Інтернет програмування

лек 6

Об'єкти, прототипи, наслідування

Зав.каф.КН, PhD Граф М.С.

Об’єкти

```
var obj = {};
```

```
var student = {
  “name” : “Dima”,
“age” : 20
};
```

```
student.name; // “Dima”
student.age;    // 20
```

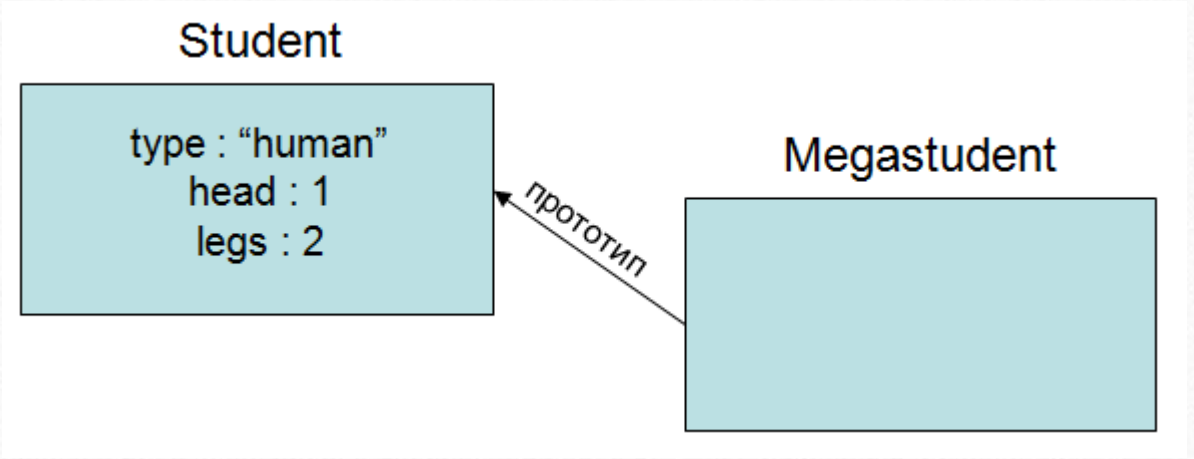
```
var student = {
   name : “Dima”,
age : 20,
  “” : “Something!”
};
student[“name”]; // “Dima”
student[“age”];    // 20
student[“”];         // “Something!”
```

```
var student = {
   name : “Dima”,
age : 20
};
student.height;    // undefined

student.name = “Peter”;
student.height = 176;

student.name;      // “Peter”
student.height;    // 176
```

Прототипи




```
Student.face;      // undefined
Megastudent.face;  // undefined
```

```
Student.face = “okay”;
Student.face;      // “okay”
Megastudent.face;  // “okay”
```

```
Megastudent.face = “awesome”;
Megastudent.face; // “awesome”
Student.face;      // “okay”
```

Як вказати прототип

```
var Student = {
   type : “human”,
head : 1,
   legs : 2
};
var Megastudent = Object.create(Student);
```

```
var Student  = {
   type : “human”,
head : 1,
   legs : 2
};
var Megastudent = Object.create(Student);
Megastudent.head = 2;
Megastudent.head;     // 2
delete Megastudent.head;
Megastudent.head;     // 1
```

This – пояснення

```
function getIt() { return this.x; }
```

Створення об’єкту:

```
var obj1 = { get : getIt, x : 1 }
```

Створимо другий такий об'єкт

```
var obj2 = { get : getIt, x : 2 }
```

При викликах виходить наступне:

```
obj1.x  // 1
obj2.x  // 2
obj1.get()  // 1
obj2.get()  // 2
```

