Кросс-платформенная разработка

Лекция 5

Темы

- Объекты

- Объекты

- Объекты

- Объекты

Objects

```
const user = {     // an object
    name: "John",  // by key "name" store value "John"
    age: 30        // by key "age" store value 30
};
```

Properties

```
// get property values of the object:
alert( user.name ); // John
alert( user.age ); // 30


user.isAdmin = true;

delete user.age;
```

How **const** works

```
const user = {
    name: "John"
};

user.name = "Pete";

alert(user.name);
```

We can access properties hash-like

```
const user = {
  name: "John",
  "likes birds": true
};

alert(user['likes birds']);

let key = prompt("What key?");
alert(user[key]);
```

Short-hand

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...other properties
  };
}
```

```
function makeUser(name, age) {
  return {
    name, // same as name: name
    age,  // same as age: age
    // ...
  };
}
```

Checking for existance

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist
```

For-in loop

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key );  // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
}
```

By reference of by value?

```
let message = "Hello!";
let phrase = message;


let user = { name: "John" };
let admin = user;
```

Check it

```
let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // changed by the "admin" reference

alert(user.name);
```

But I need to make copies!

```
let user = {
  name: "John",
  age: 30
};

let clone = {};

for (let key in user) {
  clone[key] = user[key];
}

clone.name = "Pete"; // changed the data in it

alert( user.name ); // still John in the original object
```

Object.assign

```
let user = { name: "John" };

let permissions1 = { canView: true };
let permissions2 = { canEdit: true };

// copies all properties from permissions1 and permissions2 into user
Object.assign(user, permissions1, permissions2);

// now user = { name: "John", canView: true, canEdit: true }
```

Actual cloning

```
let user = {
    name: "John",
    age: 30
};

let clone = Object.assign({}, user);
```

Memory Management / Garbage Collection

Reachable variables:

1 Base set of inherently reachable values, that cannot be deleted for obvious reasons.

For instance:

- Local variables and parameters of the current function.
- Variables and parameters for other functions on the current chain of nested calls.
- Global variables.
- (there are some other, internal ones as well)

These values are called *roots*.

2 Any other value is considered reachable if it’s reachable
from a root by a reference or by a chain of references

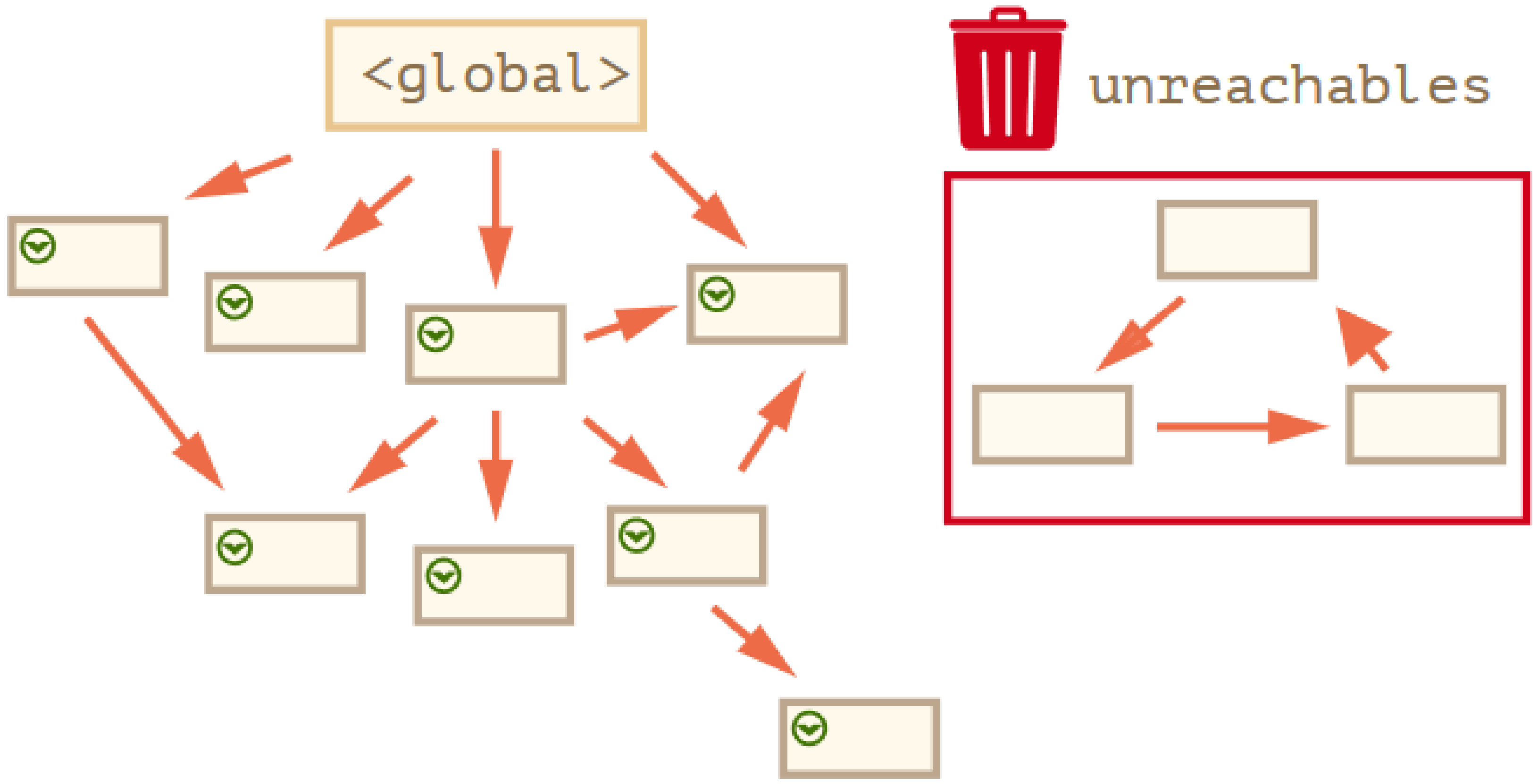
Interlinked objects

