link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=139 sentences=453, words=3595

1. webpack的插件机制

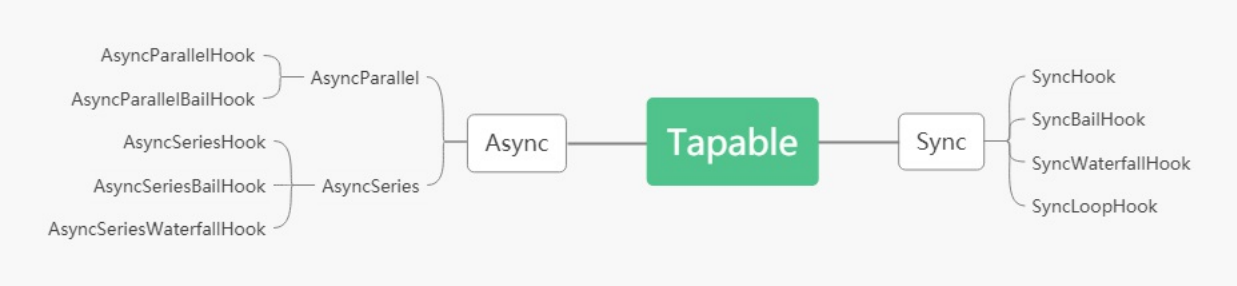
- 在具体介绍webpack内置插件与钩子可视化工具之前，我们先来了解一下webpack中的插件机制。 webpack实现插件机制的大体方式是：
    - 创建 - webpack在其内部对象上创建各种钩子；
    - 注册 - 插件将自己的方法注册到对应钩子上，交给webpack；
    - 调用 - webpack编译过程中，会适时地触发相应钩子，因此也就触发了插件的方法。
- Webpack本质上是一种事件流的机制，它的工作流程就是将各个插件串联起来，而实现这一切的核心就是Tapable，webpack中最核心的负责编译的Compiler和负责创建bundle的Compilation都是Tapable的实例
- 通过事件和注册和监听，触发webpack生命周期中的函数方法

```
const {
    SyncHook,
    SyncBailHook,
    SyncWaterfallHook,
    SyncLoopHook,
    AsyncParallelHook,
    AsyncParallelBailHook,
    AsyncSeriesHook,
    AsyncSeriesBailHook,
    AsyncSeriesWaterfallHook
} = require('tapable');
```

2. tapable分类

2.1 按同步异步分类

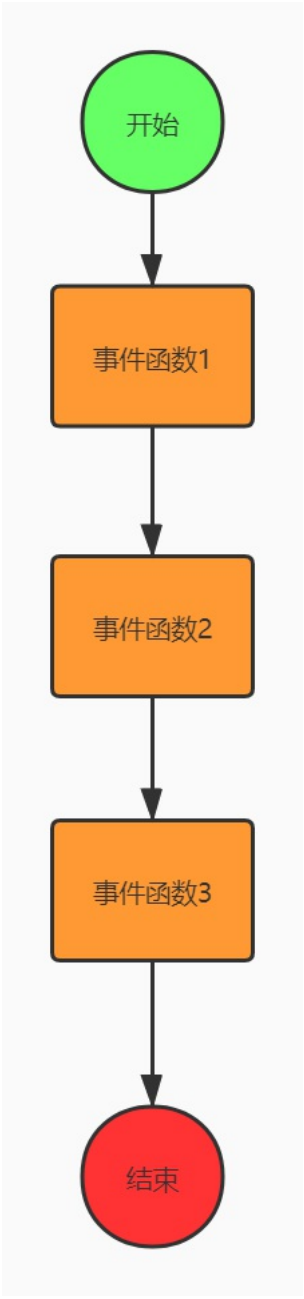
- Hook 类型可以分为 同步Sync和 异步Async，异步又分为 并行和 串行



