Прототипы

Объект

```
const student = {
    name: 'Billy'
};
```

Методы объекта

```
const student = {
    name: 'Billy',
    getName: function () {
        return this.name;
    },
    sleep: function () {}
};

student.getName();
// Billy
```

Методы объекта

```
const student = {
    name: 'Billy',
    getName: () => {
        return this.name;
    },
    sleep: () => {}
};

student.getName();
// ?
```

Методы объекта

```
const student = {
    name: 'Billy',
    getName() {
        return this.name;
    },
    sleep() {}
};

student.getName();
// Billy
```

```
const student = {
    name: 'Billy',
    getName() {
        return this.name;
    },
    sleep() {}
};
```

```
const lecturer = {
    name: 'Sergey',
    getName() {
        return this.name;
    },
    talk() {}
};
```

```
const student = {
    name: 'Billy',
    getName() {
        return this.name;
    },
    sleep() {}
};
```

```
const lecturer = {
    name: 'Sergey',
    getName() {
        return this.name;
    },
    talk() {}
};
```

```
const person = {
    getName() {
        return this.name;
    }
};
```

Задача – научить student пользоваться общим кодом, который вынесли в person

Заимствование метода

```
const student = {
    name: 'Billy',
};

const person = {
    getName() {
        return this.name;
    }
};

person.getName.call(student);

// student.getName();
```

Для создания такой связи между объектами есть специальное внутреннее поле [[Prototype]]

[[Prototype]]

```
const person = {
    getName() {
        return this.name;
    }
};

const student = {
    name: 'Billy',
    sleep() {},
    [[Prototype]]: <ссылка на person>
};

student.getName();
// Billy
```

Объект, на который указывает ссылка в [[Prototype]], называется прототипом

Если у объекта нет собственного метода – интерпретатор ищет его в прототипе

При вызове метода объекта в this записывается ссылка на этот объект, а не на прототип

[[Prototype]]

```
const person = {
    getName() {
        return this.name;
    }
};

const student = {
    name: 'Billy',
    sleep() {},
    [[Prototype]]: <ссылка на person>
};

student.getName();
// Billy
```

setPrototypeOf

```
const person = {
    getName() {
        return this.name;
    }
};

const student = {
    name: 'Billy',
    sleep() {}
};

Object.setPrototypeOf(student, person);

student.getName(); // Billy
```

Цепочка прототипов

```
const creature = {
    getName() { return this.name; }
};

const person = {
    [[Prototype]]: <creature>
};

const student = {
    name: 'Billy',
    [[Prototype]]: <person>
};

student.getName();
```

Когда поиск остановится?

```
const creature = {};



const person = {
    [[Prototype]]: <creature>
};

const student = {
    [[Prototype]]: <person>
};

student.getName();
```

Интепретатор будет идти по цепочке прототипов в поиске поля, пока не встретит null в поле [[Prototype]]

Object.prototype

```
const creature = {
    [[Prototype]]: <Object.prototype>
};

const person = {
    [[Prototype]]: <creature>
};

const student = {
    [[Prototype]]: <person>
};

student.getName();
```

Object.prototype – прототип для всех объектов по умолчанию. Содержит общие методы для всех объектов.

Object.prototype.hasOwnProperty()

```
Object.prototype = {
    hasOwnProperty() {}
};

const student = {
    name: 'Billy'
};

student.hasOwnProperty('name');
// true

student.hasOwnProperty('age');
// false
```

Когда поиск остановится?

