ASYNC JAVASCRIPT

Beograd, 25.10.2019.

AGENDA SLIDE

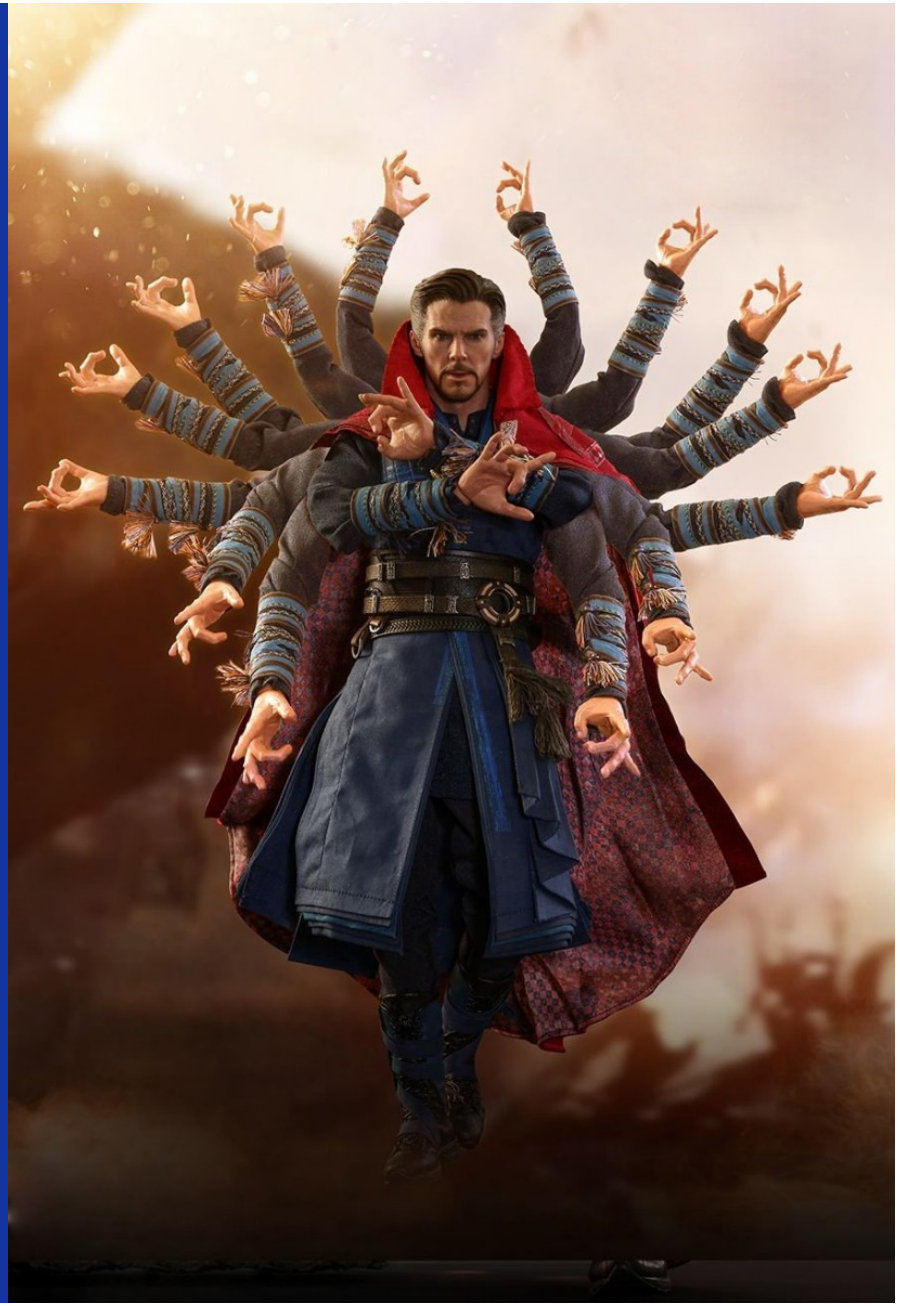
01 Event loop

02 Callbacks

03 Promises

01

EVENT LOOP



EVENT LOOP

```
function now() {
    return 21;
}

function later() {
    answer = answer * 2;
    console.log( "MLater answer:", answer );
}

var answer = now();

setTimeout( later, 1000 );
```

