merge ОБРАЗОВАНИЕ И КАРЬЕРА

13-14 ноября ИННОПОЛИС

DataArt

Пространственно-временные топологии

Виктор Вершанский

Senior Developer, DataArt

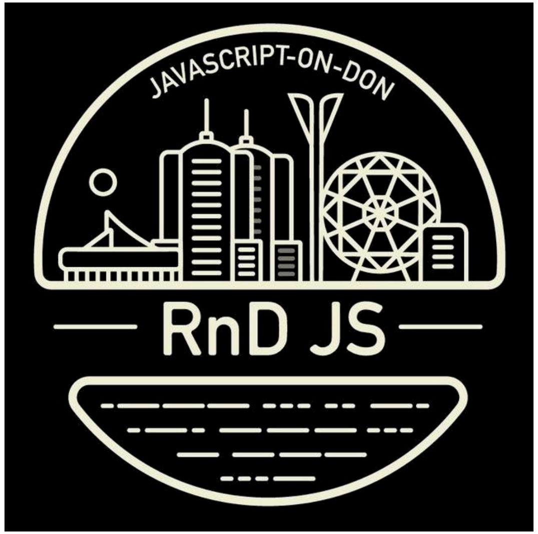
… examples ...

… With many Thanks ...

… With many Thanks ...

DataArt

… With many Thanks ...



DataArt

… With many Thanks ...

DataArt

BIO

- На JS начал писать после первого курса (1999)
- Преподавал экономику 8 лет (к.э.н.)
- Потом стал писать в «~~стол~~» Open Source
- Надоело писать, стал рассказывать
- Рассказывать получается так же и как писать
- Но уже не могу остановиться, так как привык ...
- Памагити, послушайте ещё раз !!111

Какую проблему решаем?

null is not a mistake

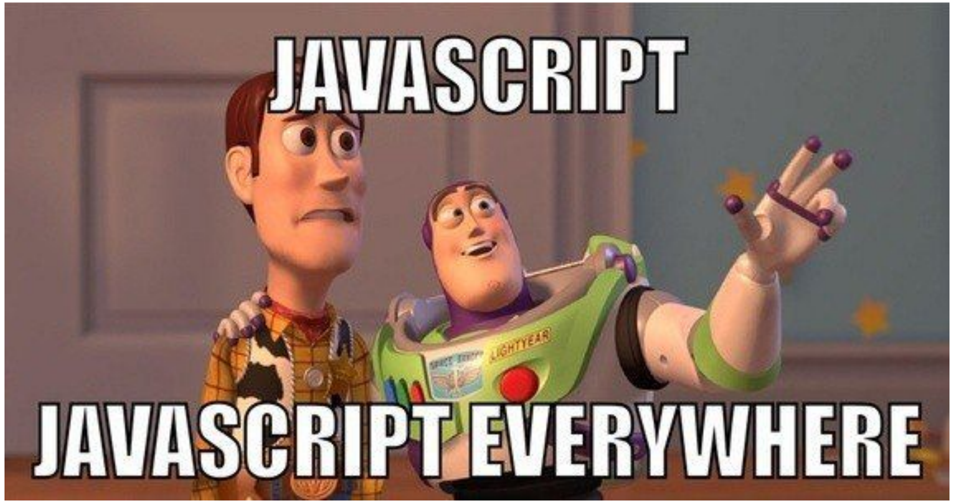
my apologies to Sir Charles Antony Richard Hoare

Какую проблему решаем?

type of null is also good

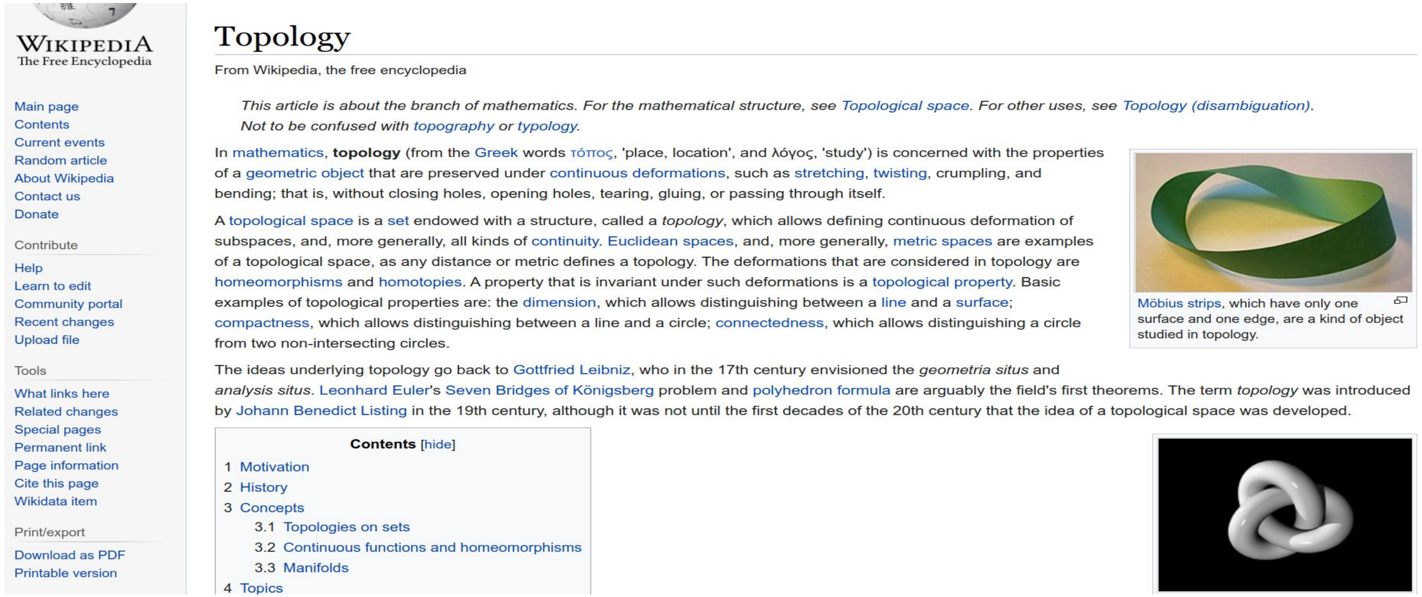
my apologies to Brendan Eich

Какую проблему решаем?