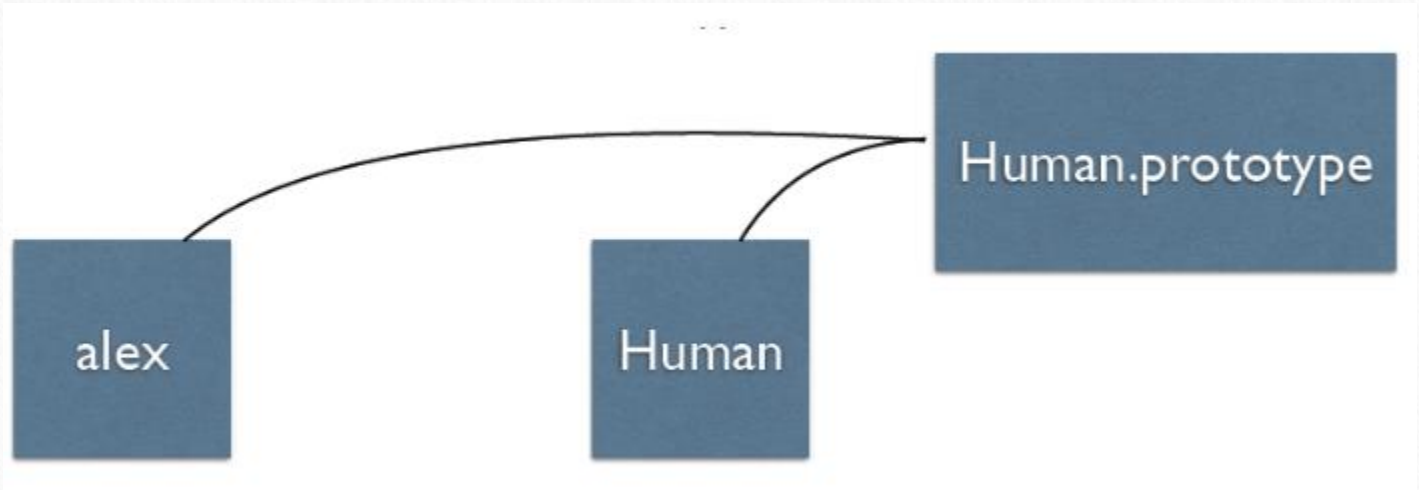
Наприклад, можемо задати функцію безпосередньо в об'єкті, при оголошенні якості.

```
var obj = {
   base : 8,
   average : function (x, y) {
      return (this.base+x+y)/3;
   };
};
```

Наслідування

```
var alex = new Human();
```

Human() – це функція. Якщо ми її просто викличемо, то можливо вона навіть зробить щось корисне. Але якщо ми її викличемо зі словом **new** , то цей виклик перетвориться на виклик конструктора, і виклик конструктора буде дещо відрізнятися від виклику цієї функції самої по собі. Коли ми викликаємо функцію **Human з префіксом new**, то відбувається таке – всередині функції створюється новий об'єкт, й у ньому немає. Далі ця функція дивиться на прототип.



Наслідування

Цей рядок, повернення необ'єкта, повернення чогось, не працюватиме взагалі при цьому виклику.

```
function Human() {
   return 1;
}
```

Якщо ми цю функцію викличемо як завжди, без new, а просто Human, те, що повертається, буде повернуто. Це буде звичайним викликом звичайної функції. Якщо ж ця функція повертає об'єкт, як у цьому випадку:

```
function Human() {
   return { a:1 };
}
```

Наслідування

Якщо ми хочемо просто створити об'єкт і не надавати йому ніякої властивості name, це буде порожня функція, то можна просто закрити дужку, і її можна використовувати як конструктор.

```
function Human(name) {
   this.name = name;
}
Human.prototype.say = function(what) {
   alert(this.name + " : " + what);
}
var alex = new Human("Alex");
alex // { name : "Alex"}
alex.say("hello!");     // Alex : hello!
Human("Galex");
name                    // "Galex"
```

Щоб показати зв'язок із прототипом, я створю у прототипі функції Human (у прототипі об'єкта Human) нову функцію. Я зроблю це так.

```
Human.prototype.say = function(what) {
   alert(this.name + " : " + what);
}
```

Наслідування

У нас є конструктор і ми маємо прототип цієї функції.
Тепер зробимо так:

```
var alex = new Human("Alex");
```

