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=193 sentences=957, words=6251

1. webpack 的插件机制

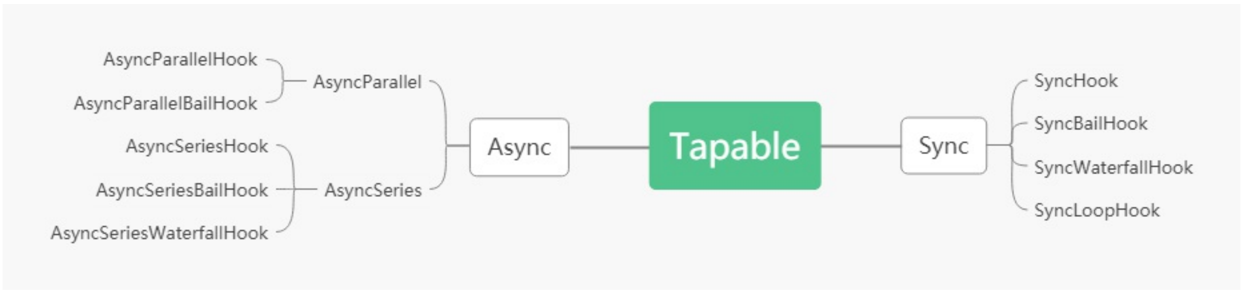
- 在具体介绍 webpack 内置插件与钩子可视化工具之前，我们先来了解一下 webpack 中的插件机制。 webpack 实现插件机制的大体方式是：
  - 创建 - webpack 在其内部对象上创建各种钩子；
  - 注册 - 插件将自己的方法注册到对应钩子上，交给 webpack;
  - 调用 - webpack 编译过程中，会适时地触发相应钩子，因此也就触发了插件的方法。
- Webpack 本质上是一种事件流的机制，它的工作流程就是将各个插件串联起来，而实现这一切的核心就是 Tapable，webpack 中最核心的负责编译的 Compiler 和负责创建 bundle 的 Compilation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

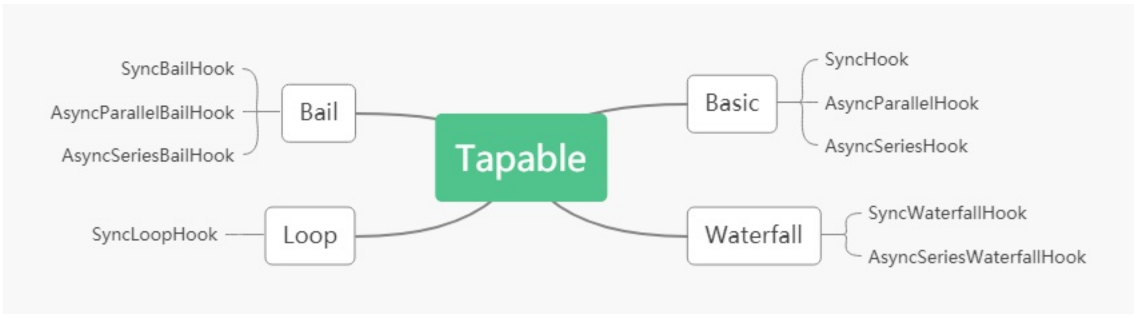
2. tapable 分类

2.1 按同步异步分类

- Hook 类型可以分为 `&#x540C;&#x6B65;Sync和 &#x5F02;&#x6B65;Async，异步又分为 &#x5E76;&#x884C;和 &#x4E32;&#x884C;`

