JS

ECMAScript 5. Объектно-ориентированное программирование

Инструктор: Максим

Содержание

1. Объектно-ориентированное программирование
2. Объекты
   - Создание
   - Свойства
   - Методы
   - Синтаксис массивов
   - Упрощенный синтаксис создания
   - Удаление свойств и методов
   - Вложенные
   - Массивы в объектах
   - Проверка наличия свойств и методов
   - Перебор свойств и методов

1. Объектно-ориентированное программирование. Концепция

Каждый стиль программирования имеет свою **концептуальную** основу и требует уникального подхода к решению задачи. Основой ООП стиля является **объектная модель**. Эта модель состоит из следующих четырех **главных** элементов:

- **Абстракция**
- **Инкапсуляция**
- **Модульность**
- **Иерархия**

Кроме главных, в этой модели существуют еще три **дополнительных** элемента:

- **Контроль типов**
- **Параллелизм**
- **Персистентность**

1. Объектно-ориентированное программирование. Концепция. Абстракция

Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко описывает его концептуальные границы с точки зрения наблюдателя

Абстракция концентрирует внимание на **внешнем представлении объекта** и позволяет отделить существенные особенности поведения от их реализации

Существует целый спектр абстракций, начиная с объектов, почти точно соответствующих реалиям предметной области, и заканчивая совершенно излишними объектами, существуют следующие абстракции:

- **Абстракция сущности** – объект, представляющий собой полезную модель некой сущности в предметной области
- **Абстракция действия** – объект, состоящий из обобщенного множества методов, каждый из которых выполняет однотипные функции
- **Произвольная абстракция** – объект, включающий в себя набор методов, не имеющих друг с другом ничего общего

1. Объектно-ориентированное программирование. Концепция. Инкапсуляция

Абстракция и **инкапсуляция** дополняют друг друга. В центре внимания **абстракции** находится наблюдаемое **поведение** объекта, а **инкапсуляция** сосредоточена на **реализации**, обеспечивающей заданное поведение. Как правило, инкапсуляция осуществляется с помощью сокрытия информации, т.е. утаивания всех несущественных деталей объекта. Обычно скрывается как структура объекта, так и реализация методов

Никакая часть сложной системы не должна зависеть от внутреннего устройства какой-либо другой части. В то время как **абстракция помогает думать о том, что они делают**, **инкапсуляция позволяет легко перестраивать программы**

Для того чтобы абстракция работала, ее реализация должна быть инкапсулирована, т.е. каждый **класс** должен состоять из двух частей: **интерфейса** и **реализации**

1. Объектно-ориентированное программирование. Концепция. Модульность

Модульность – это разделение программы на фрагменты, которые компилируются по отдельности, но связаны между собой

В большинстве языков, поддерживающих принцип модульности, интерфейс модуля отделен от его реализации, т.е. можно сказать, что **модульность** и **инкапсуляция** тесно **связаны** между собой

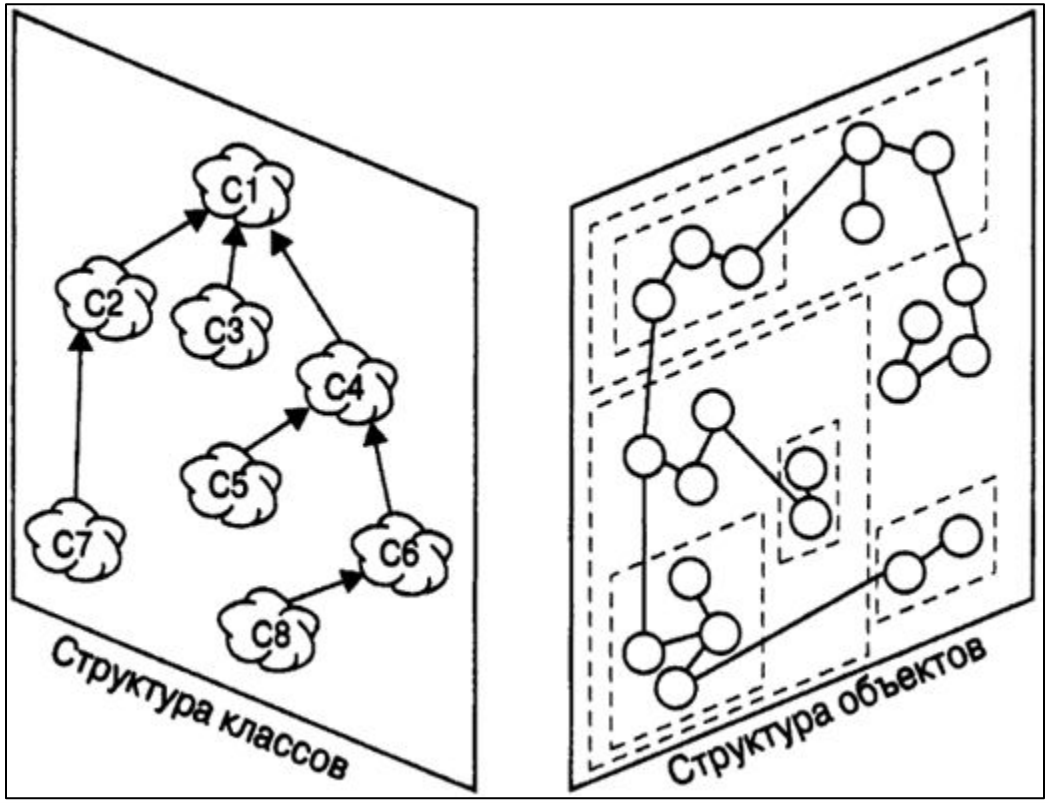
Модули служат физическими **контейнерами**, содержащими классы, возникающие при логическом проектировании системы