О чём пойдёт речь...

О чём пойдёт речь...



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute

Help
Learn to edit
Community portal
Recent changes
Upload file

Tools

What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export

Download as PDF
Printable version

Topology

From Wikipedia, the free encyclopedia

This article is about the branch of mathematics. For the mathematical structure, see Topological space. For other uses, see Topology (disambiguation).
Not to be confused with topography or typology.

In mathematics, **topology** (from the Greek words τόπος, 'place, location', and λόγος, 'study') is concerned with the properties of a geometric object that are preserved under continuous deformations, such as stretching, twisting, crumpling, and bending; that is, without closing holes, opening holes, tearing, gluing, or passing through itself.

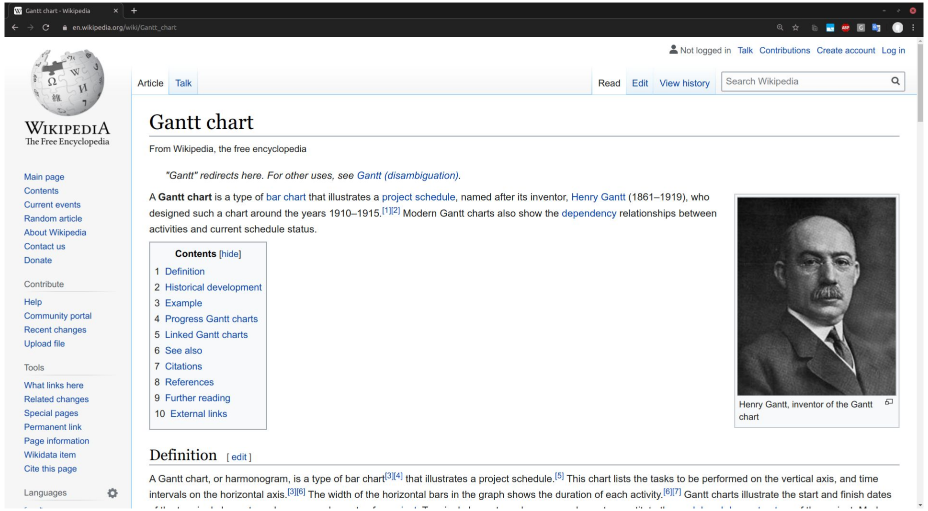
A topological space is a set endowed with a structure, called a *topology*, which allows defining continuous deformation of subspaces, and, more generally, all kinds of continuity. Euclidean spaces, and, more generally, metric spaces are examples of a topological space, as any distance or metric defines a topology. The deformations that are considered in topology are homeomorphisms and homotopies. A property that is invariant under such deformations is a topological property. Basic examples of topological properties are: the dimension, which allows distinguishing between a line and a surface; compactness, which allows distinguishing between a line and a circle; connectedness, which allows distinguishing a circle from two non-intersecting circles.

Möbius strips, which have only one surface and one edge, are a kind of object studied in topology.

The ideas underlying topology go back to Gottfried Leibniz, who in the 17th century envisioned the *geometria situs* and *analysis situs*. Leonhard Euler's Seven Bridges of Königsberg problem and polyhedron formula are arguably the field's first theorems. The term *topology* was introduced by Johann Benedict Listing in the 19th century, although it was not until the first decades of the 20th century that the idea of a topological space was developed.

Contents [hide]

При чём здесь Динамика Процессов ?



Gantt chart

From Wikipedia, the free encyclopedia

"Gantt" redirects here. For other uses, see Gantt (disambiguation).

A **Gantt chart** is a type of bar chart that illustrates a project schedule, named after its inventor, Henry Gantt (1861–1919), who designed such a chart around the years 1910–1915.[1][2] Modern Gantt charts also show the dependency relationships between activities and current schedule status.

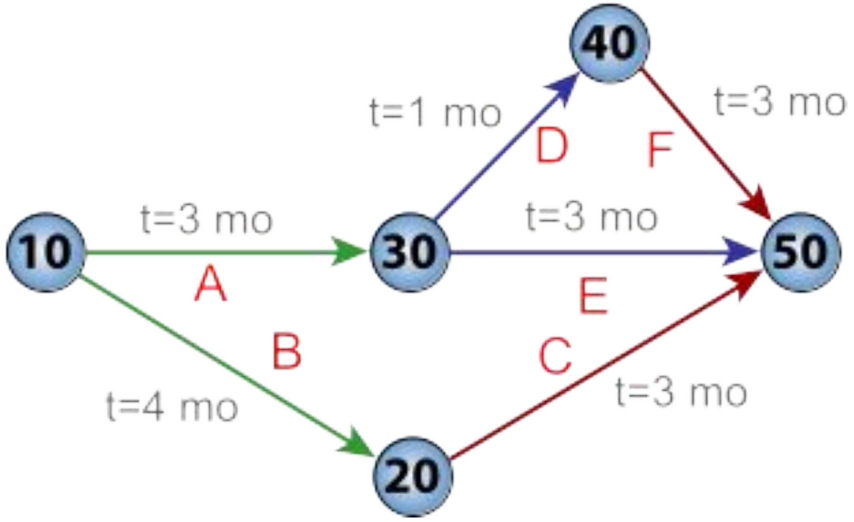
Contents [hide]

