---

link: null
title: 珠峰架构师成长计划
description: tapable
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=120 sentences=483, words=3294

---

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

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

tapable (http://img.zhufengpeixun.cn/tapable.png)

类型 使用要点 Basic 不关心监听函数的返回值 Bail 保险式: 只要监听函数中有返回值(不为undefined)，则跳过之后的监听函数 Waterfall 瀑布式: 上一步的返回值交给下一步使用 Loop 循环类型: 如果该监听函数返回true,则这个监听函数会反复执行，如果返回undefined则退出循环

3.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

```
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
```

4.SyncBailHook

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

```
const slice = Array.prototype.slice;
class SyncBailHook{
    constructor(args) {
        this.args= args;
        this.taps=[];
    }
    tap(name,fn) {
        this.taps.push(fn);
    }
    call() {
        let args = slice.call(arguments,0,this.args.length);
        let result;
        let i=0;
        while(i<this.taps.length&&!result){
            result = this.taps[i++](...args);
        }
    }
}

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);
```

5.SyncWaterfallHook

1. 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
2. 回调函数接受的参数来自于上一个函数的结果
3. 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
4. 当回调函数返回非undefined不会停止回调栈的调用

```
const slice = Array.prototype.slice;
class SyncWaterfallHook{
    constructor(args) {
        this.args= args;
        this.taps=[];
    }
    tap(name,fn) {
        this.taps.push(fn);
    }
    call() {
        let args = slice.call(arguments,0,this.args.length);
        let first=args[0];
        let result;
        let i=0;
        while(i<this.taps.length){
            first = result||first;
            result = this.taps[i++](first,...args.slice(1));
        }
    }
}

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);
```

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

6.SyncLoopHook

- 1. 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

```
class SyncLoopHook {
    constructor(args) {
        this._args = args;
        this.taps = [];
    }
    tap(name, fn) {
        this.taps.push(fn);
    }
    call() {
        let args = Array.from(arguments).slice(0, this._args.length);
        let loop = true;
        while (loop) {
            for (let i = 0; i < this.taps.length; i++) {
                let fn = this.taps[i];
                let result = fn(...args);
                loop = typeof result != 'undefined';
                if (loop) break;
            }
        }
    }
}
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
```

7. AsyncParallelHook

- 异步并行执行钩子**7.1 tap #**
- 同步注册

```
class AsyncParallelHook{
    constructor() {
        this.taps=[];
    }
    tap(name,fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args=Array.from(arguments);
        let callback=args.pop();
        this.taps.forEach(fn => fn(...args));
        callback();
    }
}
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');
});
```

** 7.2 tapAsync #**

- 异步注册，全部任务完成后执行最终的回调

```
class AsyncParallelHook{
    constructor() {
        this.taps=[];
    }
    tapAsync(name,fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args=Array.from(arguments);
        let callback=args.pop();
        let i=0,length = this.taps.length;
        function done(err) {
            if (++i == length) {
                callback(err);
            }
        }
        this.taps.forEach(fn => {
            fn(...args,done);
        });
    }
}
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');
});
```

** 7.3 tapPromise #**

- promise注册钩子
- 全部完成后执行才算成功

```
class AsyncParallelHook{
    constructor() {
        this.taps=[];
    }
    tapPromise(name,fn) {
        this.taps.push(fn);
    }
    promise() {
        let promises = this.taps.map(fn => fn());
        return Promise.all(promises);
    }
}
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');
})
```

8. 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

** 8.1 tap #**

- 用tap注册
- 如果有一个任务有返回值则调用最终的回调

```
class AsyncParallelBailHook{
    constructor() {
        this.taps=[];
    }
    tap(name,fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args=Array.from(arguments);
        let callback=args.pop();
        for (let i=0;i<this.taps.length;i++){
            let ret=this.taps[i](...args);
            if (ret) {
                return callback(ret);
            }
        }
    }
}
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');
});
```

** 8.2 tapAsync #**

- 异步注册
- 有一个任务返回错误就直接调最终的回调

```
class AsyncParallelBailHook{
    constructor() {
        this.taps=[];
    }
    tapAsync(name,fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args=Array.from(arguments);
        let finalCallback=args.pop();
        let count=0,total=this.taps.length;
        function done(err) {
            if (err) {
                return finalCallback(err);
            } else {
                if (++count == total) {
                    return finalCallback();
                }
            }
        }
        for (let i=0;ilet fn=this.taps[i];
            fn(...args,done);
        }
    }
}
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');
});
```

** 8.3 tapPromise #**

- 只要有一个任务有resolve或者reject值，不管成功失败都结束

```
class AsyncParallelBailHook {
    constructor() {
        this.taps = [];
    }
    tapPromise(name, fn) {
        this.taps.push(fn);
    }
    promise() {
        let args = Array.from(arguments);
        let promises = this.taps.map(fn => fn(...arguments));

        return new Promise(function (resolve, reject) {
            promises.forEach(promise => promise.then((data) => {
                if (data) resolve(data);
            }, error => {
                if (error) reject(error);
            }));
        });
    }
}
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');
})
```

9. 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

** 9.1 tap #**

```
let { AsyncSeriesHook } = require('tapable');
class AsyncSeriesHook1 {
    constructor() {
        this.taps = [];
    }
    tap(name, fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args = Array.from(arguments);
        let finalCallback = args.pop();
        for (let i = 0; i < this.taps.length; i++) {
            let fn = this.taps[i];
            fn(...args);
        }
        finalCallback();
    }
}
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');
});
```

** 9.2 tapAsync #**

```
class AsyncSeriesBailHook{
    constructor() {
        this.taps=[];
    }
    tapAsync(name,fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args = Array.from(arguments);
        let finalCallback = args.pop();
        let index = 0, length = this.taps.length;
        let next = () => {
            let fn = this.taps[index++];
            if (fn) {
                fn(...args, next);
            } else {
                finalCallback();
            }
        }
        next();
    }
}
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');
});
```