Модули, как правило, являются элементарными и неделимыми структурными элементами программного обеспечения и могут использоваться в приложениях неоднократно, разделение программы на модули должно допускать повторное использование кода

1. Объектно-ориентированное программирование. Концепция. Иерархия

Иерархия – это упорядочивание абстракций. Наиболее **важные** виды иерархии:

- **Наследование** – иерархия **общее/частное**, которая описывает структуру **классов** и основана на отношении – **является** (is-a)
- **Агрегация** – иерархия **целое/часть**, которая описывает структуру **объектов** и основана на отношении – **является частью** (has-a)



1. Объектно-ориентированное программирование. Концепция. Контроль типов

Тип – это точная характеристика структуры и поведения, присущих некоторой совокупности объектов. Термины **тип** и **класс** являются **синонимами**

Контроль типов – это правило использования объектов, не допускающие или ограничивающие взаимную замену объектов разных классов

С одной стороны контроль типов бывает:

- **Статический** – переменная, параметр метода, возвращаемое значение метода связывается с типом в момент объявления и тип не может быть изменён позже
- **Динамический** – переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной

С другой стороны: **строгий** и **слабый**. Эти термины не являются однозначно трактуемыми, и чаще всего используются для указания на достоинства и недостатки конкретного языка. Языки со строгим контролем типов, не позволяют нарушать соответствия между типами, например, невозможно вызвать метод с параметрами, тип которых отличается от определенных в методе, подать больше или меньше параметров в метод чем в нем определенно. Языки со слабым контролем типом наоборот, позволяют программистам нарушать соответствия между типами

1. Объектно-ориентированное программирование. Концепция. Параллелизм

Параллелизм делает акцент на абстракции и синхронизации процессов

Каждый объект может представлять собой отдельный **поток управления** (абстракция процесса), такой объект называется активный

Система, построенная на основе ООП, представлена как совокупность взаимодействующих объектов, часть из которых являются **активными** и независимыми от других

Из этого можно сделать вывод, что параллелизм выделяет активные объекты от пассивных

1. Объектно-ориентированное программирование. Концепция. Персистентность

Любой программный **объект занимает** определенный объем **памяти** и **существует** определенное **время**

Множество **способов существования** объекта неисчерпаемо: от промежуточных объектов, возникающих лишь во время вычисления выражений, до постоянных баз данных, существующих даже после завершения работы программы

Персистентность – это способность объекта преодолевать **временные** рамки (т.е. продолжать свое существование после исчезновения своего создателя) или **пространственные** переделы (т.е. выходить за пределы своего первоначального адресного пространства)

1. Объектно-ориентированное программирование. Объект. Состояние

Основными конструктивными элементами ООП являются **классы** и **объекты**

Объект обладает **состоянием** и **поведением**

Состояние объекта характеризуется перечнем всех (как правило, статических) **полей** (атрибутов, свойств) данного объекта и текущими (как правило, динамическими) **значениями** каждого из этих полей

Полями объекта называются отличительные характеристики, черты, качества или особенности, обеспечивающие его уникальность

Значения полей могут быть простыми **количественными характеристиками** или **обозначать другой объект**

1. Объектно-ориентированное программирование. Объект. Поведение

Поведение объекта характеризуется перечнем всех **методов** (операций, функций, процедур) данного объекта