Henry Gantt, inventor of the Gantt chart

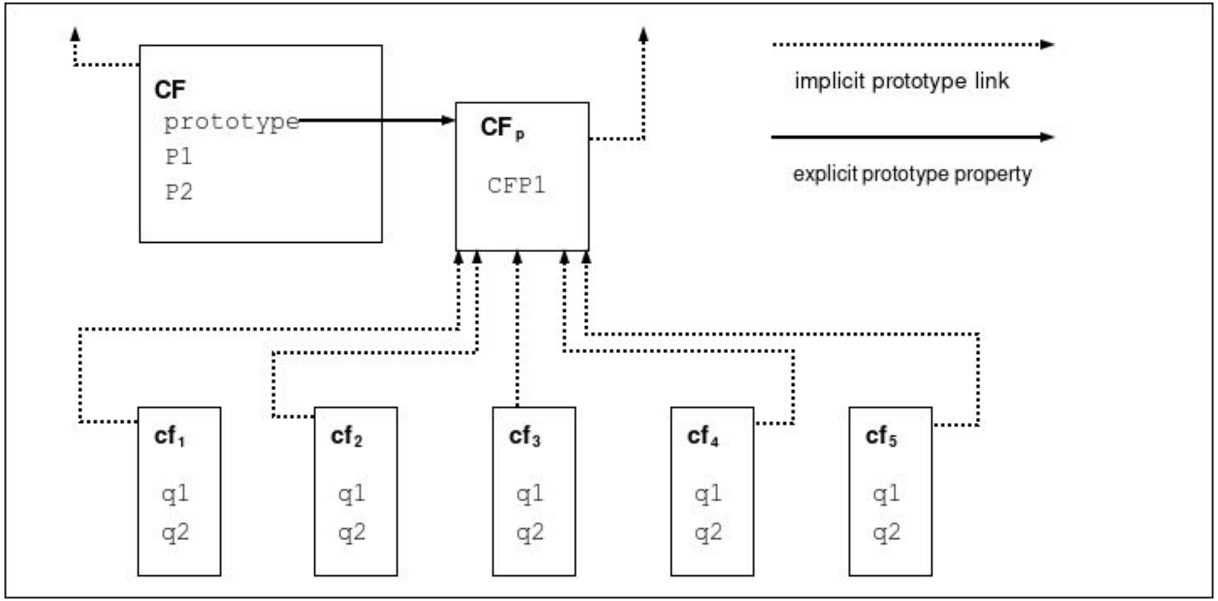
Definition [edit]

A Gantt chart, or harmonogram, is a type of bar chart[3][4] that illustrates a project schedule.[5] This chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis.[3][6] The width of the horizontal bars in the graph shows the duration of each activity.[6][7] Gantt charts illustrate the start and finish dates

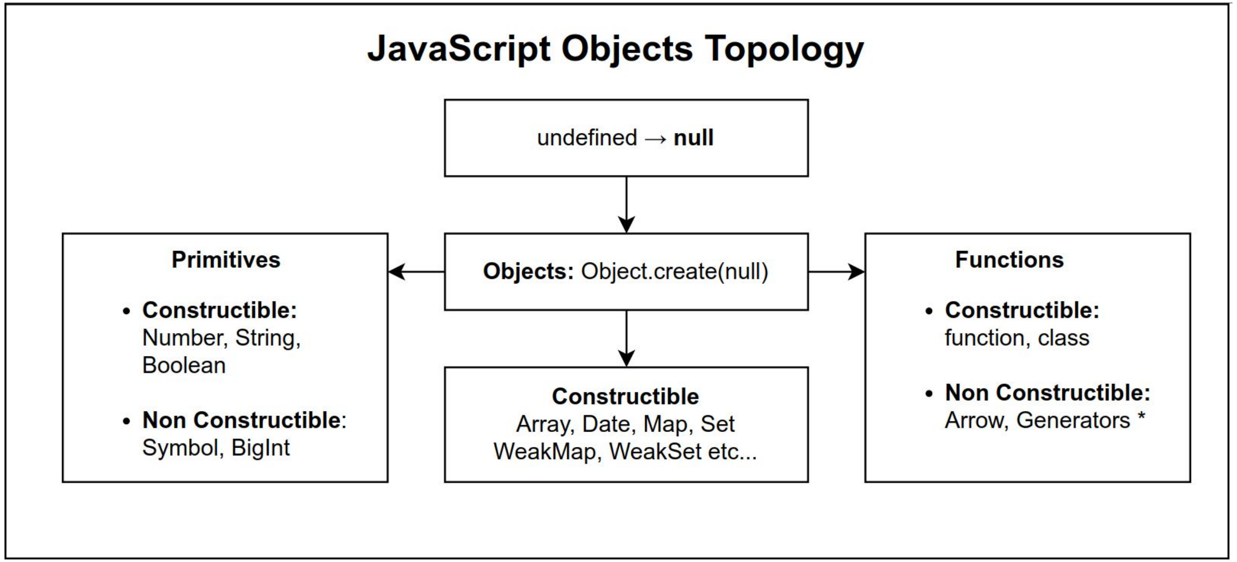
Можно ли показать Динамику в JavaScript ?