```
function match(contestant1,contestant2) {
    contestant1.opponent = contestant2;
    contestant2.opponent = contestant1;

    return {
        home: contestant1,
        away: contestant2
    }
}
```

```
let game = match({
    name: "John"
}, {
    name: "David"
});
```

Difficulties



Optimizations

- **Generational collection**
- **Incremental collection**
- **Idle-time collection**

Functions and objects

```
let user = {
  name: "John",
  age: 30
};

user.sayHi = function() {
  alert("Hello!");
};

user.sayHi(); // Hello!
```

Methods

```
// these objects do the same

user = {
  sayHi: function() {
    alert("Hello");
  }
};

user = {
  sayHi() { // same as "sayHi: function()"
    alert("Hello");
  }
};
```

this

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }

};

user.sayHi(); // John
```

Who is this?

```
function sayHi() {
  alert( this.name );
}
```

this is a calling object!

```
let user = { name: "John" };
let admin = { name: "Admin" };

function sayHi() {
  alert( this.name );
}


user.f = sayHi;
admin.f = sayHi;

user.f();
admin.f();
```

this in arrow functions

```
let user = {
  firstName: “John",
  sayHi() {
    let arrow = () => alert(this.firstName);
    arrow();
  }
};

user.sayHi();
```

Constructors

```
function User(name) {
  this.name = name;
  this.isAdmin = false;
}

let user = new User("Jack");

alert(user.name); // Jack
alert(user.isAdmin); // false
```

1.Named with capital letter first.
2. Should be executed only with "new" operator.

Safe properties navigation

```
let user = {}; // the user happens to be without address

alert(user.address.street); // Error!

alert( user && user.address && user.address.street );

alert( user?.address?.street );
```

?. can’t be used for assigment

```
user?.name = "John"; // Error, doesn't work
```

Objects to primitives

```
user.toPrimitive();
user.toString();
user.valueOf();
```

References

- https://javascript.info/
- https://developer.mozilla.org/
- https://www.chaijs.com/