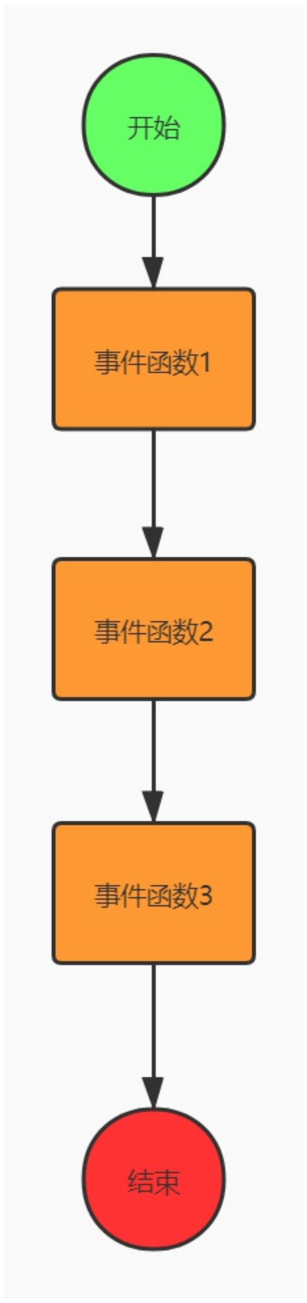


2.1 按返回值分类



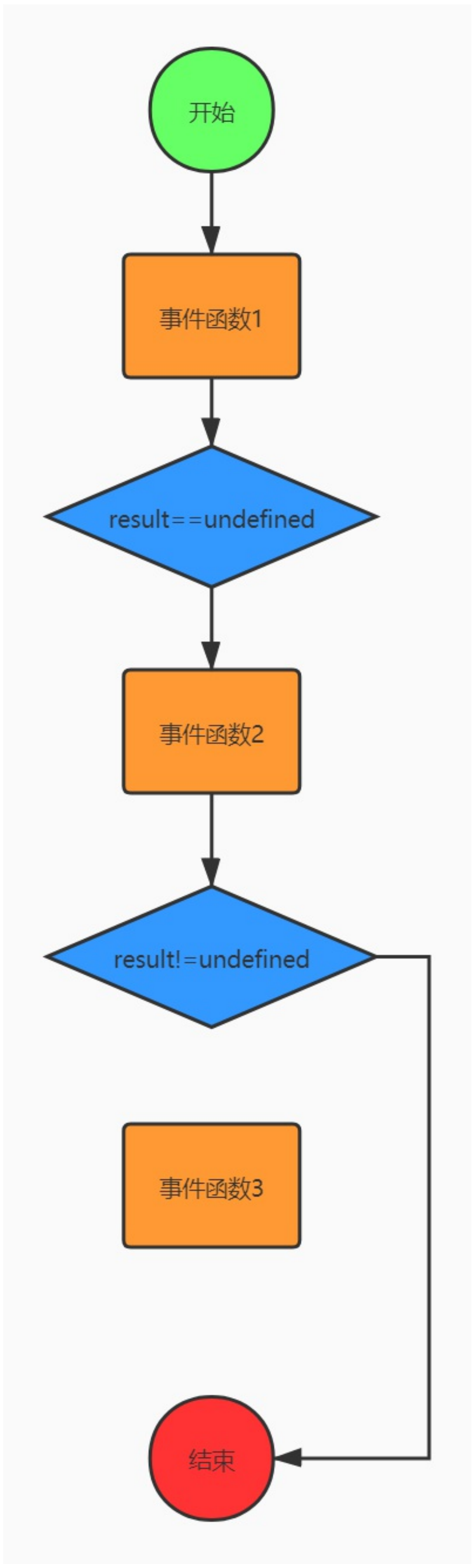
2.1.1 Basic

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