2.1 按返回值分类

2.1.1 Basic

- 执行每一个事件函数，不关心函数的返回值,有 SyncHook、AsyncParallelHook、AsyncSeriesHook

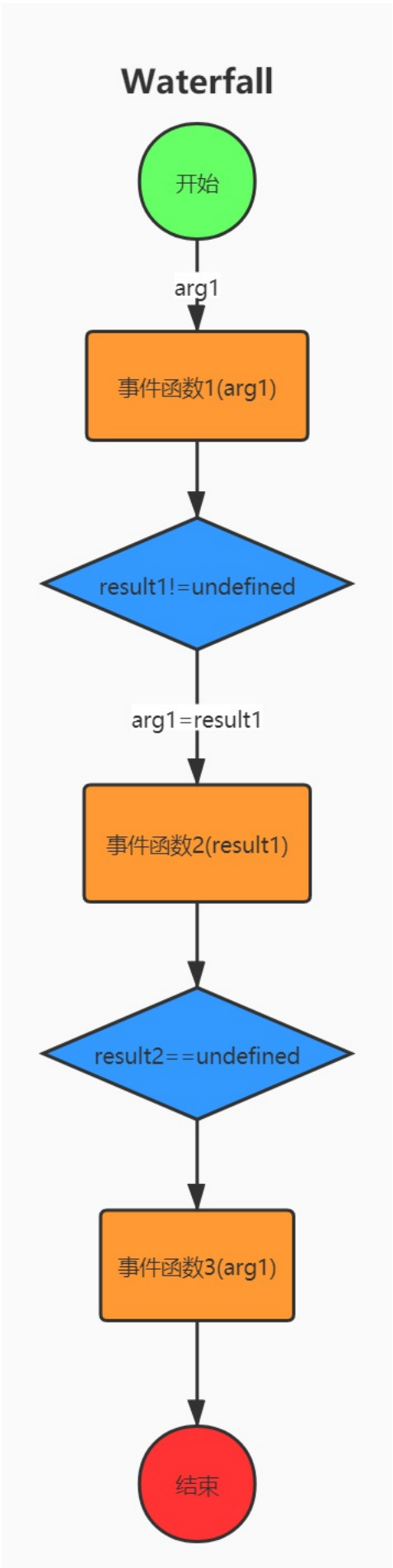


2.1.2 Bail

- 执行每一个事件函数，遇到第一个结果 `result !== undefined` 则返回，不再继续执行。有：SyncBailHook、AsyncSeriesBailHook, AsyncParallelBailHook

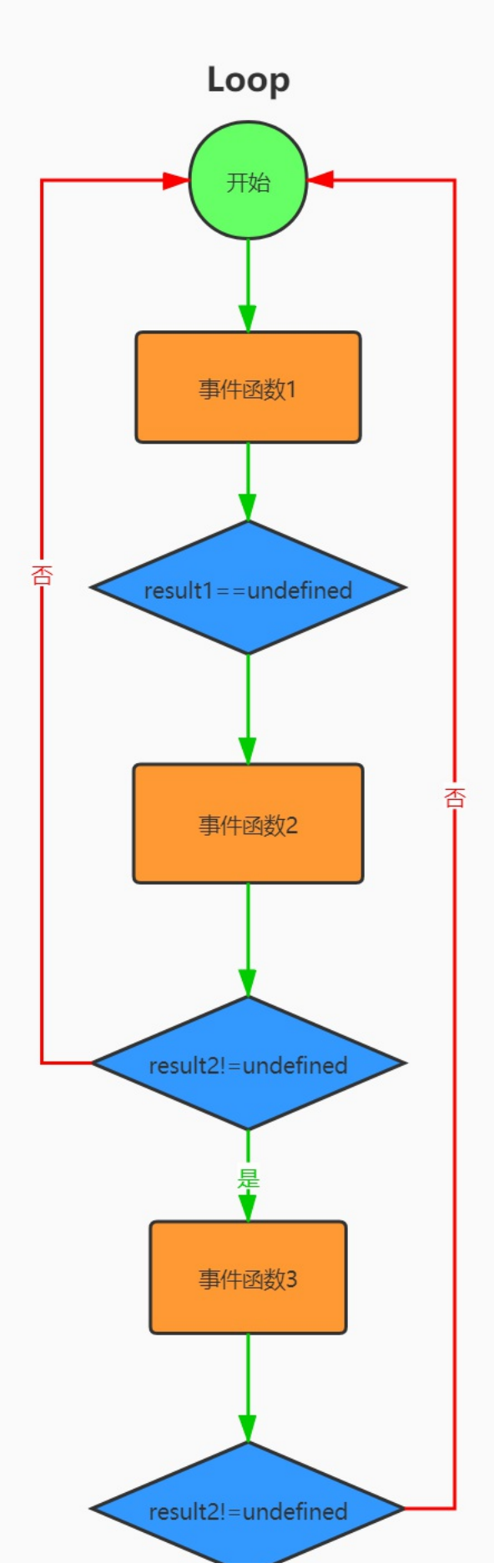
2.1.3 Waterfall

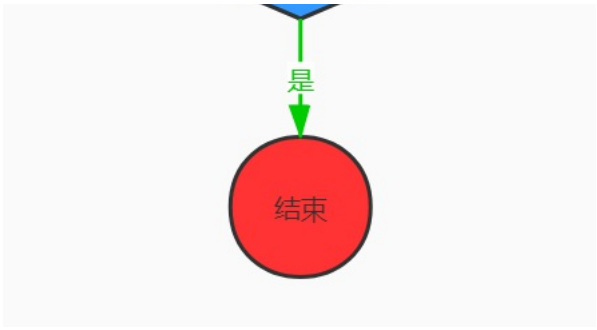
- 如果前一个事件函数的结果 `result !== undefined`,则 result 会作为后一个事件函数的第一个参数,有SyncWaterfallHook，AsyncSeriesWaterfallHook