EVENT LOOP

- JavaScript itself has actually never had any direct notion of asynchrony built into it.The JS engine itself has never done anything more than execute a single chunk of your program at any given moment
- The JS engine doesn’t run in isolation. It runs inside a hosting environment(browsers or NodeJs)

EVENT LOOP

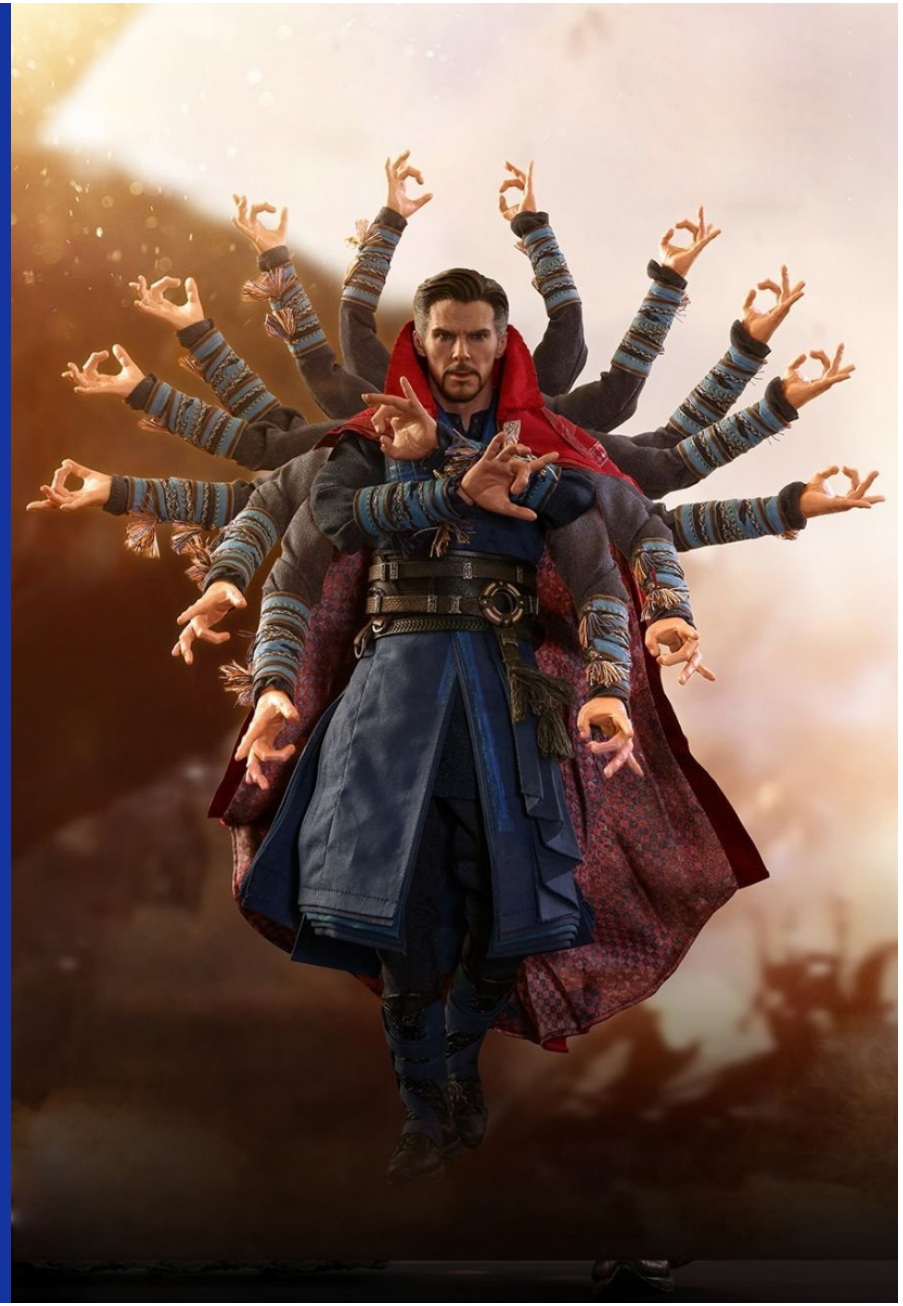
```
var eventLoop = [ ];
var event;

while (true) {
    if (eventLoop.length > 0) {
        event = eventLoop.shift();

        try {
            event();
        }
        catch (err) {
            reportError(err);
        }
    }
}
```