Методом называется определенное **воздействие одного объекта на другой** с целью вызвать некую реакцию, т.е. один объект может вызывать методы у других объектов (посылать сообщения) и отвечать на вызванные у него методы другими объектами (реагировать на сообщения)

Все методы можно разделить на следующие виды:

- **Модификатор** – изменение состояния объекта
- **Селектор** – получение состояния объекта
- **Итератор** – получение всего состояния объекта в строго определенном порядке
- **Конструктор** – создает объект и/или инициализирует его состояние
- **Деструктор** – удаляет объект и/или стирает состояние объекта

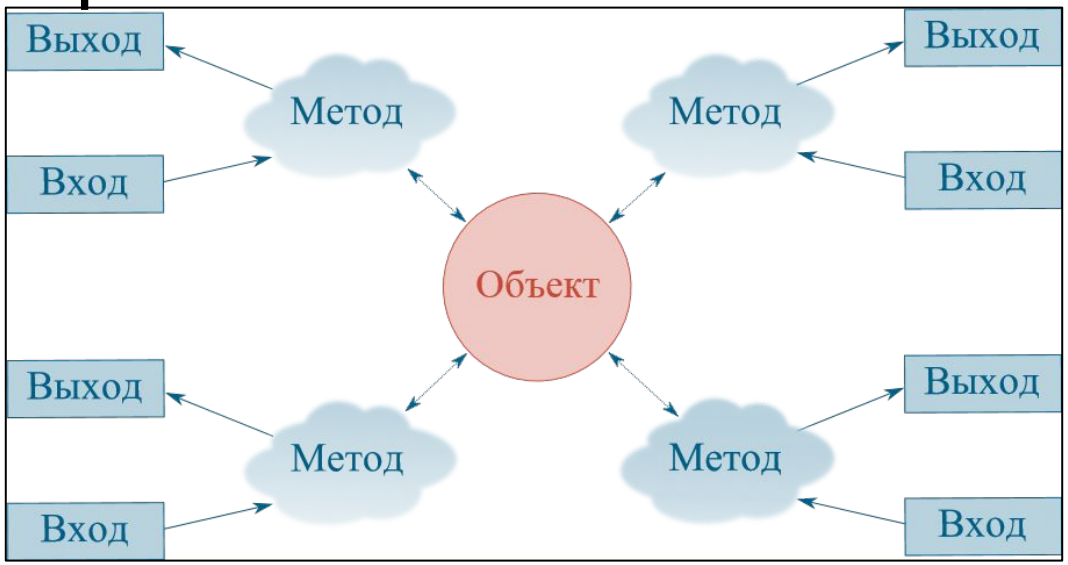
1. Объектно-ориентированное программирование. Объект

Состояние объекта представляет собой **суммарный результат его поведения**

Ни один объект не существует в изоляции от других. Все объекты либо подвергаются воздействию, либо сами воздействуют на другие объекты

Структура и **поведение схожих** объектов **определяется** в общем для них **классе**

Термины **экземпляр** и **объект** являются **синонимами**



1. Объектно-ориентированное программирование. Объект. Отношения

Сами по себе объекты совершенно не интересны. Поведение системы определяется лишь **взаимодействием между объектами**

Отношения между двумя объектами основываются на взаимных предположениях о допустимых методах и ожидаемом поведении:

- **Ассоциация/Связь** – объект **сотрудничает** с другими объектами, используя связи с ними, другими словами, связь – это конкретное соединение, через которое объект (клиент) запрашивает услугу (вызывает метод) у другого объекта (сервера) или управляет им
- **Агрегация** – описывает иерархию **целое/часть** и позволяет переходить от целого (агрегата) к его частям (компонентам). Объект, представляющий собой часть другого объекта, имеет связь со своим агрегатом. Используя эту связь, агрегат может посылать сообщения (вызывать методы) своим частям. Время жизни частей и время жизни целого не зависят друг от друга
- **Композиция** – похоже на агрегация, только **более сильная связь**. Объект не только является частью другого объекта, но и вообще не может принадлежат еще кому-то. Время жизни частей зависит от времени жизни целого
- **Делегирование** – объект **поручает реализацию** своего поведения **другим** объектам

1. Объектно-ориентированное программирование. Класс

В то время как объект обозначает конкретную сущность, определенную во времени и в пространстве, **класс описывает** совокупность объектов, обладающих общей структурой и одинаковым поведением

Класс – это множество объектов, обладающих общей структурой, поведением и семантикой

Отдельный **объект** является просто **экземпляром класса**

Существуют следующие **отношения между двумя классами**:

- **Наследование** – показывает, что один из двух связанных классов (подтип, наследник) является частной формой другого (надтип, родитель), который называется обобщением первого
- **Реализация** – отношение между двумя классами, в которой один класс реализует поведение, заданное другим

Обратите внимание, отношение **делегирование** и **наследование** являются **альтернативой** друг другу

2. Объекты

Во многих классических объектно-ориентированных языках, как C++, Java, C# вся концепция ООП основана вокруг понятий **класса** и **объекта**. Класс представляет некий план объекта, а объект – конкретную реализацию этого плана

В JavaScript до принятия стандарта ECMAScript 2015 не использовались классы, а использовались **прототипы** объектов

Если в программе нужно описать сущность человека, у которого есть имя, возраст, пол и так далее, то естественно не получится представить сущность человека в виде простого типа данных, числа или строки. Потребуется несколько строк или чисел, чтобы должным образом описать человека. В этом плане человек будет выступать как сложная комплексная структура, у которого будут отдельные свойства – возраст, рост, имя, фамилия и т.д.

Для работы с подобными структурами в JavaScript используются **объекты**. Каждый объект может хранить **свойства**, которые описывают его **состояние**, и **методы**, которые описывают его **поведение**

2. Объекты. Создание

Есть **несколько способов** создания нового объекта

Первый способ заключается в использовании **конструктора Object()**:

```
var user = new Object();
```

В данном случае объект называется user. Он определяется также, как и любая обычная переменная с помощью ключевого слова var

Выражение new Object() представляет вызов конструктора – функции, создающей новый объект. Для вызова конструктора применяется оператор **new**. Вызов конструктора фактически напоминает вызов обычной функции

Второй способ создания объекта представляет использование **фигурных скобок**:

```
var user = {};
```

Обратите внимание, второй способ является более распространенным

2. Объекты. Свойства

После создания объекта можно определить в нем свойства. Чтобы **определить свойство**, после названия объекта **через точку** нужно указать **имя** свойства и присвоить ему **значение**:

```
var user = {};
user.name = "Tom";
user.age = 26;
```

В данном случае объявляются два свойства name и age, которым присваиваются соответствующие значения. После этого можно использовать эти свойства, например, вывести их значения в консоли:

```
console.log(user.name); // "Tom"
console.log(user.age); // 26
```

2. Объекты. Методы

Методы объекта определяют его поведение или действия, которые он производит. **Методы представляют собой функции**. Например, определим метод, который бы выводил имя и возраст человека:

```
var user = {};
user.name = "Tom";
user.age = 26;
user.display = function () {
    console.log(user.name);
    console.log(user.age);
};
user.display(); // Вызов метода
```

Как и в случае с функциями методы сначала определяются, а потом уже вызываются

2. Объекты. Синтаксис массивов

Существует также альтернативный способ определения свойств и методов с помощью **синтаксиса массивов**:

```
var user = {};
user["name"] = "Tom";
user["age"] = 26;
user["display"] = function () {
    console.log(user.name);
    console.log(user.age);
};
user["display"](); // Вызов метода
```

Название каждого свойства или метода заключается **в кавычки и в квадратные скобки**, затем им также присваивается значение. Например, user["age"] = 26

При обращении к этим свойствам и методам можно использовать либо нотацию точки (user.name), либо обращаться так: user["name"]

2. Объекты. Упрощенный синтаксис создания

Вместо отдельного определения всех свойств и методов можно **определить** их **все сразу**:

```
var user = {
    name: "Tom",
    age: 26,
    display: function () {
        console.log(user.name);
        console.log(user.age);
    }
};
user.display(); // Вызов метода
```

При таком способе определения объекта в качестве символа **присвоения** значения используется **не знак равно**, **а двоеточие**. Например, выражение name: "Tom" присваивает свойству name строку Tom. При определении методов также используется знак двоеточия. После определения отдельных свойств и методов идет не точка запятой, а запятая

2. Объекты. Упрощенный синтаксис создания

Также следует отметить, что названия **свойств** и **методов** объекта всегда **представляют строки**. То есть, предыдущее определение объекта можно переписать так:

```
var user = {
    "name": "Tom",
    "age": 26,
    "display": function () {
        console.log(user.name);
        console.log(user.age);
    }
};
user.display(); // Вызов метода
```

2. Объекты. Упрощенный синтаксис создания

С одной стороны, разницы никакой нет между двумя определениями. С другой стороны, бывают случаи, где заключение названия свойства в строку могут помочь. Например, если название **свойства** состоит **из двух слов**, разделенных пробелом:

```
var user = {
    name: "Tom",
    age: 26,
    "full name": "Tom Johns",
    "display info": function () {
        console.log(user.name);
        console.log(user.age);
    }
};
console.log(user["full name"]); // Обращение к свойству
user["display info"](); // Вызов метода
```

