.
  Если операция прошла успешно, заменить состояние на копию.

- Когда можно обеспечить строгую гарантию *эффективно*?

  Это вопрос архитектуры приложения.

Как добиться строгой гарантии?

```
template<class T>
struct Array
{
    void resize(size_t n)
    {
        T * ndata = new T[n];
        for (size_t i = 0; i != n && i != size_; ++i)
            ndata[i] = data_[i];

        delete [] data_;
        data_ = ndata;
        size_ = n;
    }

    T *     data_;
    size_t  size_;
};
```

Как добиться строгой гарантии: вручную

```
template<class T>
struct Array
{
    void resize(size_t n) {
        T * ndata = new T[n];
        try {
            for (size_t i = 0; i != n && i != size_; ++i)
                ndata[i] = data_[i];
        } catch (...) {
            delete [] ndata;
            throw;
        }
        delete [] data_;
        data_ = ndata;
        size_ = n;
    }

    T *     data_;
    size_t  size_;
};
```

Как добиться строгой гарантии: RAII

```
template<class T>
struct Array
{
    void resize(size_t n) {
        unique_ptr<T[]> ndata(new T[n]);

        for (size_t i = 0; i != n && i != size_; ++i)
            ndata[i] = data_[i];

        data_ = std::move(ndata);
        size_ = n;
    }

    unique_ptr<T[]> data_;
    size_t          size_;
};
```

Как добиться строгой гарантии: swap

```
template<class T>
struct Array
{
    void resize(size_t n) {
        Array t(n);
        for (size_t i = 0; i != n && i != size_; ++i)
            t[i] = data_[i];

        t.swap(*this);
    }

    T       * data_;
    size_t    size_;
};
```

Проектирование с учётом исключений

Рассмотрим традиционный интерфейс стека:

```
template<class T>
struct Stack
{
    void push(T const& t)
    {
        data_.push_back(t);
    }

    T pop()
    {
        T tmp = data_.back();
        data_.pop_back();
        return tmp;
    }

    std::vector<T> data_;
};
```

Проектирование с учётом исключений

Рассмотрим традиционный интерфейс стека:

```
template<class T>
struct Stack
{
    void push(T const& t)
    {
        data_.push_back(t);
    }

    void pop(T & res)
    {
        res = data_.back();
        data_.pop_back();
    }

    std::vector<T> data_;
};
```

Использование unique_ptr

```
template<class T>
struct Stack
{
    void push(T const& t)
    {
        data_.push_back(t);
    }

    unique_ptr<T> pop()
    {
%        unique_ptr<T> tmp(new T(data_.back()));
        data_.pop_back();
        return std::move(tmp);
    }

    std::vector<T> data_;
};
```

Заключение

- Проектируйте архитектуру приложения с учётом ислючений.

Заключение

- Проектируйте архитектуру приложения с учётом ислючений.
- Функции, не бросающие исключения, нужно объявлять как `noexcept`.

Заключение

- Проектируйте архитектуру приложения с учётом ислючений.
- Функции, не бросающие исключения, нужно объявлять как `noexcept`.
- Все использующие исключения функции должны обеспечивать как минимум базовую гарантию безопасности исключений.

Заключение

- Проектируйте архитектуру приложения с учётом ислючений.
- Функции, не бросающие исключения, нужно объявлять как `noexcept`.
- Все использующие исключения функции должны обеспечивать как минимум базовую гарантию безопасности исключений.
- Там, где это возможно, старайтесь обеспечить строгую гарантию безопасности исключений.

Заключение

- Проектируйте архитектуру приложения с учётом ислючений.
- Функции, не бросающие исключения, нужно объявлять как `noexcept`.
- Все использующие исключения функции должны обеспечивать как минимум базовую гарантию безопасности исключений.
- Там, где это возможно, старайтесь обеспечить строгую гарантию безопасности исключений.
- Используйте `swap`, умные указатели и другие RAII объекты для обеспечения строгой безопасности исключений.