Waterfall
开始
arg1
事件函数1(arg1)
result1!=undefined
arg1=result1
事件函数2(result1)
result2==undefined
事件函数3(arg1)
结束

2.1.4 Loop

- 不停的循环执行事件函数，直到所有函数结果 `result === undefined`,有SyncLoopHook 和 AsyncSeriesLoopHook





3.使用

3.1 SyncHook

- 所有的构造函数都接收一个可选参数，参数是一个参数名的字符串数组
- 参数的名字可以任意填写，但是参数数组的长数必须要根实际接受的参数个数一致
- 如果回调函数不接受参数，可以传入空数组
- 在实例化的时候传入的数组长度长度有用，值没有用途
- 执行call时，参数个数和实例化时的数组长度有关
- 回调的时候是按先入先出的顺序执行的，先放的先执行

```
const {SyncHook} = require('tapable');
const hook = new SyncHook(['name','age']);
hook.tap('1',(name,age)=>{
    console.log(1,name,age);
    return 1;
});
hook.tap('2',(name,age)=>{
    console.log(2,name,age);
    return 2;
});
hook.tap('3',(name,age)=>{
    console.log(3,name,age);
    return 3;
});

hook.call('zhufeng',10);
```

```
1 zhufeng 10
2 zhufeng 10
3 zhufeng 10
```

3.2 SyncBailHook

- BailHook中的回调函数也是顺序执行的
- 调用call时传入的参数也可以传给回调函数
- 当回调函数返回非undefined值的时候会停止调用后续的回调

```
const {SyncBailHook} = require('tapable');
const hook = new SyncBailHook(['name','age']);
hook.tap('1',(name,age)=>{
    console.log(1,name,age);

});
hook.tap('2',(name,age)=>{
    console.log(2,name,age);
    return 2;
});
hook.tap('3',(name,age)=>{
    console.log(3,name,age);
    return 3;
});

hook.call('zhufeng',10);
```

3.3 SyncWaterfallHook

- SyncWaterfallHook表示如果上一个回调函数的结果不为undefined,则可以作为下一个回调函数的第一个参数
- 回调函数接受的参数来自于上一个函数的结果
- 调用call传入的第一个参数，会被上一个函数的非undefined结果替换
- 当回调函数返回非undefined不会停止回调栈的调用

```
const {SyncWaterfallHook} = require('tapable');

const hook = new SyncWaterfallHook(['name','age']);
hook.tap('1',(name,age)=>{
    console.log(1,name,age);
    return 1;
});
hook.tap('2',(name,age)=>{
    console.log(2,name,age);
    return ;
});
hook.tap('3',(name,age)=>{
    console.log(3,name,age);
    return 3;
});

hook.call('zhufeng',10);
```

3.4 SyncLoopHook

- SyncLoopHook的特点是不停的循环执行回调函数，直到函数结果等于undefined
- 要注意的是每次循环都是从头开始循环的

```
const { SyncLoopHook } = require('tapable');

let hook = new SyncLoopHook(['name', 'age']);
let counter1 = 0;
let counter2 = 0;
let counter3 = 0;
hook.tap('1', (name, age) => {
    console.log(1, 'counter1', counter1);
    if (++counter1 == 1) {
        counter1 = 0
        return;
    }
    return true;
});
hook.tap('2', (name, age) => {
    console.log(2, 'counter2', counter2);
    if (++counter2 == 2) {
        counter2 = 0
        return;
    }
    return true;
});
hook.tap('3', (name, age) => {
    console.log(3, 'counter3', counter3);
    if (++counter3 == 3) {
        counter3 = 0
        return;
    }
    return true;
});
hook.call('zhufeng', 10);
```

```
1 counter1 0
2 counter2 0
1 counter1 0
2 counter2 1
3 counter3 0
1 counter1 0
2 counter2 0
1 counter1 0
2 counter2 1
3 counter3 1
1 counter1 0
2 counter2 0
1 counter1 0
2 counter2 1
3 counter3 2
```