Только в этом случае для обращении к подобным свойствам и методам нужно использовать синтаксис массивов

2. Объекты. Удаление свойств и методов

Также можно **удалять** свойства и методы с помощью ключевого слова **delete**:

```
var user = {
    name: "Tom",
    age: 26,
    display: function () {
        console.log(user.name);
        console.log(user.age);
    }
};
console.log(user.name); // "Tom"
delete user.name; // Удаление свойства
console.log(user.name); // undefined
```

После удаления свойство будет не определено, поэтому при попытке обращения к нему, программа вернет значение **undefined**

2. Объекты. Вложенные

Одни **объекты могут содержать** в качестве свойств **другие объекты**. Например, есть объект страны, у которой можно выделить ряд свойств. Одно из этих свойств может представлять столицу. Но у столицы можно выделить свои свойства, например, название, численность населения, площадь:

```
var country = {
    name: "Украина",
    language: "украинский",
    capital: {
        name: "Киев",
        population: 2907817,
        area: 847.66
    }
};
console.log(country.capital.name); // "Киев"
console.log(country["capital"]["population"]); // 2907817
console.log(country.capital["area"]); // 847.66
```

2. Объекты. Вложенные

Для доступа к свойствам таких вложенных объектов можно использовать стандартную нотацию точки:

```
country.capital.name
```

Либо обращаться к ним как к элементам массивов:

```
country["capital"]["population"]
```

Также допустим смешанный вид обращения:

```
country.capital["year"]
```

2. Объекты. Массивы в объектах

В качестве свойств также могут использоваться **массивы**, в том числе массивы других объектов:

```
var country = {
    name: "Швейцария",
    languages: ["немецкий", "французский", "итальянский"],
    capital: {
        name: "Берн",
        population: 126598
    },
    cities: [
        {
            name: "Цюрих",
            population: 378884
        }, {
            name: "Женева",
            population: 188634
        }, {
            name: "Базель",
            population: 164937
        }
    ] };
```

2. Объекты. Массивы в объектах

```
console.log(country.name); // "Швейцария"
console.log(country.capital.name); // "Берн"
console.log(country.cities[0].name); // "Цюрих"

// Вывод всех названий городов
for (var i = 0; i < country.cities.length; i++) {
   console.log(country.cities[i].name);
}
```

В объекте country имеется свойство languages, содержащее массив строк, а также свойство cities, хранящее массив однотипных объектов

С этими массивами можно работать также, как и с любыми другими, например, перебрать с помощью цикла for. При переборе массива объектов каждый текущий элемент будет представлять отдельный объект, поэтому можно обратиться к его свойствам и методам:
country.cities[i].name

2. Объекты. Проверка наличия свойств и методов

При динамическом определении в объекте новых свойств и методов перед их использованием бывает важно **проверить**, а **есть ли уже** такие **свойства** и **методы**. Для этого может использоваться оператор **in**:

```
var user = {
    name: "Tom",
    age: 26,
    display: function () {
        console.log(user.name);
        console.log(user.age);
    }
};
var hasNameProp = "name" in user;
console.log(hasNameProp); // true - свойство name есть в user
var hasWeightProp = "weight" in user;
console.log(hasWeightProp); // false - в user нет свойства или
метода под названием weight
```

Перед ключевым словом in идет **в кавычках** название свойства или метода, а после in – название объекта. Если свойство или метод с подобным именем имеется, то оператор возвращает true. Если нет – то возвращается false

2. Объекты. Проверка наличия свойств и методов

Альтернативный способ заключается в проверке на значение **undefined**. Если свойство или метод равен undefined, то эти свойство или метод не определены:

```
var hasNameProp = user.name !== undefined;
console.log(hasNameProp); // true
var hasWeightProp = user.weight !== undefined;
console.log(hasWeightProp); // false
```

2. Объекты. Перебор свойств и методов

С помощью конструкции **for in** можно **перебрать объект как обычный массив** и получить все его свойства и методы, и их значения:

```
var user = {
    name: "Tom",
    age: 26,
    display: function () {
        console.log(user.name);
        console.log(user.age);
    }
};
for (var key in user) {
    console.log(key + ": " + user[key]);
}
```

В итоге консоль браузера отобразит следующий вывод:

```
name: Tom
age: 26
display: function () {
    console.log(user.name);
    console.log(user.age);
}
```

Источники

http://learn.javascript.ru/
http://metanit.com/
http://professorweb.ru/

Спасибо за внимание!