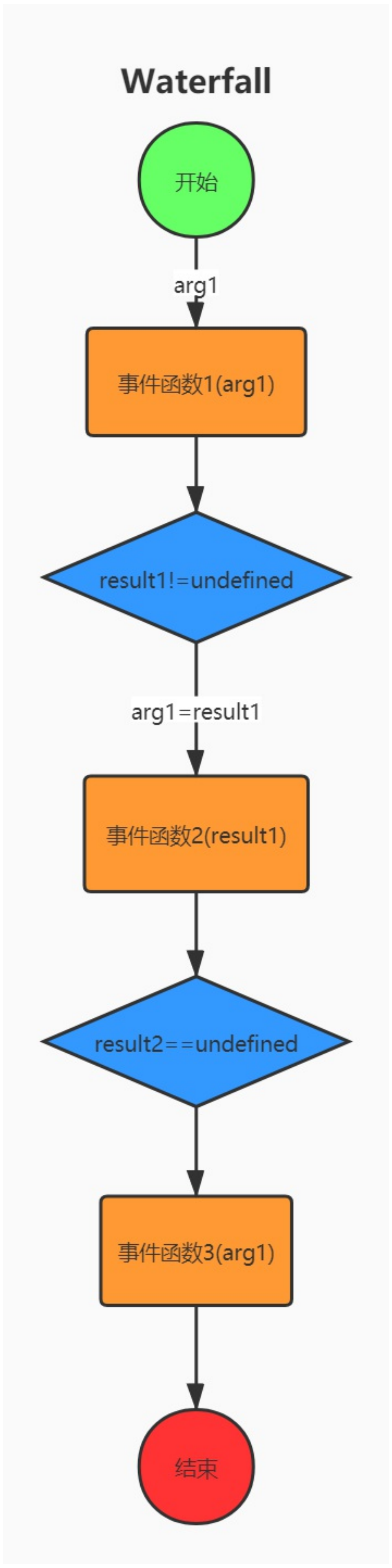
2.1.2 Bail #

- 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



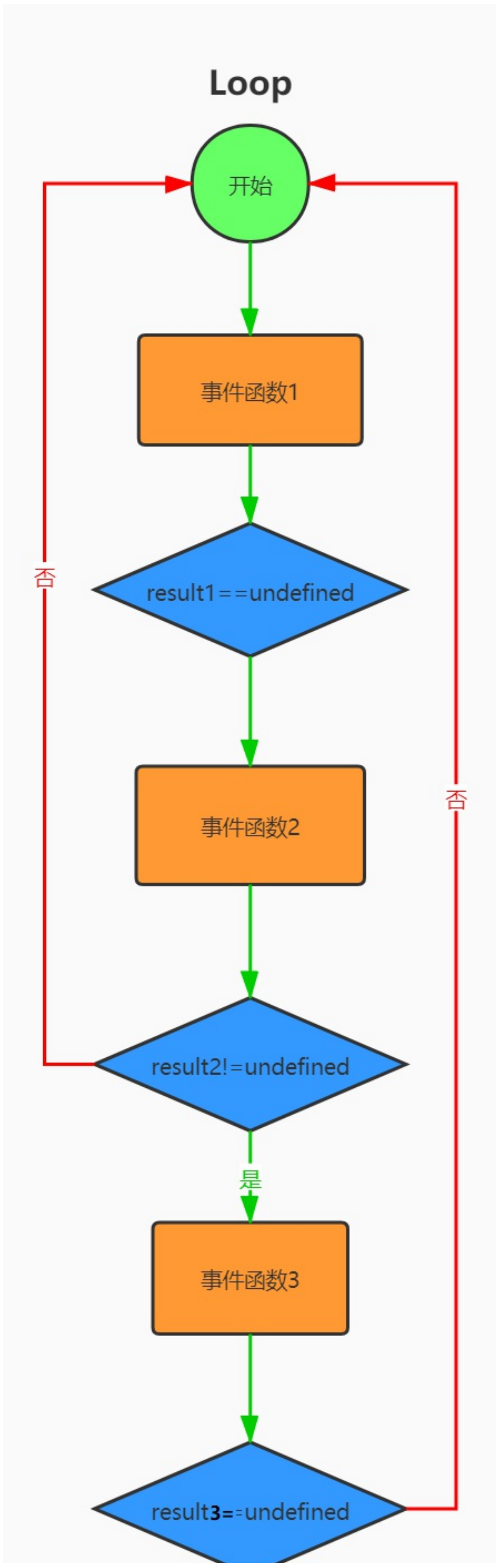
2.1.3 Waterfall [#](#)