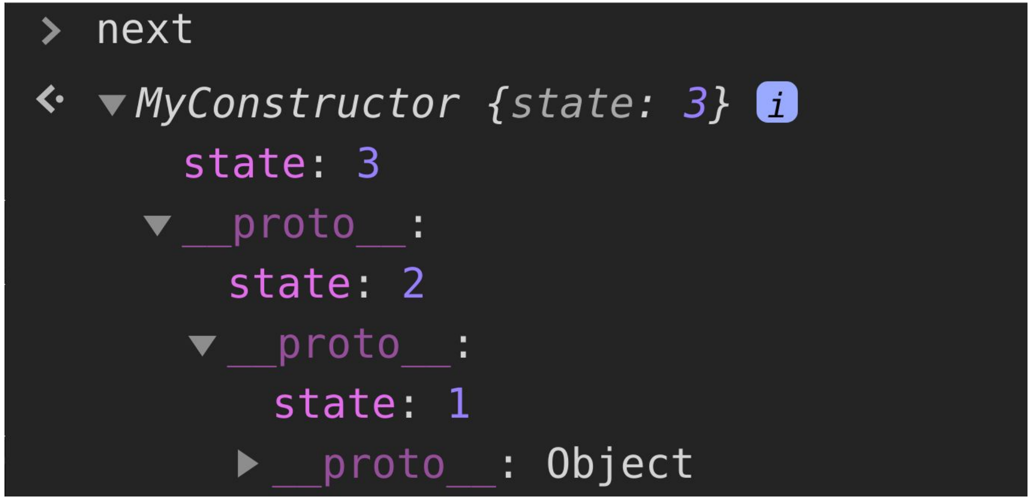
О чём, о чём :
о прототипах, конечно же



О чём, о чём : о прототипах, конечно же

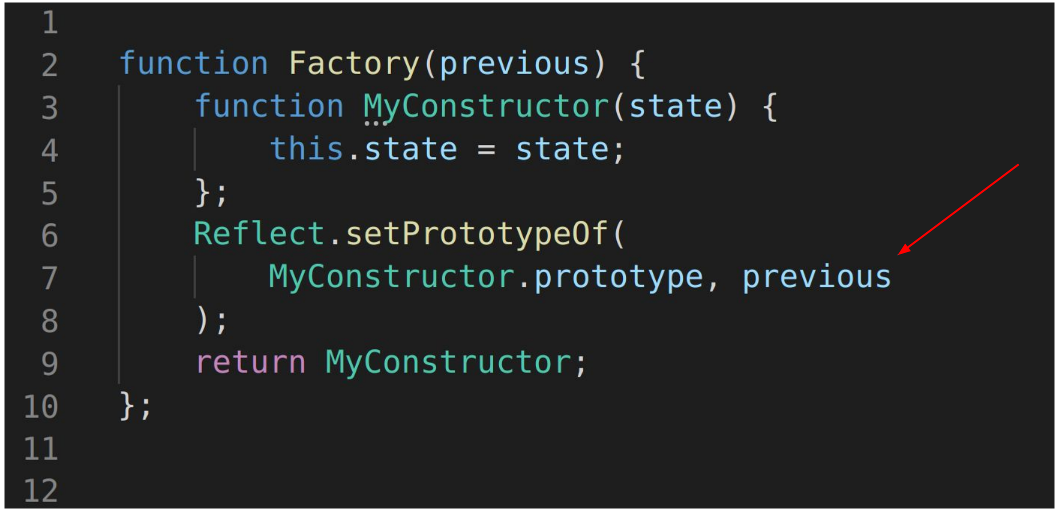

JavaScript Objects Topology
undefined → **null**
Objects: Object.create(null)
Primitives
Constructible: Number, String, Boolean
Non Constructible: Symbol, BigInt
Constructible Array, Date, Map, Set WeakMap, WeakSet etc...
Functions
Constructible: function, class
Non Constructible: Arrow, Generators *

Как можно выразить Динамику прототипами ?



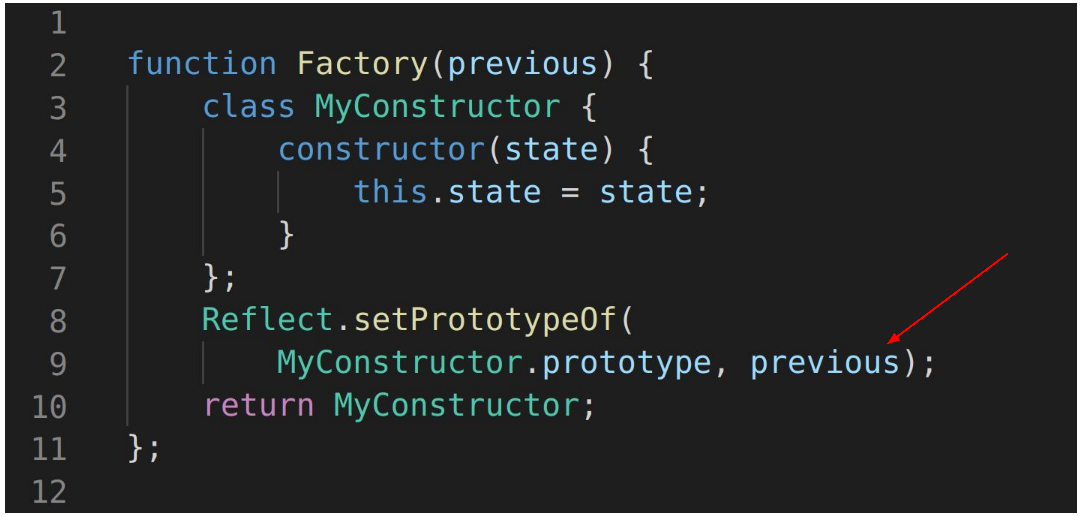
```
> next
<· ▼MyConstructor {state: 3} i
      state: 3
    ▼__proto__:
        state: 2
      ▼__proto__:
          state: 1
        ▶__proto__: Object
```

Инструментарий : Фабрика Конструкторов



```

function Factory(previous) {
    function MyConstructor(state) {
        this.state = state;
    };
    Reflect.setPrototypeOf(
        MyConstructor.prototype, previous
    );
    return MyConstructor;
};


```

Инструментарий:
Фабрика Конструкторов



```javascript

function Factory(previous) {
    class MyConstructor {
        constructor(state) {
            this.state = state;
        }
    };
    Reflect.setPrototypeOf(
        MyConstructor.prototype, previous);
    return MyConstructor;
};

```

Что это даёт?