EVENT LOOP

```
var a = 1;
var b = 2;

function foo() {
    a++;
    b = b * a;
    a = b + 3;
}

function bar() {
    b--;
    a = 8 + b;
    b = a * 2;
}

ajax( "http://some.url.1", foo );
ajax( "http://some.url.2", bar );
```

02

CALLBACKS



CALLBACKS

```
// A
ajax( "..", function(..){
    // C
} );
// B

----------------------------------

// A
setTimeout( function(){
    // C
}, 1000 );
// B
```

NESTED/CHAINED CALLBACKS

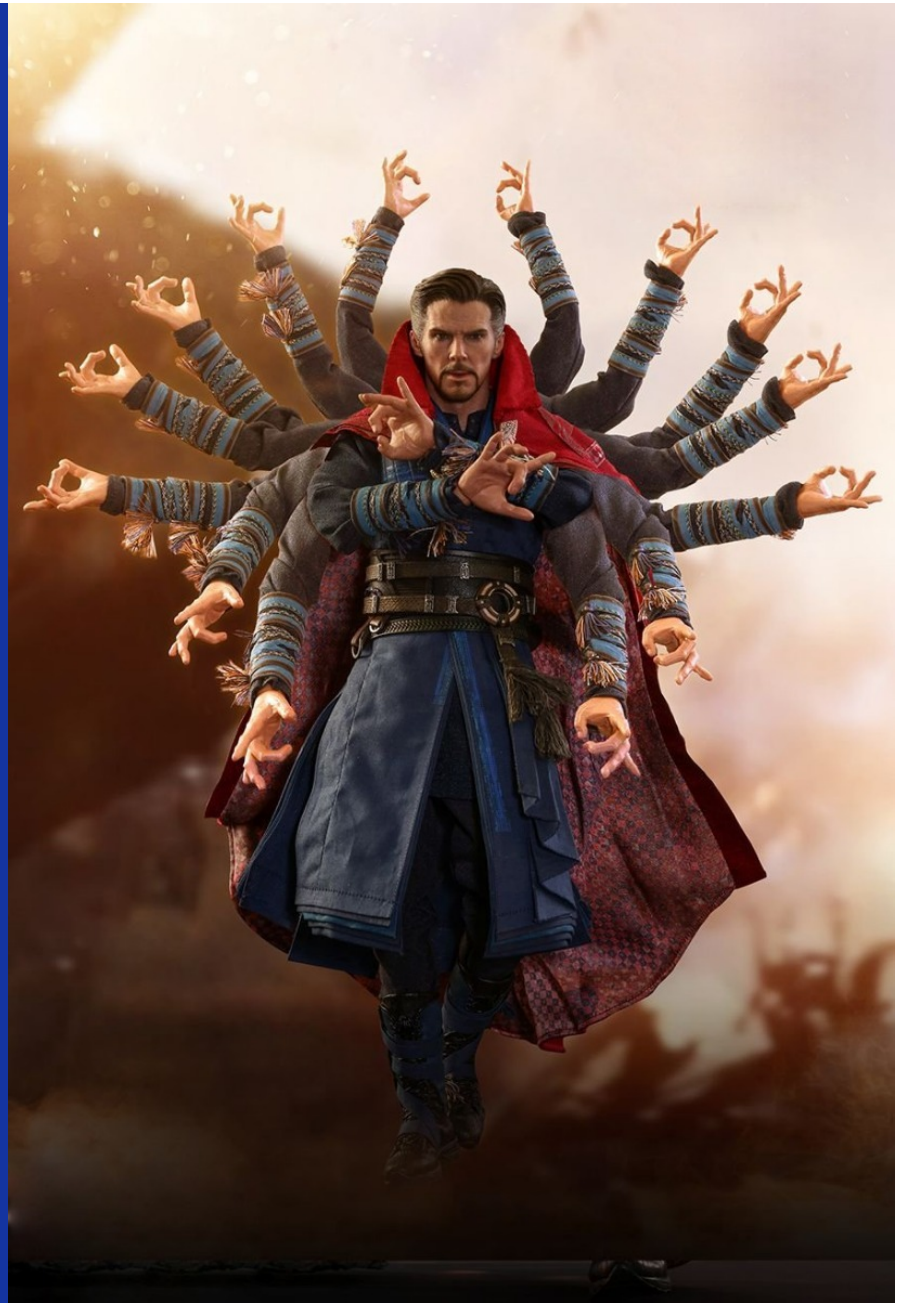
```
listen( "click", function handler(evt){
    setTimeout( function request(){
        ajax( "http://some.url.1", function response(text){
            if (text == "hello") {
                handler();
            }
            else if (text == "world") {
                request();
            }
        } );
    }, 500) ;
} );
```