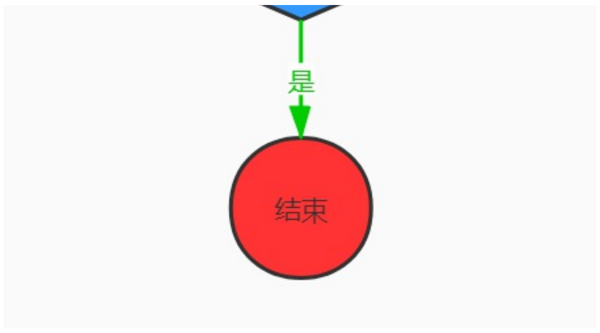
- 如果前一个事件函数的结果 `result !== undefined`,则 result 会作为后一个事件函数的第一个参数,有 SyncWaterfallHook，AsyncSeriesWaterfallHook


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("zhufeng", (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("zhufeng", (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("zhufeng").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
- 对于promise来说，resolve还reject并没有区别
  - 区别在于你是否传给它们的参数

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("zhufeng", (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("zhufeng", (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("zhufeng").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("zhufeng", (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("zhufeng").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("zhufeng", (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("zhufeng", (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("zhufeng").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("zhufeng", 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("zhufeng", 10, (err, data) => {
  console.log(err, data);
  console.timeEnd("cost");
});
```

3.9.1 tapPromise

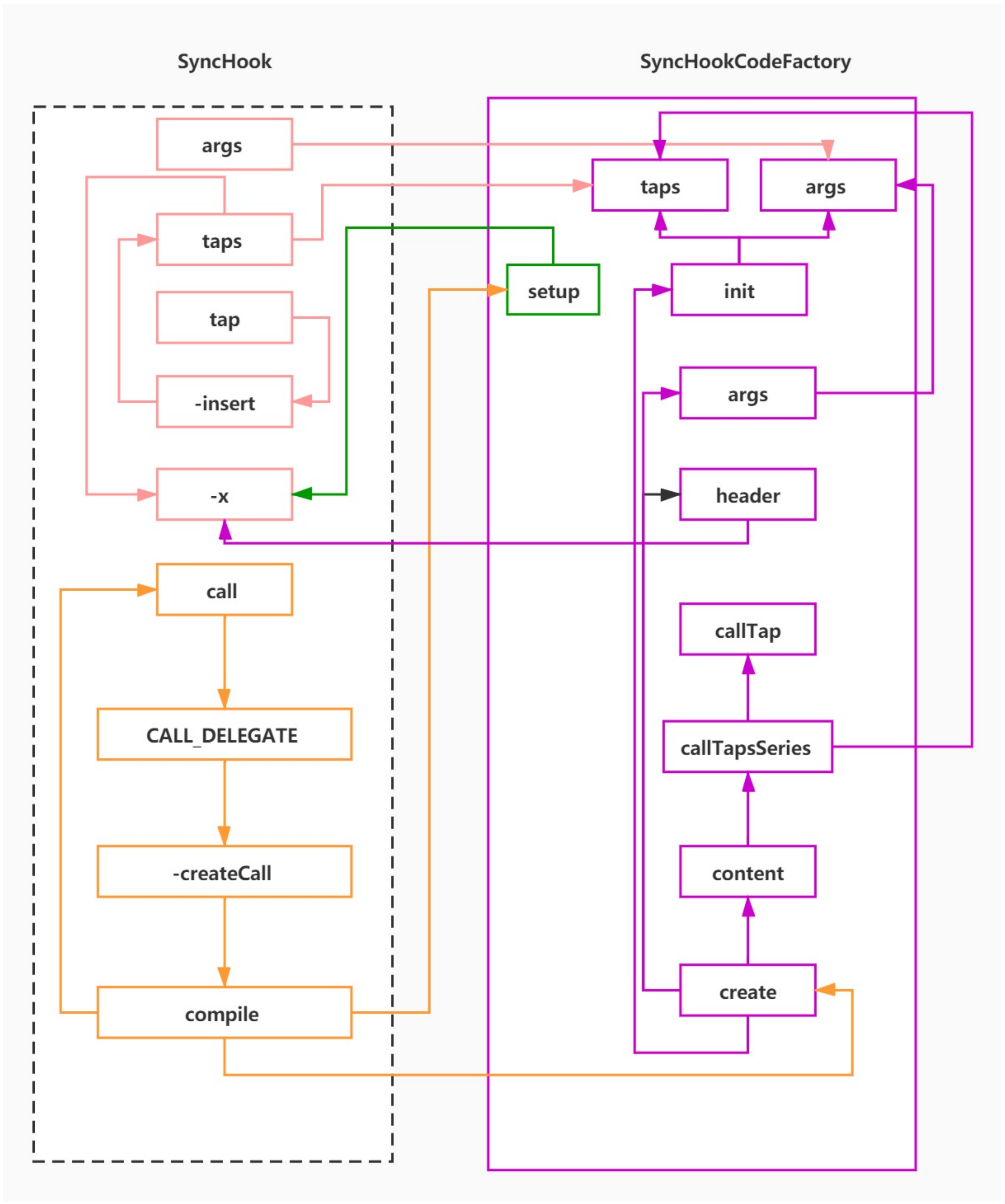
```
let { AsyncSeriesWaterfallHook } = require("tapable");
let queue = new AsyncSeriesWaterfallHook(["name"]);
console.time("cost");
queue.tapPromise("1", function (name) {
  return new Promise(function (resolve) {
    setTimeout(function () {
      console.log(name, 1);
      resolve(1);
    }, 1000);
  });
});
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("zhufeng").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"]);
let fn1 = (name, age) => {
    console.log(1, name, age);
}
syncHook.tap({name:'1'},fn1 );
let fn2 =  (name, age) => {
    console.log(2, name, age);
}
syncHook.tap("2",fn2);
syncHook.call("zhufeng", 10);
```



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.call = CALL_DELEGATE;
    }
    tap(options,fn){
        this._tap("sync", options, fn);
    }
    _tap(type, options, fn) {
        if(typeof options === 'string')
            options={name:options};
        let tapInfo ={...options,type,fn};
        this._insert(tapInfo);
    }
    _resetCompilation(){
        this.call = CALL_DELEGATE;
    }
    _insert(tapInfo){
        this._resetCompilation();
        this.taps.push(tapInfo);
    }
    compile(options) {
        throw new Error("Abstract: should be overridden");
    }
    _createCall(type){
        return this.compile({
            taps:this.taps,
            args:this.args,
            type
        });
    }
}
const CALL_DELEGATE = function(...args) {
    this.call = this._createCall("sync");
    return this.call(...args);
};
module.exports = Hook;
```