3.5 AsyncParallelHook

- 异步并行执行钩子

3.5.1 tap

- 同步注册

```
let {AsyncParallelHook}=require('tapable');
let queue = new AsyncParallelHook(['name']);
console.time('cost');
queue.tap('1',function(name){
  console.log(1);
});
queue.tap('2',function(name){
  console.log(2);
});
queue.tap('3',function(name){
  console.log(3);
});
queue.callAsync('zfpx',err=>{
  console.log(err);
  console.timeEnd('cost');
});
```

3.5.2 tapAsync

- 异步注册，全部任务完成后执行最终的回调

```
let {AsyncParallelHook}=require('tapable');
let queue = new AsyncParallelHook(['name']);
console.time('cost');
queue.tapAsync('1',function(name,callback){
  setTimeout(function(){
      console.log(1);
      callback();
  },1000)
});
queue.tapAsync('2',function(name,callback){
  setTimeout(function(){
      console.log(2);
      callback();
  },2000)
});
queue.tapAsync('3',function(name,callback){
  setTimeout(function(){
      console.log(3);
      callback();
  },3000)
});
queue.callAsync('zfpx',err=>{
  console.log(err);
  console.timeEnd('cost');
});
```

3.5.3 tapPromise

- promise注册钩子
- 全部完成后执行才算成功

```
let {AsyncParallelHook}=require('tapable');
let queue = new AsyncParallelHook(['name']);
console.time('cost');
queue.tapPromise('1',function(name){
    return new Promise(function(resolve,reject){
        setTimeout(function(){
            console.log(1);
            resolve();
        },1000)
    });

});
queue.tapPromise('2',function(name){
    return new Promise(function(resolve,reject){
        setTimeout(function(){
            console.log(2);
            resolve();
        },2000)
    });
});
queue.tapPromise('3',function(name){
    return new Promise(function(resolve,reject){
        setTimeout(function(){
            console.log(3);
            resolve();
        },3000)
    });
});
queue.promise('zfpx').then(()=>{
    console.timeEnd('cost');
})
```

3.6 AsyncParallelBailHook

- 带保险的异步并行执行钩子
- 有一个任务返回值不为空就直接结束

3.6.1 tap

- 如果有一个任务有返回值则调用最终的回调

```
let {AsyncParallelBailHook} = require('tapable');
let queue=new AsyncParallelBailHook(['name']);
console.time('cost');
queue.tap('1',function(name){
  console.log(1);
  return "Wrong";
});
queue.tap('2',function(name){
  console.log(2);
});
queue.tap('3',function(name){
  console.log(3);
});
queue.callAsync('zfpx',err=>{
  console.log(err);
  console.timeEnd('cost');
});
```

3.6.2 tapAsync

- 有一个任务返回错误就直接调最终的回调

```
let {AsyncParallelBailHook} = require('tapable');
let queue=new AsyncParallelBailHook(['name']);
console.time('cost');
queue.tapAsync('1',function(name,callback){
  console.log(1);
  callback('Wrong');
});
queue.tapAsync('2',function(name,callback){
  console.log(2);
  callback();
});
queue.tapAsync('3',function(name,callback){
  console.log(3);
  callback();
});
queue.callAsync('zfpx',err=>{
  console.log(err);
  console.timeEnd('cost');
});
```

** 3.6.3 tapPromise #**

- 只要有一个任务有resolve或者reject值，不管成功失败都结束

```
let { AsyncParallelBailHook } = require('tapable');
let queue = new AsyncParallelBailHook(['name']);
console.time('cost');
queue.tapPromise('1', function (name) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            console.log(1);
            resolve(1);
        }, 1000)
    });
});
queue.tapPromise('2', function (name) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            console.log(2);
            resolve();
        }, 2000)
    });
});

queue.tapPromise('3', function (name) {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            console.log(3);
            resolve();
        }, 3000)
    });
});

queue.promise('zfpx').then((result) => {
    console.log('成功', result);
    console.timeEnd('cost');
}, err => {
    console.error('失败', err);
    console.timeEnd('cost');
})
```

3.7 AsyncSeriesHook

- 异步串行钩子
- 任务一个一个执行,执行完上一个执行下一个

** 3.7.1 tap #**

```
let { AsyncSeriesHook } = require('tapable');
let queue = new AsyncSeriesHook(['name']);
console.time('cost');
queue.tap('1', function (name) {
    console.log(1);
});
queue.tap('2', function (name) {
    console.log(2);
});
queue.tap('3', function (name) {
    console.log(3);
});
queue.callAsync('zhufeng', err => {
    console.log(err);
    console.timeEnd('cost');
});
```