NESTED/CHAINED CALLBACKS

```
doA( function(){
    doB();

    doC( function(){
        doD();
    } )

    doE();
} );

doF();
```

03

PROMISES



PROMISES

```
var x, y = 2;

console.log( x + y );
```

PROMISES

```
function add(getX,getY,cb) {
    var x, y;
    getX( function(xVal){
        x = xVal;

        if (y != undefined) {
            cb( x + y );
        }
    } );
    getY( function(yVal){
        y = yVal;

        if (x != undefined) {
            cb( x + y );
        }
    } );
}

add( fetchX, fetchY, function(sum){
    console.log( sum );
} );
```

PROMISES

```
function add(xPromise,yPromise) {
    // `Promise.all([ .. ])` takes an array of promises,
    // and returns a new promise that waits on them
    // all to finish
    return Promise.all( [xPromise, yPromise] )

    // when that promise is resolved, let's take the
    // received `X` and `Y` values and add them together.
    .then( function(values){
        // `values` is an array of the messages from the
        // previously resolved promises
        return values[0] + values[1];
    } );
}

// `fetchX()` and `fetchY()` return promises for
// their respective values, which may be ready
// now or later.
add( fetchX(), fetchY() )

// we get a promise back for the sum of those
// two numbers.
// now we chain-call `then(..)` to wait for the
// resolution of that returned promise.
.then( function(sum){
    console.log( sum ); // that was easier!
} );
```

PROMISES

```
add( fetchX(), fetchY() )
.then(
    // fullfillment handler
    function(sum) {
        console.log( sum );
    },
    // rejection handler
    function(err) {
        console.error( err ); // bummer!
    }
);
```

PROMISE “EVENTS”

```
function foo(x) {
    // start doing something that could take a while

    // construct and return a promise
    return new Promise( function(resolve,reject){
        // eventually, call `resolve(..)` or `reject(..)`,
        // which are the resolution callbacks for
        // the promise.
    } );
}

var p = foo( 42 );

bar( p );

baz( p );
```

PROMISE TRUST

- Call the callback too early
- Call the callback too late (or never)
- Call the callback too few or too many times
- Fail to pass along any necessary environment/parameters
- Swallow any errors/exceptions that may happen

CALL THE CALLBACK TOO EARLY

```
var promise = new Promise(
  function(resolve){
    resolve(42);
  }
);

promise.then(function(value) {
  console.log(value);
});
```

CALL THE CALLBACK TOO LATE (OR NEVER)

```
p.then( function(){
    p.then( function(){
        console.log( "C" );
    } );

    console.log( "A" );
} );

p.then( function(){
    console.log( "B" );
} );
```

SWALLOWING ANY ERRORS/EXCEPTIONS

```
var p = new Promise( function(resolve,reject){
    foo.bar();  // `foo` is not defined, so error!
    resolve( 42 );  // never gets here :(
} );

p.then(
    function fulfilled(){
        // never gets here :(
    },
    function rejected(err){
        // `err` will be a `TypeError` exception object
        // from the `foo.bar()` line.
    }
);
```