4.2.3 SyncHook.js #

tapable\SyncHook.js

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

4.2.4 HookCodeFactory.js #

HookCodeFactory.js

```
class HookCodeFactory {
    setup(hookInstance, options) {
        hookInstance._x = options.taps.map(item => item.fn);
    }
    init(options) {
        this.options = options;
    }
    deinit() {
        this.options = null;
    }
    args(options = {}) {
        let { before, after } = options;
        let allArgs = this.options.args || [];
        if (before) allArgs = [before, ...allArgs];
        if (after) allArgs = [...allArgs, after];
        if (allArgs.length > 0)
            return allArgs.join(', ');
        return "";
    }
    header() {
        let code = "";
        code += "var _x = this._x;\n";
        return code;
    }
    create(options) {
        this.init(options);
        let fn;
        switch (this.options.type) {
            case 'sync':
                fn = new Function(
                    this.args(),
                    this.header() + this.content()
                )
                break;
            default:
                break;
        }
        this.deinit();
        return fn;
    }
    callTapsSeries() {
        if (this.options.taps.length === 0) {
            return '';
        }
        let code = "";
        for (let j =0;j< this.options.taps.length ; j++) {
            const content = this.callTap(j);
            code += content;
        }
        return code;
    }
    callTap(tapIndex) {
        let code = "";
        code += `var _fn${tapIndex} = _x[${tapIndex}];\n`
        let tap = this.options.taps[tapIndex];
        switch (tap.type) {
            case 'sync':
                code += `_fn${tapIndex}(${this.args()});\n`;
                break;
            default:
                break;
        }
        return code;
    }
}
module.exports = HookCodeFactory;
```