** 3.7.2 tapAsync #**

```
let { AsyncSeriesHook } = require('tapable');
let queue = new AsyncSeriesHook(['name']);
console.time('cost');
queue.tapAsync('1',function(name,callback){
   setTimeout(function(){
       console.log(1);
   },1000)
});
queue.tapAsync('2',function(name,callback){
    setTimeout(function(){
        console.log(2);
        callback();
    },2000)
});
queue.tapAsync('3',function(name,callback){
    setTimeout(function(){
        console.log(3);
        callback();
    },3000)
});
queue.callAsync('zfpx',err=>{
    console.log(err);
    console.timeEnd('cost');
});
```

** 3.7.3 tapPromise #**

```
let { AsyncSeriesHook } = require('tapable');
let queue = new AsyncSeriesHook(['name']);
console.time('cost');
queue.tapPromise('1', function (name) {
    return new Promise(function (resolve) {
        setTimeout(function () {
            console.log(1, name);
            resolve();
        }, 1000)
    });
});
queue.tapPromise('2', function (name) {
    return new Promise(function (resolve) {
        setTimeout(function () {
            console.log(2, name);
            resolve();
        }, 2000)
    });
});
queue.tapPromise('3', function (name) {
    return new Promise(function (resolve) {
        setTimeout(function () {
            console.log(3, name);
            resolve();
        }, 3000)
    });
});
queue.promise('zfpx').then(data => {
    console.log(data);
    console.timeEnd('cost');
});
```

3.8 AsyncSeriesBailHook

- 只要有一个返回了不为undefined的值就直接结束**3.8.1 tap #**

```
let {AsyncSeriesBailHook} = require('tapable');
let queue = new AsyncSeriesBailHook(['name']);
console.time('cost');
queue.tap('1',function(name){
  console.log(1);
  return "Wrong";
});
queue.tap('2',function(name){
  console.log(2);
});
queue.tap('3',function(name){
  console.log(3);
});
queue.callAsync('zfpx',err=>{
  console.log(err);
  console.timeEnd('cost');
});
```

3.8.1 tapAsync

```
let {AsyncSeriesBailHook}=require('tapable');
let queue = new AsyncSeriesBailHook(['name']);
console.time('cost');
queue.tapAsync('1',function(name,callback){
 setTimeout(function(){
     console.log(1);
     callback('wrong');
 },1000)
});
queue.tapAsync('2',function(name,callback){
  setTimeout(function(){
      console.log(2);
      callback();
  },2000)
});
queue.tapAsync('3',function(name,callback){
  setTimeout(function(){
      console.log(3);
      callback();
  },3000)
});
queue.callAsync('zfpx',err=>{
  console.log(err);
  console.timeEnd('cost');
});
```

3.8.1 tapPromise

```
let {AsyncSeriesBailHook} = require('tapable');
let queue = new AsyncSeriesBailHook(['name']);
console.time('cost');
queue.tapPromise('1',function(name){
 return new Promise(function(resolve){
     setTimeout(function(){
         console.log(1);
         resolve();
     },1000)
 });
});
queue.tapPromise('2',function(name,callback){
  return new Promise(function(resolve,reject){
      setTimeout(function(){
          console.log(2);
          reject('失败了');
      },2000)
  });
});
queue.tapPromise('3',function(name,callback){
  return new Promise(function(resolve){
      setTimeout(function(){
          console.log(3);
          resolve();
      },3000)
  });
});
queue.promise('zfpx').then(data=>{
  console.log(data);
  console.timeEnd('cost');
},error=>{
  console.log(error);
  console.timeEnd('cost');
});
```

3.9 AsyncSeriesWaterfallHook

- 只要有一个返回了不为undefined的值就直接结束**3.9.1 tap #**

```
let { AsyncSeriesWaterfallHook } = require('tapable');
let queue = new AsyncSeriesWaterfallHook(['name', 'age']);
console.time('cost');
queue.tap('1', function (name, age) {
  console.log(1, name, age);
  return 'return1';
});
queue.tap('2', function (data, age) {
  console.log(2, data, age);
  return 'return2';
});
queue.tap('3', function (data, age) {
  console.log(3, data, age);
});
queue.callAsync('zfpx', 10, err => {
  console.log(err);
  console.timeEnd('cost');
});
```