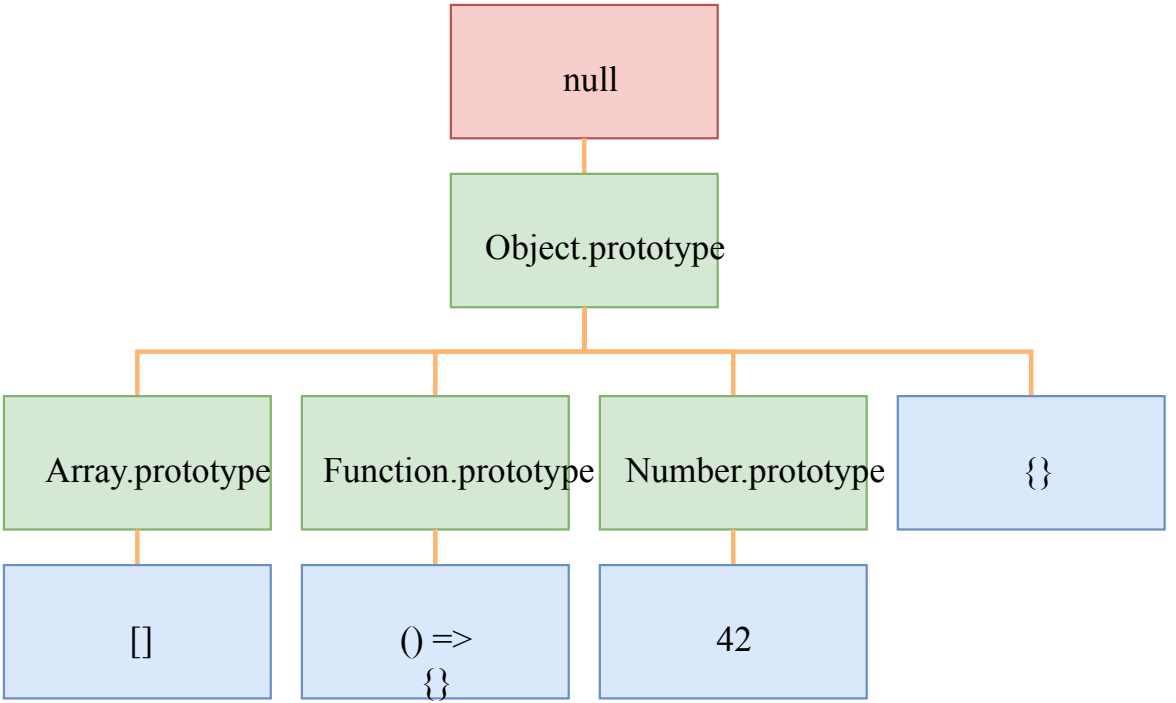
```
Object.prototype = { [[Prototype]]: null };

const creature = { [[Prototype]]: <Object.prototype> };

const person = { [[Prototype]]: <creature> };

const student = { [[Prototype]]: <person> };

student.getName(); TypeError: student.getName is not a function
```

Array.prototype

```
Array.prototype = {
    concat() {},
    slice() {},
    splice() {},
    forEach() {},
    filter() {},
    map() {},
    [[Prototype]]: <Object.prototype>
};
```

Function.prototype

```
Function.prototype = {
    call() {},
    apply() {},
    bind() {},
    [[Prototype]]: <Object.prototype>
};
```


null
Object.prototype
Array.prototype
Function.prototype
Number.prototype
{}
[]
() =>
{}
42

Цикл в цепочке прототипов

```
const student = {};
const person = {};

Object.setPrototypeOf(student, person);
Object.setPrototypeOf(person, student); Error

student.getName();
```

TypeError: Cyclic __proto__ value

setPrototypeOf

```
const student = {};
const person = {};


Object.setPrototypeOf(student, person);

Object.setPrototypeOf(student, null);

Object.setPrototypeOf(student, 42); Error
```

TypeError: Object prototype may only be an Object or null

getPrototypeOf

```
const student = {};
const person = {};


Object.setPrototypeOf(student, person);

Object.getPrototypeOf(student) === person; // true

Object.getPrototypeOf(Object.prototype) === null; // true
```

create

```
const person = {
    getName() {
        return this.name;
    }
};

const student = Object.create(person);

student.name = 'Billy';
```

create

```
const person = {
    getName() {
        return this.name;
    }
};

const student = Object.create(person, {
    name: { value: 'Billy' }
});
```

create быстрее, чем setPrototypeOf

super

```
const person = {
    getName() { return this.name; }
};

const student = {
    name: 'Billy',
    getName() { return 'Student ' + super.getName(); }
};

Object.setPrototypeOf(student, person)

student.getName(); // Student Billy
```

При создании метода, его внутреннее поле [[HomeObject]] заполняется ссылкой на объект, в котором он определён

super ссылается на прототип объекта
из поля [[HomeObject]]

В виде псевдокода

```
student.getName.[[HomeObject]] == student;

super == Object.getPrototypeOf(student.getName.[[HomeObject]]);
```