5.AsyncParallelHook.callAsync

5.1 使用

```
const { AsyncParallelHook } = require('tapable');
const hook = new AsyncParallelHook(['name', 'age']);
console.time('cost');

hook.tapAsync('1', (name, age, callback) => {
    setTimeout(() => {
        console.log(1, name, age);
        callback();
    }, 1000);
});
hook.tapAsync('2', (name, age,callback) => {
    setTimeout(() => {
        console.log(2, name, age);
        callback();
    }, 2000);
});
hook.tapAsync('3', (name, age,callback) => {
    setTimeout(() => {
        console.log(3, name, age);
        callback();
    }, 3000);
});
debugger
hook.callAsync('zhufeng', 10, (err) => {
    console.log(err);
    console.timeEnd('cost');
});
```

5.2 实现

5.2.1 index.js

tapable\index.js

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

5.2.2 AsyncParallelHook.js

tapable\AsyncParallelHook.js

```
let Hook = require('./Hook');
const HookCodeFactory = require('./HookCodeFactory');
class AsyncParallelHookCodeFactory extends HookCodeFactory{
    content(){
        return this.callTapsParallel()
    }
}
let factory = new AsyncParallelHookCodeFactory();
class AsyncParallelHook extends Hook{
    compile(options) {
        factory.setup(this,options);
        return factory.create(options);
    }
}
module.exports = AsyncParallelHook;
```

5.2.3 Hook.js

tapable\Hook.js

```
class Hook{
    constructor(args){
     if(!Array.isArray(args)) args=[];
     this.args = args;
     this.taps = [];
     this.call = CALL_DELEGATE;
+     this.callAsync = CALL_ASYNC_DELEGATE;
    }
    tap(options,fn){
        this._tap("sync", options, fn);
    }
+    tapAsync(options,fn){
+        this._tap("async", options, fn);
+    }
    _tap(type, options, fn) {
        if(typeof options
            options={name:options};
        let tapInfo ={...options,type,fn};
        this._insert(tapInfo);
    }
    _resetCompilation(){
        this.call = CALL_DELEGATE;
    }
    _insert(tapInfo){
        this._resetCompilation();
        this.taps.push(tapInfo);
    }
    compile(options) {
        throw new Error("Abstract: should be overridden");
    }
    _createCall(type){
        return this.compile({
            taps:this.taps,
            args:this.args,
            type
        });
    }
}
const CALL_DELEGATE = function(...args) {
    this.call = this._createCall("sync");
    return this.call(...args);
};
+const CALL_ASYNC_DELEGATE = function(...args) {
+    this.callAsync = this._createCall("async");
+    return this.callAsync(...args);
+};
module.exports = Hook;
```

5.2.4 HookCodeFactory.js

tapable\HookCodeFactory.js

```
class HookCodeFactory {
    setup(hookInstance, options) {
        hookInstance._x = options.taps.map(item => item.fn);
    }
    init(options) {
        this.options = options;
    }
    deinit() {
        this.options = null;
    }
    args(options = {}) {
        let { before, after } = options;
        let allArgs = this.options.args || [];
        if (before) allArgs = [before, ...allArgs];
        if (after) allArgs = [...allArgs, after];
        if (allArgs.length > 0)
            return allArgs.join(', ');
        return "";
    }
    header() {
        let code = "";
        code += "var _x = this._x;\n";
        return code;
    }
    create(options) {
        this.init(options);
        let fn;
        switch (this.options.type) {
            case 'sync':
                fn = new Function(
                    this.args(),
                    this.header() + this.content()
                )
                break;
+            case 'async':
+                fn = new Function(
+                    this.args({after:'_callback'}),
+                    this.header()+this.content()
+                )
+                break;
            default:
                break;
        }
        this.deinit();
        return fn;
    }
+    callTapsParallel(){
+        let code = `var _counter = ${this.options.taps.length};\n`;
+        code+=`
+            var _done = function () {
+                _callback();
+            };
+        `;
+        for (let j =0;j< this.options.taps.length ; j++) {
+            const content = this.callTap(j);
+            code += content;
+        }
+        return code;
+    }
    callTapsSeries() {
        if (this.options.taps.length
            return '';
        }
        let code = "";
        for (let j =0;j< this.options.taps.length ; j++) {
            const content = this.callTap(j);
            code += content;
        }
        return code;
    }
    callTap(tapIndex) {
        let code = "";
        code += `var _fn${tapIndex} = _x[${tapIndex}];\n`
        let tap = this.options.taps[tapIndex];
        switch (tap.type) {
            case 'sync':
                code += `_fn${tapIndex}(${this.args()});\n`;
                break;
+            case 'async':
+                code += `
+                    _fn${tapIndex}(${this.args({after:`function (_err${tapIndex}) {
+                        if (--_counter === 0) _done();
+                    }`})});
+                `;
+                break;
            default:
                break;
        }
        return code;
    }
}
module.exports = HookCodeFactory;
```