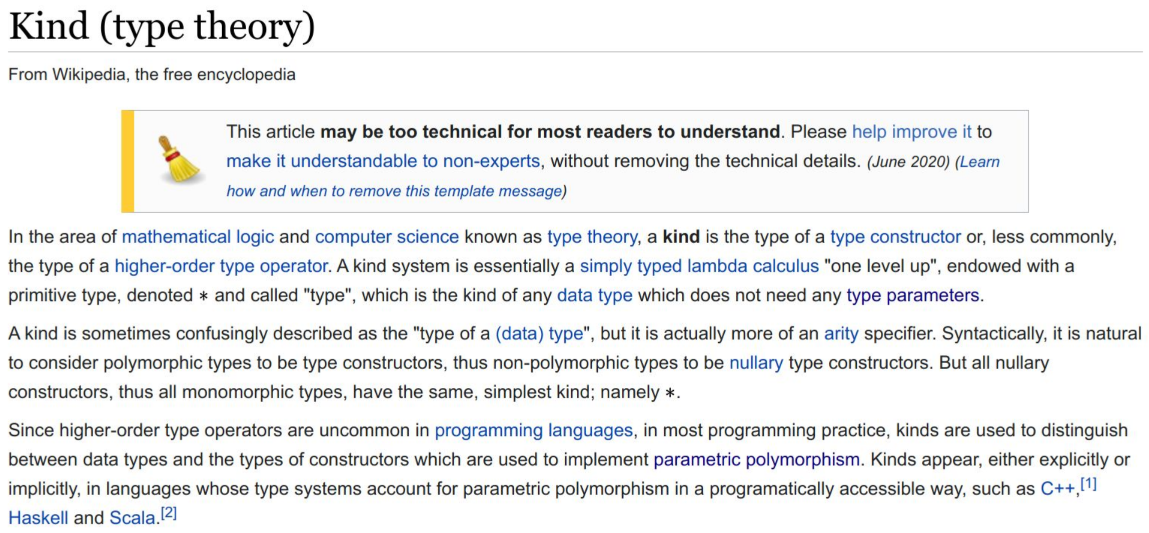
- Теперь Конструкторы наследуются от Экземпляров
- Runtime стал более динамичным, но сохранил суть
- Динамика происходившего лежит в Prototype Chain
- Можно добавить различные плюшки: TimeStamps
- Это наследование ... но «оно Другое»

Что это Не даёт?

- Теперь нужно будет работать с Жизненным Циклом
- С типами «всё сложно», тоже нужно будет думать
- Prototype Chain – сложная концепция, мучительно
- Можно забыть добавить различные плюшки
- Это наследование ... и «оно Другое»

DataArt

Родословные Зависимых Типов



Kind (type theory)

From Wikipedia, the free encyclopedia

This article **may be too technical for most readers to understand**. Please help improve it to make it understandable to non-experts, without removing the technical details. *(June 2020) (Learn how and when to remove this template message)*

In the area of mathematical logic and computer science known as type theory, a **kind** is the type of a type constructor or, less commonly, the type of a higher-order type operator. A kind system is essentially a simply typed lambda calculus "one level up", endowed with a primitive type, denoted $*$ and called "type", which is the kind of any data type which does not need any type parameters.

A kind is sometimes confusingly described as the "type of a (data) type", but it is actually more of an arity specifier. Syntactically, it is natural to consider polymorphic types to be type constructors, thus non-polymorphic types to be nullary type constructors. But all nullary constructors, thus all monomorphic types, have the same, simplest kind; namely $*$.

Since higher-order type operators are uncommon in programming languages, in most programming practice, kinds are used to distinguish between data types and the types of constructors which are used to implement parametric polymorphism. Kinds appear, either explicitly or implicitly, in languages whose type systems account for parametric polymorphism in a programatically accessible way, such as C++,[1] Haskell and Scala.[2]

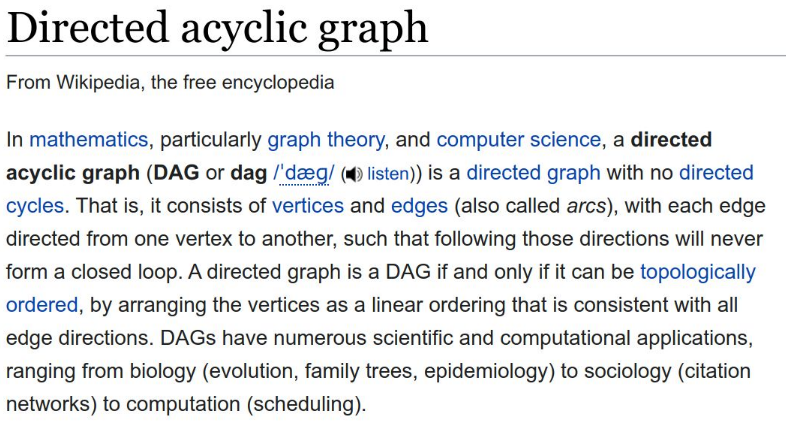
Kind (Type Theory)

- **Kind** – это **тип** Type Constructor’a
- **Type Constructor** – свойство типизированного формального языка, которое строит **новые** типы **из существующих**
- Но ведь **Фабрика Конструкторов** и строит новые Конструкторы из **Экземпляров**!