У обычных полей-функций
[[HomeObject]] не заполняется

super

```
const person = {
    getName() { return this.name; }
};

const student = {
    name: 'Billy',
    getName: function() {
        return 'Student ' + super.getName(); Error
    }
};
```

SyntaxError: 'super' outside of function or class

Значение [[HomeObject]] нельзя изменить

super

```
const person = {
    getName() { return 'and person ' + this.name; }
};

const student = {
    name: 'Billy',
    getName() { return 'Student ' + super.getName(); }
};

Object.setPrototypeOf(student, person)

const getName = student.getName; // this потеряли, но не super

getName(); // Student and person undefined
```

super навсегда привязывается к объекту,
в отличии от this

Свойства полей

Установка полей объекта

```
const student = {
    name: 'Billy'
};

student.age = 21;

student['age'] = 21;

Object.defineProperty(student, 'planet', {
    value: 'Earth'
});
```

Установка полей объекта со свойствами

```
const student = {};

Object.defineProperty(student, 'name', {
    writeable: false, // Можно не указывать, по умолчанию false
    value: 'Billy'
});

Object.defineProperty(student, 'age', {
    writeable: true,
    value: 21
});
```

getOwnPropertyDescriptor

```
const student = {};

Object.defineProperty(student, 'name', {
    value: 'Billy'
});

Object.getOwnPropertyDescriptor(student, 'name');
// {
//      value: 'Billy',
//      writable: false,
//      enumerable: false,
//      configurable: false
// }
```

Эффект затенения полей

```
const person = {
    planet: 'Earth'
};

const student = Object.create(person);

student.planet = 'Mars';

console.info(student.planet); // Mars

console.info(person.planet); // Earth
```

Работает не всегда!

Object.prototype.toString()

```
Object.prototype = {
    toString() {}
};

const student = {
    name: 'Billy'
};

console.info('Hello, ' + student); // Hello, [object Object]
```

Object.prototype.toString()

```
Object.prototype = {
    toString() {}
};

const student = {
    name: 'Billy'
};

student.toString = function {
    return this.name;
}

console.info('Hello, ' + student); // Hello, Billy
```

Неперезаписываемые поля

```
const student = {};

Object.defineProperty(student, 'name', {
    writable: false, // Можно не указывать, по умолчанию false
    value: 'Billy'
});

student.name = 'Willy';

console.info(student.name); // Billy
```

Неявное поведение!

Неперезаписываемые поля и use strict

```
'use strict';

const student = {};

Object.defineProperty(student, 'name', {
    value: 'Billy'
});

student.name = 'Willy';

console.info(student.name);
```

TypeError: Cannot assign to read only property 'name' of object

Неперезаписываемые поля в прототипах

```
const person = {};

Object.defineProperty(person, 'planet', {
    value: 'Earth'
});

const student = {};

Object.setPrototypeOf(student, person);

student.planet = 'Mars';
```

TypeError: Cannot assign to read only property

Неперезаписываемые поля, а не неизменяемые

```
const student = {};

Object.defineProperty(student, 'contacts', {
    value: {
        email: 'billy@example.com',
        telegram: '@billy'
    }
});

student.contacts.telegram = '@willy';

console.info(student.contacts.telegram); // @willy
```

Перечисляемые поля

```
const student = {
    name: 'Billy',
    age: 21
};

for (let key in student) {
    console.info(key);
} // name, age
```

Перечисляемые поля и прототипы

```
const person = { planet: 'Earth' };

const student = {
    name: 'Billy',
    age: 20
};

Object.setPrototypeOf(student, person);

for (let key in student) {
    console.info(key);
} // name, age, planet
```

hasOwnProperty

```
const person = { planet: 'Earth' };

const student = {
    name: 'Billy',
    age: 21
};

Object.setPrototypeOf(student, person);

for (let key in student) {
    if (student.hasOwnProperty(key)) {
        console.info(key);
    }
} // name, age
```

keys

```
const person = { planet: 'Earth' };

const student = {
    name: 'Billy',
    age: 21
};

Object.setPrototypeOf(student, person);

Object.keys(student); // ['name', 'age']
```

entries

```
const person = { planet: 'Earth' };

const student = {
    name: 'Billy',
    age: 21
};

Object.setPrototypeOf(student, person);

for (let [key, value] of Object.entries(student)) {
    console.info(key);
} // name, age
```