Але ми знаємо, що цей об'єкт успадкований від того ж прототипу, який є прототипом Human. а у прототипу Human є властивість say. тому я можу зробити так:

```
alex.say("hello!");  // Alex : hello!
```

Попробуем вызвать функцию без new **Human(“Galex”);**

Якщо ми викликаємо цю функцію, то це не буде вказувати на новостворений об'єкт, тому що немає новоствореного об'єкта. Ми викликаємо функцію без нового, і новий об'єкт не створюється. Ця функція легко виконується, рядок за рядком, і єдиним рядком тут заявляється this.name = name. Я запустив цю функцію, і ось що сталося: зголосився рядок this.name = "Galex";. Т.к. ми в момент виклику знаходимося не в якомусь об'єкті, а в глобальному об'єкті, у зовнішньому об'єкті Javascript. У цьому об'єкті з'явилося нове властивість name. І ми можемо його подивитися, просто викликавши його так: name. Грубо кажучи, вийшла світова змінна. І це небажано допускати у вашому коді. Великий мінус глобального об'єкта – це може призвести до помилок насправді, т.к. у браузері, крім вашого коду, може бути запущено ще багато іншого коду.

Або можна зробити невеликий трюк. Можна переписати функцію. При створенні функції перед вказівкою this необхідно перевірити, де я зараз перебуваю. У момент виклику функції, де відбувається виклик.

```
function Human(name) {
   if (! (this instanceof Human)){return new Human(name); }
   this.name = name;
}
```

Instanceof

Оператор instanceof використовується для перевірки, чи належить об'єкт даному типу.

Синтаксис

var isInstance = екземпляр об’єкта ObjectType

Аргументи

object – Об'єкт

ObjectType – Конструктор(тип) для сравнения

instanceof перевірить, чи є в ланцюжку прототипів лівого аргументу, прототип правого аргументу.

Наслідування

- Об'єкт містить якісь властивості.
- Об'єкт містить спеціальну властивість, що вказує на макет об'єкта.
- Об'єкт може перевизначати будь-яку властивість прототипу. (Пам'ятайте, якщо цієї властивості в об'єкті немає, то ми дивимося прототип, і ми йдемо вгору, поки не натрапимо на таку властивість. Ми можемо створити цю властивість прямо в об'єкті, і тоді ми не буде звертатися до прототипу, ми перевизначимо для даного об'єкта це властивість).
- Конструктор створює об'єкт, прототип цього об'єкта буде прототип конструктора.

Наслідування

Є ще один спосіб створити об'єкт за допомогою функції create(). Ми наслідуємо безпосередньо від якогось об'єкта, передаємо об'єкт і отримуємо дитину від цього об'єкта.

```
var parent = { name : “Peter” };   // parent {name : “Peter”}
var child = Object.create(parent); // child {}
child.name                         // “Peter”
```

В результаті виклику цієї функції прямим прототипом об'єкта child є об'єкт parent.

немає класів, є лише об'єкти, і об'єкти мають прототипи.

Cookies

Cookie – шматок даних, який зберігається браузером на комп'ютері користувача. На відміну від змінних вони не зникають після закриття браузера або перезавантаження сторінки. Куки зберігають дані за клієнта, не торкаючись сервер. При цьому користувач завжди може видалити куки зі свого комп'ютера, якщо він вирішив, що ця інформація більше не потрібна.

Вплив на безпеку:

Cookie – це фрагменти текстової інформації, що зберігається на комп'ютері клієнта.
Хоча куки зберігаються на жорсткому диску, вони не мають доступу до іншої інформації, що зберігається там.
Не є програмами, що виконуються, куки не можуть стати джерелами вірусів або черв'яків.
Куки можуть зберігати персональні дані, але тільки після того, як користувач свідомо вкаже їх на веб-сторінці.
У деяких браузерах кукі можуть бути заборонені. Для вирішення цієї проблеми можна перевірити доступність cookie за допомогою методу **navigator.cookieEnabled** . Він повертає true у разі доступності cookie та false – якщо до них немає доступу.

Дякую за увагу!