• Получается, что мы Создали

- **Конструкторы**, принимающие **полиморфный** (любой) тип в качестве параметра для своего построения
- **Типы** в Жизненном Цикле, зависимые лишь от необходимого состояния среды исполнения в настоящий момент
- Зависимости между свойствами Экземпляров типов доступные и управляемые через цепочку прототипов

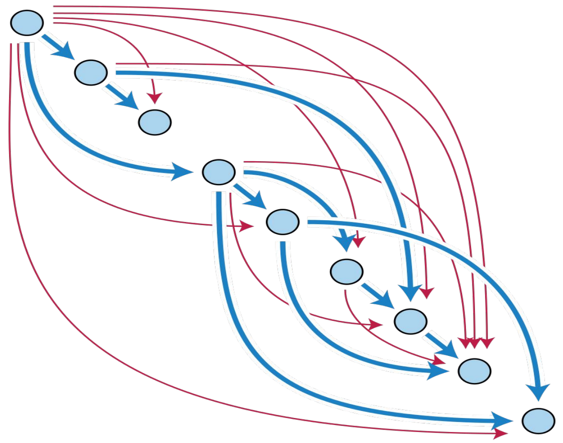
Как выглядят эти Структуры Данных



Directed acyclic graph

From Wikipedia, the free encyclopedia

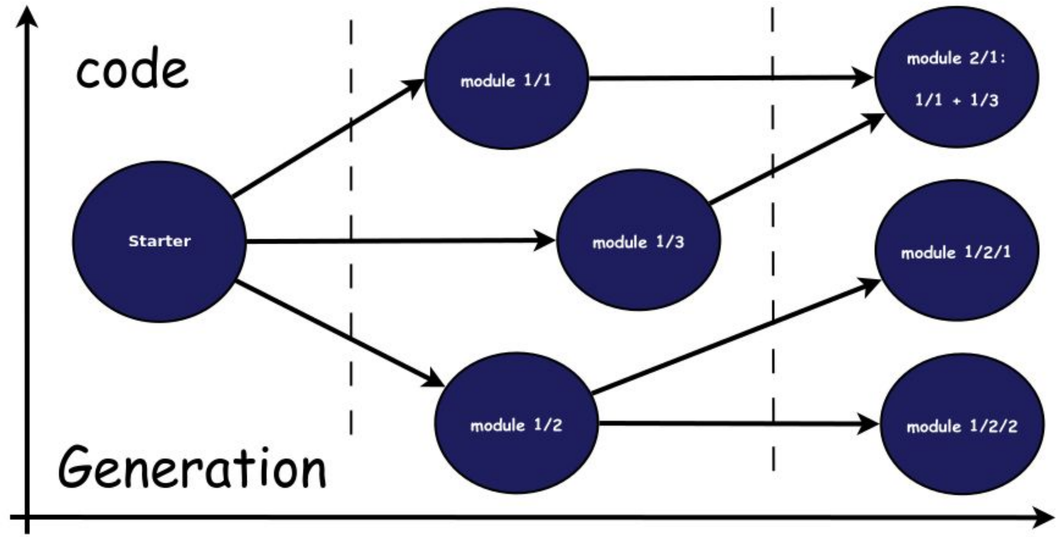
In mathematics, particularly graph theory, and computer science, a **directed acyclic graph** (**DAG** or **dag** /ˈdæɡ/ (listen)) is a directed graph with no directed cycles. That is, it consists of vertices and edges (also called *arcs*), with each edge directed from one vertex to another, such that following those directions will never form a closed loop. A directed graph is a DAG if and only if it can be topologically ordered, by arranging the vertices as a linear ordering that is consistent with all edge directions. DAGs have numerous scientific and computational applications, ranging from biology (evolution, family trees, epidemiology) to sociology (citation networks) to computation (scheduling).