3.9.1 tapAsync

```
let { AsyncSeriesWaterfallHook } = require('tapable');
let queue = new AsyncSeriesWaterfallHook(['name', 'age']);
console.time('cost');
queue.tapAsync('1', function (name, age, callback) {
  setTimeout(function () {
      console.log(1, name, age);
      callback(null, 1);
  }, 1000)
});
queue.tapAsync('2', function (data, age, callback) {
  setTimeout(function () {
      console.log(2, data, age);
      callback(null, 2);
  }, 2000)
});
queue.tapAsync('3', function (data, age, callback) {
  setTimeout(function () {
      console.log(3, data, age);
      callback(null, 3);
  }, 3000)
});
queue.callAsync('zfpx', 10, (err, data) => {
  console.log(err, data);
  console.timeEnd('cost');
});
```

3.9.1 tapPromise

```
let {AsyncSeriesWaterfallHook} = require('tapable');
let queue = new AsyncSeriesWaterfallHook(['name']);
console.time('cost');
queue.tapPromise('1', function (name) {
  return new Promise(function (resolve) {
      setTimeout(function () {
          console.log(name, 1);
          resolve(1);
      }, 1000);
  });
});
queue.tapPromise('2', function (data) {
  return new Promise(function (resolve) {
      setTimeout(function () {
          console.log(data, 2);
          resolve(2);
      }, 2000);
  });
});
queue.tapPromise('3', function (data) {
  return new Promise(function (resolve) {
      setTimeout(function () {
          console.log(data, 3);
          resolve(3);
      }, 3000);
  });
});
queue.promise('zfpx').then(err => {
  console.timeEnd('cost');
});
```

4.SyncHook

1. 所有的构造函数都接收一个可选参数，参数是一个参数名的字符串数组
2. 参数的名字可以任意填写，但是参数数组的长数必须要根实际接受的参数个数一致
3. 如果回调函数不接受参数，可以传入空数组
4. 在实例化的时候传入的数组长度长度有用，值没有用途
5. 执行call时，参数个数和实例化时的数组长度有关
6. 回调的时候是按先入先出的顺序执行的，先放的先执行

4.1 使用

```
const { SyncHook } = require('tapable');
let syncHook = new SyncHook(["name", "age"]);
syncHook.tap("1", (name, age) => {
    console.log(1, name, age);
});
syncHook.tap("2", (name, age) => {
    console.log(2, name, age);
});
syncHook.call("zhufeng", 10);
```

```
(function anonymous(name, age
) {
;
var _context;
var _x = this._x;
var _fn0 = _x[0];
_fn0(name, age);
var _fn1 = _x[1];
_fn1(name, age);
})
```

4.2 实现

** 4.2.1 index.js #**

tapable\index.js

```
let SyncHook = require('./SyncHook');
module.exports = {
    SyncHook
}
```

** 4.2.2 Hook.js #**

tapable\Hook.js

```
class Hook {
    constructor(args) {
        if (!Array.isArray(args)) args = [];
        this.args = args;
        this.taps = [];
        this._x = undefined;
    }
    tap(options, fn) {
        if (typeof options === "string") options = { name: options };
        options.fn = fn;
        this._insert(options);
    }
    _insert(item) {
        this.taps[this.taps.length] = item;
    }
    call(...args) {
        let callMethod = this._createCall();
        return callMethod.apply(this, args);
    }
    _createCall() {
        return this.compile({
            taps: this.taps,
            args: this.args
        });
    }
}

module.exports = Hook;
```

** 4.2.3 SyncHook#**

tapable\SyncHook.js

```
const Hook = require("./Hook");
const HookCodeFactory = require("./HookCodeFactory");
const factory = new HookCodeFactory();
class SyncHook extends Hook {
    constructor(args) {
        super(args);
    }
    compile(options) {
        factory.setup(this, options);
        return factory.create(options);
    }
}
module.exports = SyncHook;
```

** 4.2.4 HookCodeFactory.js#**

tapable\HookCodeFactory.js

```
class HookCodeFactory {
    args() {
        return this.options.args.join(",");
    }
    setup(instance, options) {
        this.options = options;
        instance._x = options.taps.map(t => t.fn);
    }
    header() {
        return "var _x = this._x;\n";
    }
    content() {
        let code = "";
        for (let idx = 0; idx < this.options.taps.length; idx++) {
            code += `var _fn${idx} = _x[${idx}];\n
                _fn${idx}(${this.args()});\n`;
        }
        return code;
    }
    create() {
        return new Function(this.args(), this.header() + this.content());
    }
}
module.exports = HookCodeFactory;
```