** 9.3 tapPromise #**

```
class AsyncSeriesHook {
    constructor() {
        this.taps = [];
    }
    tapPromise(name, fn) {
        this.taps.push(fn);
    }
    promise() {
        let args = Array.from(arguments);

        let [first, ...fns] = this.taps;
        return fns.reduce((a, b) => {
           return a.then(() => b(...args));
        }, first(...args));
    }
}
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');
});
```

10. AsyncSeriesBailHook

- 只要有一个返回了不为undefined的值就直接结束

** 10.1 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');
});
```

** 10.2 tabAsync #**

```
class AsyncSeriesBailHook{
    constructor() {
        this.taps=[];
    }
    tapAsync(name,fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args=Array.from(arguments);
        let callback=args.pop();
        let i=0,size = this.taps.length;
        let next=(err) => {
            if (err) return  callback(err);
            let fn=this.taps[i++];
            fn?fn(...args,next):callback();
        }
        next();
    }
}
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');
});
```

** 10.3 tapPromise #**

- 只要有一个promise失败了就整个失败了

```
class AsyncSeriesBailHook{
    constructor() {
        this.taps=[];
    }
    tapPromise(name,fn) {
        this.taps.push(fn);
    }
    promise() {
        let args=Array.from(arguments);
        let [first, ...fns] = this.taps;
        let promise =  fns.reduce((a, b) => {
            return a.then(() => b(),(err)=>Promise.reject(err));
        }, first(...args));
        return promise;
    }
}
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');
});
```

11. AsyncSeriesWaterfallHook

- 和 SeriesWaterfallHook差不多

** 11.1 tap #**

```
class AsyncSeriesWaterfallHook {
    constructor() {
        this.taps = [];
    }
    tap(name, fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args = Array.from(arguments);
        let callback = args.pop();
        let first = args[0];
        let result;
        let i = 0;
        while (i < this.taps.length) {
            first = result || first;
            result = this.taps[i++](first, ...args.slice(1));
        }
        callback();
    }
}
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');
});
```

** 11.2 tapAsync #**

```
class AsyncSeriesWaterfallHook {
    constructor() {
        this.taps = [];
    }
    tapAsync(name, fn) {
        this.taps.push(fn);
    }
    callAsync() {
        let args = Array.from(arguments);
        let callback = args.pop();
        let [first, ...otherArgs] = args;
        let i = 0, size = this.taps.length;
        let next = (err, data) => {
            if (err) return callback(err);
            let fn = this.taps[i++];
            if (fn) {
                if (i == 0) {
                    fn(...args, next);
                } else {
                    fn(data || first, ...otherArgs, next);
                }

            } else {
                callback(err, data);
            }
        }
        next();
    }
}
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');
});
```

** 11.3 tapPromise #**

```
let {AsyncSeriesWaterfallHook} = require('tapable');
class AsyncSeriesWaterfallHook {
    constructor() {
        this.taps = [];
    }
    tapPromise(name, fn) {
        this.taps.push(fn);
    }
    promise(...args) {

        let [first, ...fns] = this.taps;
        return fns.reduce((a, b) => {
            return a.then((data) => b(data));
        }, first(...args));
    }
}
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');
});
```

12.intercept

- 所有钩子都提供额外的拦截器API
- 可以拦截钩子注册，钩子触发，和钩子函数的每次执行

```
class SyncHook {
    constructor(args) {
        this.args = args;
        this.taps = [];
    }
    intercept(options) {
        this.interceptOptions = options;
    }
    tap(name, fn) {
        this.interceptOptions.register && this.interceptOptions.register(
            { type: 'sync', fn, name }
        );
        this.taps.push(fn);
    }
    call() {
        this.interceptOptions.call && this.interceptOptions.call();
        this.taps.forEach(fn => {
            this.interceptOptions.tap && this.interceptOptions.tap();
            fn(...Array.from(arguments));
        });
    }
}
const hook = new SyncHook(["name"]);
hook.intercept({

    call: () => {
        console.log('call');
    },

    tap() {
        console.log('tap');
    },

    register: (tapInfo) => {
        console.log('register', tapInfo);
        return tapInfo;
    }
})
hook.tap("1", (name) => {
    console.log(1, name);
});
hook.tap("2", (name) => {
    console.log(2, name);
});

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

```
register { type: 'sync', fn: [Function], name: '1' }
register { type: 'sync', fn: [Function], name: '2' }
call
tap
1 zhufeng
tap
2 zhufeng
```

13. Context(上下文)

- 可以指定循环时候的上下文，循环的上下文在多次循环之间保持不变

```
const {SyncLoopHook} = require("tapable");
const hook = new SyncLoopHook(["name"]);
let counter=0;

hook.tap({context: true,name:"1"}, (context,name) => {
  context[counter] = counter;
  console.log(1, context,name);
  if(++counter >= 2){
    return;
  }
  return true;
});

hook.intercept({
    context: true,
    loop(context){
       console.log('loop',context);
    }
})
hook.call('zhufeng');
```

14. hook原理

** 14.1 index.js#**

index.js

```
const SyncHook = require("./SyncHook");

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

** 14.2 SyncHook.js #**

SyncHook.js

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

** 14.3 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(type) {
    return this.compile({
      taps: this.taps,
      args: this._args
    });
  }
}

module.exports = Hook;
```

** 14.4 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(options) {
    return new Function(this.args(), this.header() + this.content());
  }
}
module.exports = HookCodeFactory;
```

15.参考

- webpack-internal-plugin-relation (https://github.com/alienzhou/webpack-internal-plugin-relation)