prototype.js

class.js

Обзор на ‘Прототипно-базираното ООП’

{

[и сравнение с традиционното ООП]

< Наследяване. Примери в JS >

< Какво е указателят this в JS? >

}

prototype.js

class.js

```
01 {

  [“Прототипно” ООП]

     < Николай Алексиев >

}
```

Защо съществува прототипно-базираното ООП?

Прототипно базирано ООП е стил в програмирането, който се използва за решаването на проблема как да бъде описан един обект в програма. Това е цел за която е съществувало решение и преди прототипното програмиране, което води до въпроса защо ни е нужно. Има две главни причини заради които е създадено и с които то се отличава от предишните решения.

- По интуитивен начин на работа.
- Отваря възможности при езици които се интерпретират.

prototype.js

class.js

```
02 {
    [vs “Традиционно” ООП]

        < Росица Деянова >
}
```

Защо да заложим на прототип?

Прототипно или клас базирано ООП? И двата подхода са изследвани и дебатирани в академичните среди. Няма съгласие дали едното е по-добро от другото, но има няколко причини, поради които бихте предпочели прототипното ООП.

* По-просто за разбиране
* По-гъвкаво
* По-динамично

Можем да демонстрираме динамичността му, чрез решаване на проблема с функцитята **indexOf**, която не е дефинирана в Internet Explorer 8.

prototype.js

class.js

Елементи на “традиционното” ООП

* Клас
* Полета
* Конструктор
* Методи

Елементи на “прототипното” ООП

* Променлива, приемаща стойностите на полетата
* Методи, добавени чрез прототипа

prototype.js

class.js

03 {

[Наследяване. Примери в JS]

< Мария Янева >

}

Прототипно-базирано наследяване

Няма класове, а процес на клониране на съществуващи обекти, които служат като прототипи

Верига от прототипи (Prototype Chain)

inheritance.js

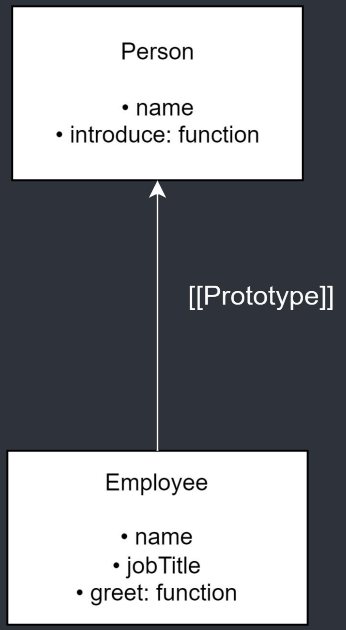
prototype.html

Демо

{

- Създаване на прототип
- Добавяне на свойства и методи към прототипа
- Наследяване при стандартите EcmaScript 5 и EcmaScript 6

}



prototype.js

class.js

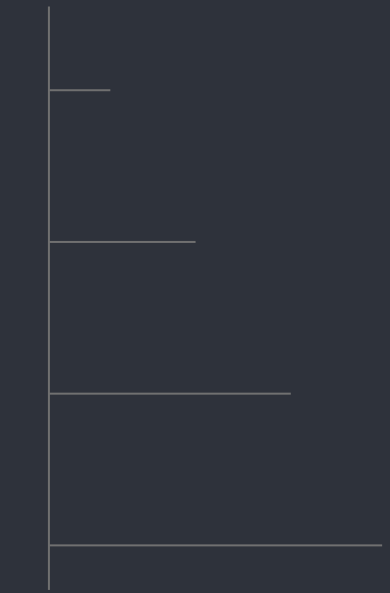
04 {

[Указателят this в JS]

< Яница Хаджиева >

}

Какво е указателят ‘this’ в JS? {



}

Променлива, която се отнася до текущият обект или настоящият контекст.

Стойността на “this” , се определя от това как е извикана функцията,когато е използван във функция.

Стойността може да е различна всеки път, когато функцията бъде извикана.

Режими на използване - стриктен и не стриктен (strict mode, non–strict mode).

СИНТАКСИС И СТОЙНОСТ

Синтаксис

- this

Стойност и свойства

- контекст на изпълнението - глобален, функционален и оценяващ
- (global, function or eval),
- в не стриктен режим винаги е препратка към обект,
- в стриктен режим може да бъде всяка стойност.

Как използваме ‘this’?

Глобален контекст (GLOBAL CONTEXT)

- “this” се отнася(реферира) към глобален обект , без значение в стриктен или в не стриктен режим е.

Функционален контекст (FUNCTION CONTEXT)

- Вътре във функция ,
- “this” и комуникация с обекти

  - За да зададем стойност, извикваме функцията ,като използваме call(), или apply().

Клас контекст(Class context)

- В рамките на конструктор на клас,“this” е обикновен обект.
- Всички нестатични методи в класа се добавят към прототипа на “this”.

➔ Забележка: Статичните методи не са свойства на “this”. Те са свойства на самия клас.

Метод за свързване (The bind method)

- Нова функция със същото тяло и обхват като старата, но когато “this ” се появи в оригиналната функция, новата функция е обвързана постоянно само с първия аргумент на bind.

DOM EVENT HANDLER; {

“this”

“This” е зададен на елемента, върху който е поставен listener (слушателят).

}

prototype.js

class.js

БЛАГОДАРИМ ВИ ЗА ВНИМАНИЕТО!