CHAIN FLOW

```
var p = Promise.resolve( 21 );

var p2 = p.then( function(v){
    console.log( v );   // 21

    // fulfill `p2` with value `42`
    return v * 2;
} );

// chain off `p2`
p2.then( function(v){
    console.log( v );   // 42
} );
```

CHAIN FLOW

```
var p = Promise.resolve( 21 );

p
.then( function(v){
    console.log( v );   // 21

    // fulfill the chained promise with value `42`
    return v * 2;
} )
// here's the chained promise
.then( function(v){
    console.log( v );   // 42
} );
```

CHAIN FLOW

```
var p = Promise.resolve( 21 );

p.then( function(v){
    console.log( v );   // 21

    // create a promise and return it
    return new Promise( function(resolve,reject){
        // fulfill with value `42`
        resolve( v * 2 );
    } );
} )
.then( function(v){
    console.log( v );   // 42
} );
```

CHAIN FLOW

```
var p = Promise.resolve( 21 );

p.then( function(v){
    console.log( v );   // 21

    // create a promise to return
    return new Promise( function(resolve,reject){
        // introduce asynchrony!
        setTimeout( function(){
            // fulfill with value `42`
            resolve( v * 2 );
        }, 100 );
    } );
} )
.then( function(v){
    // runs after the 100ms delay in the previous step
    console.log( v );   // 42
} );
```

CHAIN FLOW

```
function delay(time) {
    return new Promise( function(resolve,reject){
        setTimeout( resolve, time );
    } );
}

delay( 100 ) // step 1

.then( function STEP2(){
    console.log( "step 2 (after 100ms)" );
    return delay( 200 );
} )

.then( function STEP3(){
    console.log( "step 3 (after another 200ms)" );
} )

.then( function STEP4(){
    console.log( "step 4 (next Job)" );
    return delay( 50 );
} )

.then( function STEP5(){
    console.log( "step 5 (after another 50ms)" );
} )
```

CHAIN FLOW

```
// assume an `ajax( {url}, {callback} )` utility

// Promise-aware ajax
function request(url) {
    return new Promise( function(resolve,reject){
        // the `ajax(..)` callback should be our
        // promise's `resolve(..)` function
        ajax( url, resolve );
    } );
}

request( "http://some.url.1/" )

.then( function(response1){
    return request( "http://some.url.2/?v=" + response1 );
} )

.then( function(response2){
    console.log( response2 );
} );
```

PROMISE REJECTION

```
var rejectedPr = new Promise( function(resolve,reject){

    resolve( "Oops" ) );
} );

rejectedPr.then(
    function fulfilled(){
        // never gets here
    },

    function rejected(err){
        console.log( err ); // "Oops"
    }
);
```

PROMISE REJECTION

```
  var rejectedPr = new Promise( function(resolve,reject){
    // resolve this promise with a rejected promise
    resolve( Promise.reject( "Oops" ) );
} );

rejectedPr.then(
    function fulfilled(){
        // never gets here
    },

    function rejected(err){
        console.log( err ); // "Oops"
    }
  );
```

PROMISE.ALL

```
var p1 = request( "http://some.url.1/" );
var p2 = request( "http://some.url.2/" );

Promise.all( [p1,p2] )
.then( function(msgs){
    // both `p1` and `p2` fulfill and pass in
    // their messages here
    return request(
        "http://some.url.3/?v=" + msgs.join(",")
    );
} )

.then( function(msg){
    console.log( msg );
} );
```

PROMISE.RACE

```
var p1 = request( "http://some.url.1/" );
var p2 = request( "http://some.url.2/" );

Promise.race( [p1,p2] )

.then( function(msg){
    // either `p1` or `p2` will win the race
    return request(
        "http://some.url.3/?v=" + msg
    );
} )

.then( function(msg){
    console.log( msg );
} );
```

LITERATURE

- Eloquent JavaScript: https://eloquentjavascript.net
- You Don't Know JS: https://github.com/getify/You-Dont-Know-JS/tree/1st-ed

THANK YOU

levi nine

Technology Services