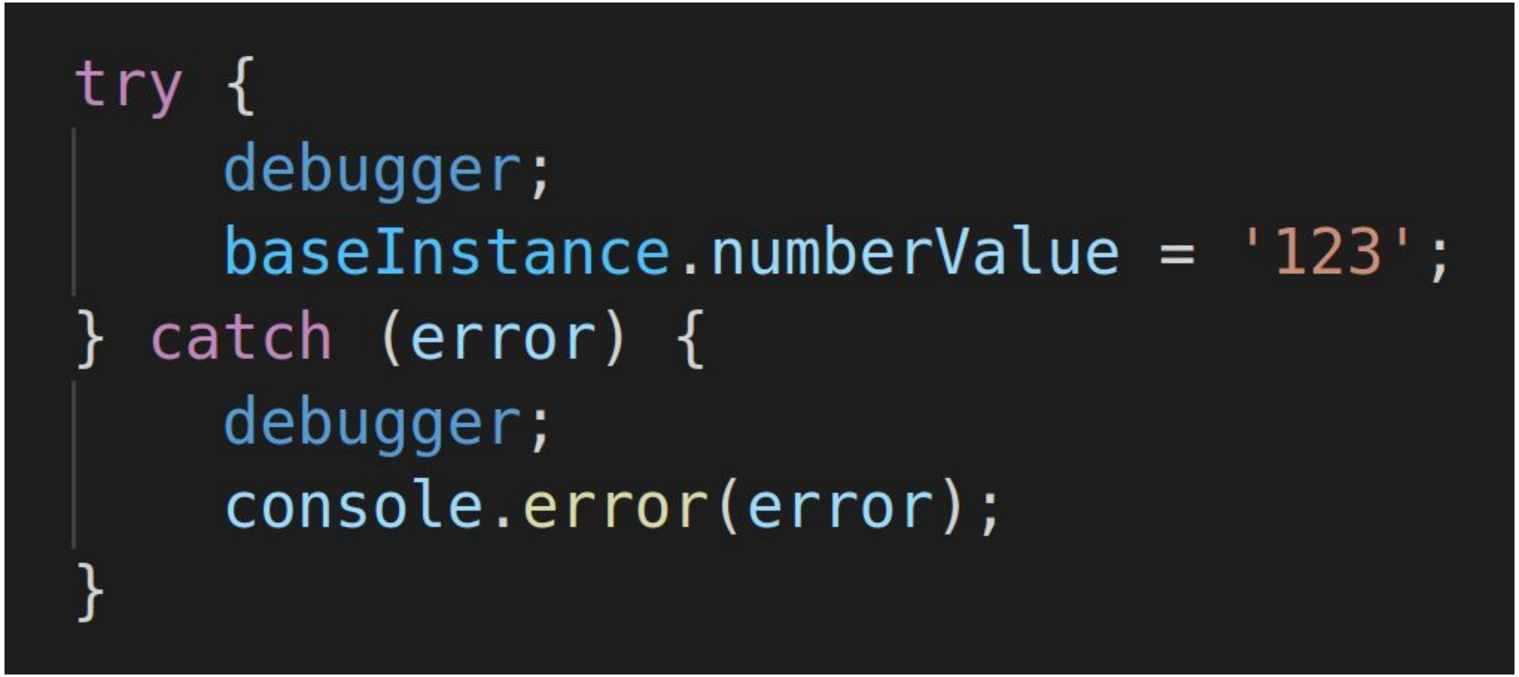
• Как уведомлять наследников об Изменениях

- Не использовать одинаковые Имена Свойств: **Liskov Substitution Principle**
- Определять методы Единожды в корне цепочки: **Single Responsibility Principle**
- При необходимости подписаться на изменения: **Interface Segregation** – для реализации действий специфичных именно для данной конкретной сущности