Неперечисляемые поля

```
const student = { name: 'Billy' };

Object.defineProperty(student, 'age', {
    enumerable: false,
    value: 21
});

Object.keys(student); // ['name']

JSON.stringify(student); // '{"name":"Billy"}'

Object.assign({}, student); // { name: 'Billy' }
```

Неперечисляемые поля в прототипах

```
const person = {};

Object.defineProperty(person, 'planet', {
    value: 'Earth'
});

const student = { name: 'Billy' };

Object.setPrototypeOf(student, person);

for (let key in student) {
    console.info(key);
} // name
```

Неперечисляемые поля по умолчанию

```
Object.prototype = {
    toString() {}
};

const student = { name: 'Billy' };

for (let key in student) {
    console.info(key);
} // name
```

getOwnPropertyNames

```
const student = { name: 'Billy' };

Object.defineProperty(student, 'age', {
    value: 21
});

Object.getOwnPropertyNames(student);
// ['name', 'age']
```

set/get

```
let name = null; // Не будет доступна снаружи модуля

const student = {
    get name() { return 'Student ' + name; }

    set name(value) {
        name = value;
    }
};

module.exports = student;
```

set/get

```
const student = require('./student');

student.name = 'Billy';

console.info(student.name); //Student Billy;
```

set/get в прототипах

```
let planet = null;

const person = {
    get planet() { return planet; },
    set planet(value) { planet = value; }
};

const student = {}

Object.setPrototypeOf(student, person);

student.planet = 'Mars';

student.hasOwnProperty('planet'); // false;
```

set/get

```
let name = null;

const student = {
    name: 'Willy',
    get name() { return 'Student ' + name; },
    set name(value) { name = value;}
};

student.name = 'Billy'

console.info(student.name); // Student Billy
```

set/get

```
let name = null;

const student = {
    get name() { return 'Student ' + name; },
    name: 'Willy', // get становится undefined
    set name(value) { name = value;}
};

student.name = 'Billy'

console.info(student.name); // undefined
```

set/get

```
let name = null;

const student = {
    name: 'Willy', // get становится undefined
    set name(value) { name = value;}
};

student.name = 'Billy'

console.info(student.name); // undefined
```

set/get

```
let name = null;

const student = {
    get name() { return 'Student ' + name; }
};

student.name = 'Billy'

console.info(student.name); // Student null
```

Поле одновременно может быть либо нормальным либо геттером/сеттером

Если есть хотя бы один из методов get или set, то поле становится геттером/сеттером

Геттеры/сеттеры

```
let name = null;

const student = {
    get name() { return 'Student ' + name; },
    set name(value) { name = value;}
};

Object.getOwnPropertyDescriptor(student, 'name');
// {
//      get: [Function: get],
//      set: [Function: set],
//      enumerable: true,
//      configurable: true
// }
```

Нормальные поля

```
const student = {
    name: 'Billy'
};

Object.getOwnPropertyDescriptor(student, 'name');
// {
//      value: 'Billy',
//      writable: true,
//      enumerable: true,
//      configurable: true
// }
```

Нормальные поля

```
Object.defineProperty(student, 'name', {
    value: 'Billy'
});

Object.getOwnPropertyDescriptor(student, 'name');
// {
//      value: 'Billy',
//      writable: false,
//      enumerable: false,
//      configurable: false
// }
```

Либо set/get, либо writable/value

Неконфигурируемые и неудаляемые поля

```
const student = {};

Object.defineProperty(student, 'name', {
    configurable: false, // По умолчанию false
    value: 'Billy'
});

delete student.name;

console.info(student.name); // Billy
```

configurable не контролирует изменение атрибута writable

Почитать

Speaking JavaScript
Chapter 17. Objects and Inheritance.
Layer 1: Single Objects

Speaking JavaScript
Chapter 17. Objects and Inheritance
Layer 2: The Prototype Relationship Between Objects

Почитать

Exploring ES6
14. New OOP features besides classes

MDN
Object.defineProperty()

Лекции 2015 года
Про this

Почитать

Современный учебник Javascript

ООП в прототипном стиле

Прототип объекта

Современный учебник Javascript

Современные возможности ES-2015

Объекты и прототипы