5. AsyncParallelHook

5.1 使用

```
let { AsyncParallelHook } = require('./tapable');
let queue = new AsyncParallelHook(['name', 'age']);
console.time('cost');
queue.tapAsync('1', function (name, age, callback) {
    setTimeout(function () {
        console.log(1, name, age);
        callback();
    }, 1000)
});
queue.tapAsync('2', function (name, age, callback) {
    setTimeout(function () {
        console.log(2, name, age);
        callback();
    }, 2000)
});
queue.tapAsync('3', function (name, age, callback) {
    setTimeout(function () {
        console.log(3, name, age);
        callback();
    }, 3000)
});
queue.callAsync('zhufeng', 10, err => {
    console.timeEnd('cost');
});
```

```
(function anonymous(name, age, _callback
) {
    "use strict";
    var _x = this._x;
    do {
        var _counter = 3;
        var _done = () => {
            _callback();
        };
        if (_counter 0) break;
        var _fn0 = _x[0];
        _fn0(name, age, _err0 => {
            if (_err0) {
                if (_counter > 0) {
                    _callback(_err0);
                    _counter = 0;
                }
            } else {
                if (--_counter === 0) _done();
            }
        });
        if (_counter 0) break;
        var _fn1 = _x[1];
        _fn1(name, age, _err1 => {
            if (_err1) {
                if (_counter > 0) {
                    _callback(_err1);
                    _counter = 0;
                }
            } else {
                if (--_counter === 0) _done();
            }
        });
        if (_counter 0) break;
        var _fn2 = _x[2];
        _fn2(name, age, _err2 => {
            if (_err2) {
                if (_counter > 0) {
                    _callback(_err2);
                    _counter = 0;
                }
            } else {
                if (--_counter === 0) _done();
            }
        });
    } while (false);

})
```

5.2 实现

** 5.2.1 tapable\index.js#**

tapable\index.js

```
let SyncHook = require('./SyncHook');
+let AsyncParallelHook = require('./AsyncParallelHook');
module.exports = {
    SyncHook,
+    AsyncParallelHook
}
```

** 5.2.2 AsyncParallelHookCodeFactory.js#**

AsyncParallelHookCodeFactory.js

```
const HookCodeFactory = require("./HookCodeFactory");
class AsyncParallelHookCodeFactory extends HookCodeFactory {
    args({ before, after } = {}) {
        let allArgs = this.options.args || [];
        if (before) allArgs = [before, ...allArgs];
        if (after) allArgs = [...allArgs, after];
        if (allArgs.length === 0) {
            return "";
        } else {
            return allArgs.join(",");
        }
    }
    create() {
        return new Function(
            this.args({ after: "_callback" }),
            this.header() + this.content()
        );
    }
    content() {
        let code = ``;
        code += `
                var _counter = ${this.options.taps.length};
                var _done = () =>{
                    _callback();
                };
                `;
        for (let idx = 0; idx < this.options.taps.length; idx++) {
            code += `
                    var _fn${idx} = _x[${idx}];
                    _fn${idx}(${this.args()}, _err${idx} =>{
                        if (--_counter === 0) _done();
                    });
                `;
        }
        return code;
    }
}
module.exports = AsyncParallelHookCodeFactory;
```

** 5.2.3 AsyncParallelHook.js #**

AsyncParallelHook.js

```
const Hook = require("./Hook");
let AsyncParallelHookCodeFactory = require('./AsyncParallelHookCodeFactory');

const factory = new AsyncParallelHookCodeFactory();
class AsyncParallelHook extends Hook {
    constructor(args) {
        super(args);
    }
    tapAsync(options, fn) {
        if (typeof options === "string") options = { name: options };
        options.fn = fn;
        this._insert(options);
    }
    callAsync(...args) {
        let callMethod = this._createCall();
        return callMethod.apply(this, args);
    }
    compile(options) {
        factory.setup(this, options);
        return factory.create(options);
    }
}
module.exports = AsyncParallelHook;
```