Инструментарий:
Древовидные структуры

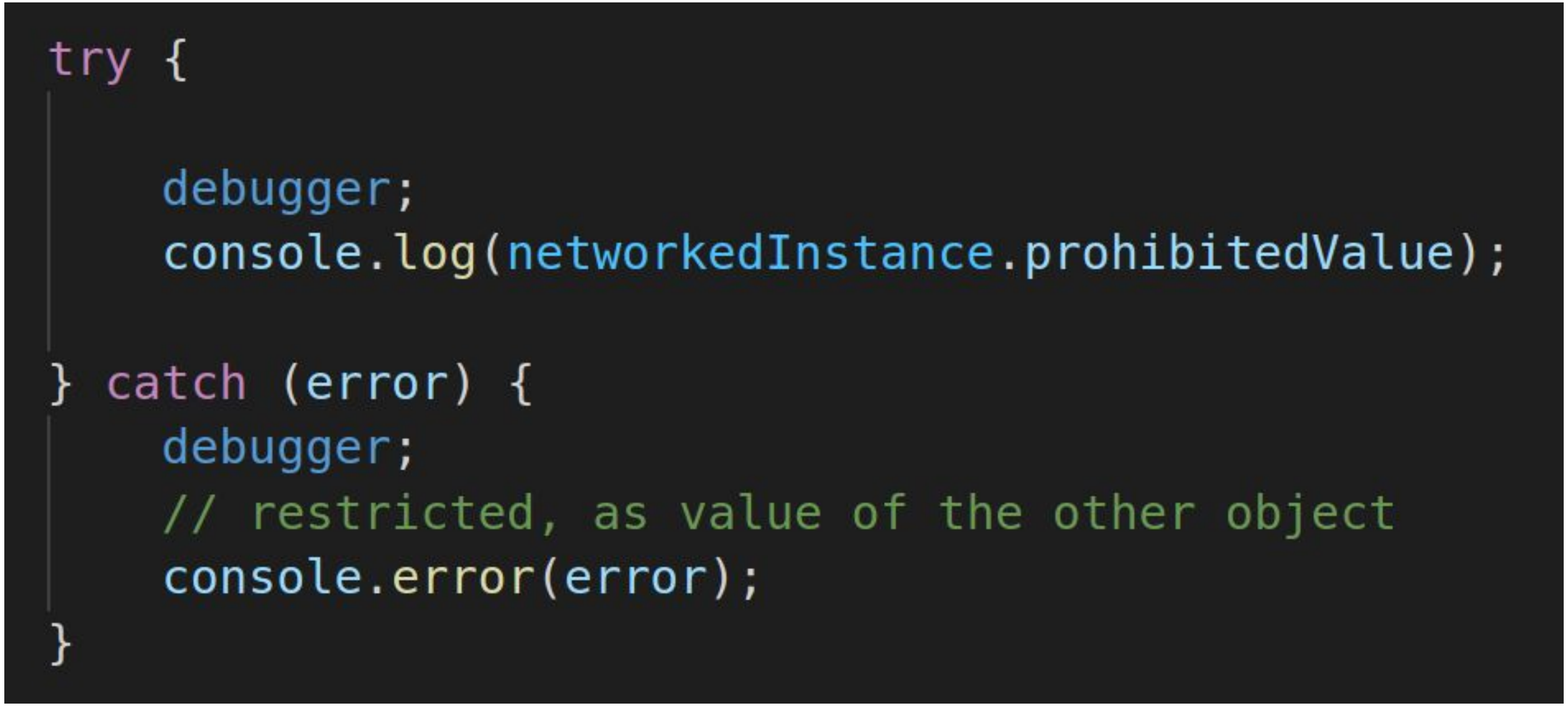


Инструментарий:
Строгая Типизация



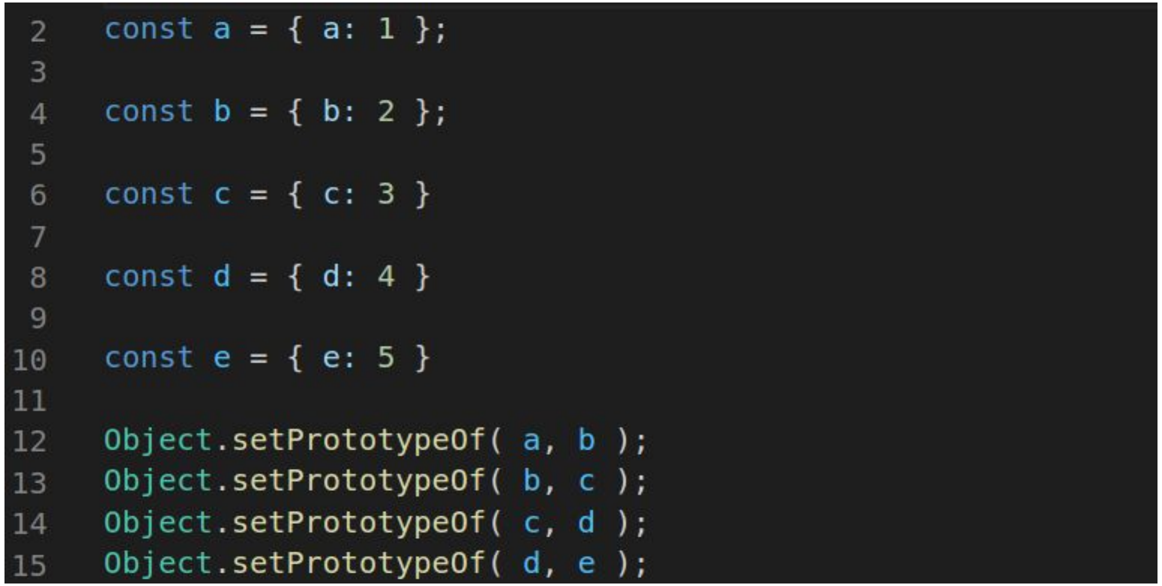
```
try {
    debugger;
    baseInstance.numberValue = '123';
} catch (error) {
    debugger;
    console.error(error);
}
```

Инструментарий: валидация структуры наследования



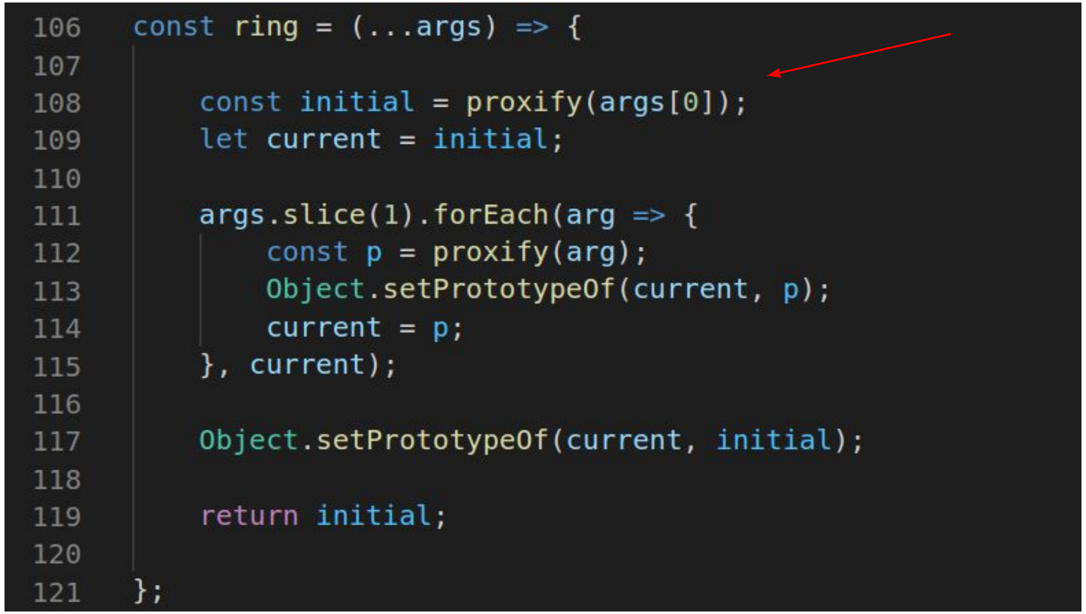
```
try {

    debugger;
    console.log(networkedInstance.prohibitedValue);

} catch (error) {
    debugger;
    // restricted, as value of the other object
    console.error(error);
}
```

Инструментарий:
Множественное Наследование



```
const a = { a: 1 };

const b = { b: 2 };

const c = { c: 3 }

const d = { d: 4 }

const e = { e: 5 }

Object.setPrototypeOf( a, b );
Object.setPrototypeOf( b, c );
Object.setPrototypeOf( c, d );
Object.setPrototypeOf( d, e );
```

Инструментарий:
Кольцевая структура Наследования



```
const ring = (...args) => {

    const initial = proxify(args[0]);
    let current = initial;

    args.slice(1).forEach(arg => {
        const p = proxify(arg);
        Object.setPrototypeOf(current, p);
        current = p;
    }, current);

    Object.setPrototypeOf(current, initial);

    return initial;

};
```

DataArt

Спасибо!

… examples ...

FIN