Обзор объектно-ориентированных методов в C#

30.04.2025

В C#определение типа — класса, структуры или записи — это схема, указывающая, что может сделать тип. Объект в основном является блоком памяти, выделенным и настроенным в соответствии с схемой. В этой статье представлен обзор этих схем и их функций. В следующей статье этой серии представлены объекты.

Инкапсуляция

Инкапсуляция иногда называется первой основой или принципом объектно-ориентированного программирования. Класс или структура может указать, насколько доступен каждый из его членов для кода вне класса или структуры. Элементы, не предназначенные для потребителей за пределами класса или сборки, скрыты, чтобы ограничить потенциал ошибок кодирования или вредоносных эксплойтов. Дополнительные сведения см. в руководстве по программированию на основе объектов .

Члены

Члены типа включают все методы, поля, константы, свойства и события. В C#глобальные переменные или методы отсутствуют, так как в некоторых других языках нет. Даже точка входа программы, метод `Main`, должна быть объявлена в классе или структуре (неявно при использовании операторов верхнего уровня).

Следующий список содержит все различные виды элементов, которые можно объявить в классе, структуре или записи.

- Поля
- Константы
- Свойства
- Методы
- Конструкторы
- События
- Финализаторы
- Индексаторы
- Операторы
- Вложенные типы

Дополнительные сведения см. в разделе "Участники".

Доступность

Некоторые методы и свойства предназначены для вызова или доступа из кода за пределами класса или структуры, известного как *клиентский код*. Другие методы и свойства могут использоваться только в классе или структуре. Важно ограничить доступность кода, чтобы достичь его только предполагаемого клиентского кода. Вы указываете, как доступны типы и их члены для клиентского кода с помощью следующих модификаторов доступа:

- общественный
- защищенный
- внутренний
- защищенный внутренний
- частный
- закрытый защищенный.

Доступ по умолчанию — это `private`.

Наследство

Классы (но не структуры) поддерживают концепцию наследования. Класс, производный от другого класса, автоматически содержит все общедоступные, защищенные и внутренние члены базового класса, кроме его конструкторов и финализаторов.

Классы могут быть объявлены абстрактными, что означает, что один или несколько их методов не имеют реализации. Хотя абстрактные классы нельзя создать напрямую, они могут служить базовыми классами для других классов, которые обеспечивают недостающую реализацию. Классы также можно объявить как запечатанные , чтобы предотвратить наследование других классов от них.

Дополнительные сведения см. в разделе "Наследование и полиморфизм".

Интерфейсы

Классы, структуры и записи могут реализовать несколько интерфейсов. Для реализации из интерфейса означает, что тип реализует все методы, определенные в интерфейсе. Дополнительные сведения см. в разделе "Интерфейсы".

Универсальные типы

Классы, структуры и записи можно определить с помощью одного или нескольких параметров типа. Клиентский код предоставляет тип при создании экземпляра типа. Например, List<T> класс в System.Collections.Generic пространстве имен определяется одним параметром типа. Клиентский код создает экземпляр `List<string>` или `List<int>`, чтобы указать тип, который содержит список. Дополнительные сведения см. в разделе «Обобщения».

Статические типы

Классы (но не структуры или записи) можно объявить как `static`. Статический класс может содержать только статические элементы и не может быть создан с помощью ключевого `new` слова. Одна копия класса загружается в память при загрузке программы, а его члены получают доступ через имя класса. Классы, структуры и записи могут содержать статические элементы. Дополнительные сведения см. в разделе "Статические классы" и "статические члены классов".

Вложенные типы

Класс, структуру или запись можно вложить в другой класс, структуру или запись. Дополнительные сведения см. в разделе "Вложенные типы".

Частичные типы

Вы можете определить часть класса, структуры или метода в одном файле кода и другую часть в отдельном файле кода. Дополнительные сведения см. в разделе "Частичные классы и методы".

Инициализаторы объектов

Можно создать экземпляр и инициализировать объекты класса или структуры, а также коллекции объектов, назначив значения своим свойствам. Дополнительные сведения см. в разделе "Как инициализировать объекты с помощью инициализатора объектов".

Анонимные типы

В ситуациях, когда неудобно или необязательно создавать именованный класс, используются анонимные типы. Элементы данных с именами определяют анонимные типы. Дополнительные сведения см. в разделе "Анонимные типы".

Члены расширения

Класс можно расширить, не создавая производный класс, создавая отдельный тип. Этот тип содержит методы, которые можно вызывать, как если бы они принадлежали исходному типу. Дополнительные сведения см. в разделе "Методы расширения".

Неявно типизированные локальные переменные

В методе класса или структуры можно использовать неявную типизацию, чтобы компилятор определил тип переменной во время компиляции. Дополнительные сведения см. в разделе var (справочник по C#).

Записи

Модификатор `record` можно добавить в класс или структуру. Типы записей имеют встроенное поведение для обеспечения равенства на основе значений. Запись (либо `record class` или `record struct`) предоставляет следующие возможности:

- Краткий синтаксис для создания ссылочного типа с неизменяемыми свойствами.
- Цените равенство. Две переменные типа записи равны, если они имеют одинаковый тип, и если для каждого поля значения в обеих записях равны. Классы используют равенство ссылок: два переменных типа класса равны, если они ссылаются на один и тот же объект.
- Краткий синтаксис для недеструктивной мутации. Выражение `with` позволяет создать новый экземпляр записи, который является копией существующего экземпляра, но с изменёнными указанными значениями свойств.
- Встроенное форматирование для отображения. Метод `ToString` выводит имя типа записи и имена и значения общедоступных свойств.
- Поддержка иерархий наследования в классах записей. Классы записей поддерживают наследование. Структуры записей не поддерживают наследование.

Дополнительные сведения см. в разделе "Записи".

Спецификация языка C#

Дополнительные сведения см. в спецификации языка C#. Спецификация языка является авторитетным источником синтаксиса и использования языка C#.