6. AsyncParallelHook promise

6.1 使用

```
let { AsyncParallelHook } = require('tapable');
let queue = new AsyncParallelHook(['name', 'age']);
console.time('cost');
queue.tapPromise('1', function (name, age) {
    return new Promise(function (resolve) {
        setTimeout(function () {
            console.log(1, name, age);
            resolve();
        }, 1000)
    });
});
queue.tapPromise('2', function (name, age) {
    return new Promise(function (resolve) {
        setTimeout(function () {
            console.log(2, name, age);
            resolve();
        }, 2000)
    });
});
queue.tapPromise('3', function (name, age) {
    return new Promise(function (resolve) {
        setTimeout(function () {
            console.log(3, name, age);
            resolve();
        }, 3000)
    });
});
queue.promise('zhufeng', 10).then(result => {
    console.timeEnd('cost');
}, error => {
    console.log(error);
    console.timeEnd('cost');
});
```

```
(function anonymous(name, age
) {
    "use strict";
    return new Promise((_resolve, _reject) => {
        var _sync = true;
        function _error(_err) {
            if (_sync)
                _resolve(Promise.resolve().then(() => { throw _err; }));
            else
                _reject(_err);
        };
        var _x = this._x;
        do {
            var _counter = 3;
            var _done = () => {
                _resolve();
            };
            if (_counter < 1) break;
            var _fn0 = _x[0];
            var _hasResult0 = false;
            var _promise0 = _fn0(name, age);
            if (!_promise0 || !_promise0.then)
                throw new Error('Tap function (tapPromise) did not return promise (returned ' + _promise0 + ')');
            _promise0.then(_result0 => {
                _hasResult0 = true;
                if (--_counter === 0) _done();
            }, _err0 => {
                if (_hasResult0) throw _err0;
                if (_counter > 0) {
                    _error(_err0);
                    _counter = 0;
                }
            });
            if (_counter < 1) break;
            var _fn1 = _x[1];
            var _hasResult1 = false;
            var _promise1 = _fn1(name, age);
            if (!_promise1 || !_promise1.then)
                throw new Error('Tap function (tapPromise) did not return promise (returned ' + _promise1 + ')');
            _promise1.then(_result1 => {
                _hasResult1 = true;
                if (--_counter === 0) _done();
            }, _err1 => {
                if (_hasResult1) throw _err1;
                if (_counter > 0) {
                    _error(_err1);
                    _counter = 0;
                }
            });
            if (_counter < 1) break;
            var _fn2 = _x[2];
            var _hasResult2 = false;
            var _promise2 = _fn2(name, age);
            if (!_promise2 || !_promise2.then)
                throw new Error('Tap function (tapPromise) did not return promise (returned ' + _promise2 + ')');
            _promise2.then(_result2 => {
                _hasResult2 = true;
                if (--_counter === 0) _done();
            }, _err2 => {
                if (_hasResult2) throw _err2;
                if (_counter > 0) {
                    _error(_err2);
                    _counter = 0;
                }
            });
        } while (false);
        _sync = false;
    });

})
```

6.2 实现

** 6.2.1 tapable\index.js #**

tapable\index.js

```
let SyncHook = require('./SyncHook');
let AsyncParallelHook = require('./AsyncParallelHook');
+let AsyncParallelHookForPromise = require('./AsyncParallelHookForPromise');
module.exports = {
    SyncHook,
    AsyncParallelHook,
+   AsyncParallelHookForPromise
}
```

** 6.2.2 AsyncParallelHookCodeFactoryForPromise.js #**

doc\tapable\AsyncParallelHookCodeFactoryForPromise.js

```
const HookCodeFactory = require("./HookCodeFactory");
class AsyncParallelHookCodeFactory extends HookCodeFactory {
    args({ before, after } = {}) {
        let allArgs = this.options.args || [];
        if (before) allArgs = [before, ...allArgs];
        if (after) allArgs = [...allArgs, after];
        if (allArgs.length === 0) {
            return "";
        } else {
            return allArgs.join(",");
        }
    }
    create() {
        return new Function(this.args(), this.header() + this.content());
    }
    content() {
        let code = ``;
        code += `
                return new Promise((_resolve)=>{
                    var _counter = ${this.options.taps.length};
                    var _done = ()=>{
                        _resolve();
                    };
                `;

        for (let idx = 0; idx < this.options.taps.length; idx++) {
            code += `
                    var _fn${idx} = _x[${idx}];
                    var _promise${idx} = _fn${idx}(${this.args()});
                    _promise${idx}.then(_result${idx} =>{
                        if (--_counter === 0) _done();
                    });
                `;
        }
        code += `
    });
    `;
        return code;
    }
}
module.exports = AsyncParallelHookCodeFactory;
```

** 6.2.3 AsyncParallelHookForPromise.js #**

AsyncParallelHookForPromise.js

```
let AsyncParallelHookCodeFactoryForPromise = require('./AsyncParallelHookCodeFactoryForPromise');
let Hook = require('./Hook');
const factory = new AsyncParallelHookCodeFactoryForPromise();
class AsyncParallelHookForPromise extends Hook {
    constructor(args) {
        super(args);
    }
    tapPromise(options, fn) {
        if (typeof options === "string") options = { name: options };
        options.fn = fn;
        this._insert(options);
    }
    promise(...args) {
        let callMethod = this._createCall();
        return callMethod.apply(this, args);
    }
    compile(options) {
        factory.setup(this, options);
        return factory.create(options);
    }
}
module.exports = AsyncParallelHookForPromise;
```