Javascript is an event driven language and this is the most important thing that makes node.js an awesome toolkits. Javascript works like human. For example if there are 5 things to do, they are

|  |  |
| --- | --- |
| 1  2  3  4  5 | 1. Make cup noodles( takes 3 mins )  2. Answer a phone call( takes 1 min )  3. Go to toilet( takes 20 secs )  4. Eat the noodles( takes 5 mins )  5. Throw the cup to trash can( takes 3 secs ) |

A normal person will first make noodles and while waiting the water to boil he can answer a phone call and go to toilet. When it’s ready he then eat the noodles and throw the cup to trash can. However with non-event driven language for example PHP is not that easy to achieve that.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | // Start making cup noodles  make\_cup\_noodles();    // While waiting for the noodles to be ready the phone rings,  // but I have to wait until the noodles to be cooked so I missed the phone call.  answer\_a\_phone\_call();    // At the same time I feel like I need to pee but I still have to wait for the noodles.  // So I wet my pants...  go\_to\_toilet();    // Then I eat the noodles with my wet pants  eat\_the\_noodles();    // and throw the cup to trash can with my wet pants  throw\_the\_cup\_to\_trash\_can(); |

With event driven language like javascript things would be different. However the above code still does not work right.

steps.js | Contents all the steps we are going to do

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | module.exports = {      cup\_noodles : 'This is the not yet cooked cup noodles',      make\_cup\_noodles : function( callback ){      var self = this;        console.log( 'Start making cup noodles' );        // simulate a time consuming function      setTimeout( function(){        self.cup\_noodles = self.cup\_noodles === 'This is the not yet cooked cup noodles' ?          'Cup noodles are ready' : self.cup\_noodles;          console.log( self.cup\_noodles );          callback && callback.call( this );      }, 3000 );    },      answer\_a\_phone\_call : function(){      var action = this.ringing === 'Ringing...' ?          'I answered the phone call' : 'I missed the phone call';          console.log( action );    },      go\_to\_toilet : function(){      this.pants = 'Off';    },      eat\_the\_noodles : function( callback ){      var self = this;        setTimeout( function(){        self.cup\_noodles = self.cup\_noodles === 'Cup noodles are ready' ?          'I finished eating' : 'I ate nothing...';          console.log( self.cup\_noodles );          callback && callback.call( this );      }, 5000 );    },      throw\_the\_cup\_to\_trash\_can : function(){      var self = this;        setTimeout( function(){        self.cup\_noodles = self.cup\_noodles === 'I finished eating' ?          'The cup noodles are finished and are in the trash can now' :          'The cup noodles are wasted';          console.log( self.cup\_noodles );      }, 30 );    }  }; |

pee.js | Pee action inside

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | module.exports = {      action : '',      pants : 'On',      explode : function( callback ){      var self = this;        callback && callback.call( this );        this.action = this.pants === 'On' ?        'Peeing on my pants' : 'Releasing...';        console.log( this.action );        setTimeout( function(){        self.pants = self.action === 'Peeing on my pants' ?          'I wet my pants' : 'I finished peeing';          console.log( self.pants );      }, 500 );    }  }; |

phone.js | Phone ring action

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18 | module.exports = {      ringing : '',      ring : function( callback ){      var self = this;        this.ringing = 'Ringing...';      console.log( this.ringing );        callback && callback.call( this );        setTimeout( function(){        self.ringing = 'Ringing stopped';        console.log( self.ringing );      }, 1000 );    }  }; |

wrong.js | Demonstrate the wrong operation

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | var steps = require( './steps' ),      phone = require( './phone' ),      pee = require( './pee' );    // Start making cup noodles  steps.make\_cup\_noodles();    // Yes I am now event driven so I don't have to wait for the noodles to be ready  // But we have a new problem here,  // with the same scope of functions they are triggered nearly at the same time.  phone.ring();    // Which means I might not be able to answer the phone call  steps.answer\_a\_phone\_call();    // Feel like going to the toilet  pee.explode();    // Same thing happens here  steps.go\_to\_toilet();    // I want to eat the noodles but it's not ready yet  steps.eat\_the\_noodles();    // The same thing happens here, I throw the cup noodles that is still being cooked.  // So in the end I still wet my paint and did not answer the phone call  // plus I still can't have my cup noodles  steps.throw\_the\_cup\_to\_trash\_can(); |

Result

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Start making cup noodles  Ringing...  I missed the phone call  Peeing on my pants  The cup noodles are wasted  I wet my pants  Ringing stopped  The cup noodles are wasted  I ate nothing... |

To make sure one thing happens after another, call those functions in a callback.

right.js | Demonstrate the right way

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | var steps = require( './steps' ),      phone = require( './phone' ),      pee = require( './pee' );    steps.make\_cup\_noodles( function(){    steps.eat\_the\_noodles( function(){      steps.throw\_the\_cup\_to\_trash\_can();    });  });    phone.ring( steps.answer\_a\_phone\_call );    pee.explode( steps.go\_to\_toilet ); |

Result

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Start making cup noodles  Ringing...  I answered the phone call  Releasing...  I finished peeing  Ringing stopped  Cup noodles are ready  I finished eating  The cup noodles are finished and are in the trash can now |

Hope the above example shows you how a event driven language works and why node.js is so fast

node.js Events

If you look at the [**node.js doc**](http://nodejs.org/docs/v0.6.1/api/), there is an [**events**](http://nodejs.org/docs/v0.6.1/api/events.html) section. I’m not going to go through each api of node.js events module here. Instead I’ll explain why and when to use this module.

From the previous example we know if we want to make sure one thing happens after another, we have to write it in the callback. But how if we have multiple and complex actions? We would end up with nested callbacks like the following

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | do\_a( function(){    do\_b( function(){      do\_c( function(){        do\_d( function(){          ...        });      });    });  }); |

It looks ugly and is tighten up. It’s not only hard to split the code into modules but also hard to extend the functionality. This is how node.js events comes in handy. With[**EventEmitter**](http://nodejs.org/docs/v0.6.1/api/events.html#events.EventEmitter) we can change the above code to the following

event.js

|  |  |
| --- | --- |
| 1  2  3 | var event = require( 'events' ).EventEmitter;    module.exports = new event; |

do\_a.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var event = require( './event' );    module.exports = function(){    console.log( 'we are going to call do\_a' );    event.emit( 'do\_a' );  }; |

do\_b.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var event = require( './event' );    module.exports = function(){    event.on( 'do\_a', function(){      console.log( 'we are going to call do\_b' );      event.emit( 'do\_b' );    });  }; |

do\_c.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var event = require( './event' );    module.exports = function(){    event.on( 'do\_b', function(){      console.log( 'we are going to call do\_c' );      event.emit( 'do\_c' );    });  }; |

do\_d.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var event = require( './event' );    module.exports = function(){    event.on( 'do\_c', function(){      console.log( 'we are going to call do\_d' );      event.emit( 'do\_d' );    });  }; |

run.js

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | var event, dos;    event = require( './event' );  todos = [ './do\_d', './do\_c', './do\_b', './do\_a' ];    event.on( 'do\_a', function(){    console.log( 'i can do something to do\_a out side of do\_a' );  });    event.on( 'do\_b', function(){    console.log( 'i can do something to do\_a out side of do\_b' );  });    event.on( 'do\_c', function(){    console.log( 'i can do something to do\_a out side of do\_c' );  });    event.on( 'do\_d', function(){    console.log( 'i can do something to do\_a out side of do\_d' );  });    todos.forEach( function( name ){    require( name )();  }); |

Result of calling run.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | we are going to call do\_a  i can do something to do\_a out side of do\_a  we are going to call do\_b  i can do something to do\_a out side of do\_b  we are going to call do\_c  i can do something to do\_a out side of do\_c  we are going to call do\_d  i can do something to do\_a out side of do\_d |

From the above result it might appear to you that it seems to be more complicated. Yes, for a small project it is. But for a larger project we can split our code into other files and still keep the ordering. It is a must for building flexible applications