6.AsyncParallelHook.callPromise

6.1 使用

```
let { AsyncParallelHook } = require("./tapable2");
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.log.timeEnd("cost");
  },
  (error) => {
    console.log(error);
    console.timeEnd("cost");
  }
);
```

6.2 实现

6.2.1 Hook.js

tapable\Hook.js

```
class Hook{
    constructor(args){
     if(!Array.isArray(args)) args=[];
     this.args = args;
     this.taps = [];
     this.call = CALL_DELEGATE;
     this.callAsync = CALL_ASYNC_DELEGATE;
+     this.promise = PROMISE_DELEGATE;
    }
    tap(options,fn){
        this._tap("sync", options, fn);
    }
    tapAsync(options,fn){
        this._tap("async", options, fn);
    }
+    tapPromise(options,fn){
+        this._tap("promise", options, fn);
+    }
    _tap(type, options, fn) {
        if(typeof options
            options={name:options};
        let tapInfo ={...options,type,fn};
        this._insert(tapInfo);
    }
    _resetCompilation(){
        this.call = CALL_DELEGATE;
    }
    _insert(tapInfo){
        this._resetCompilation();
        this.taps.push(tapInfo);
    }
    compile(options) {
        throw new Error("Abstract: should be overridden");
    }
    _createCall(type){
        return this.compile({
            taps:this.taps,
            args:this.args,
            type
        });
    }
}
const CALL_DELEGATE = function(...args) {
    this.call = this._createCall("sync");
    return this.call(...args);
};
const CALL_ASYNC_DELEGATE = function(...args) {
    this.callAsync = this._createCall("async");
    return this.callAsync(...args);
};
+const PROMISE_DELEGATE = function(...args) {
+    this.promise = this._createCall("promise");
+    return this.promise(...args);
+};
module.exports = Hook;
```

6.2.2 AsyncParallelHook.js

tapable\AsyncParallelHook.js

```
let Hook = require('./Hook');
const HookCodeFactory = require('./HookCodeFactory');
class AsyncParallelHookCodeFactory extends HookCodeFactory{
+    content({onDone}){
+        return this.callTapsParallel({onDone})
    }
}
let factory = new AsyncParallelHookCodeFactory();
class AsyncParallelHook extends Hook{
    compile(options) {
        factory.setup(this,options);
        return factory.create(options);
    }
}
module.exports = AsyncParallelHook;
```

6.2.3 HookCodeFactory.js #

tapable\HookCodeFactory.js

```
class HookCodeFactory {
    setup(hookInstance, options) {
        hookInstance._x = options.taps.map(item => item.fn);
    }
    init(options) {
        this.options = options;
    }
    deinit() {
        this.options = null;
    }
    args(options = {}) {
        let { before, after } = options;
        let allArgs = this.options.args || [];
        if (before) allArgs = [before, ...allArgs];
        if (after) allArgs = [...allArgs, after];
        if (allArgs.length > 0)
            return allArgs.join(', ');
        return "";
    }
    header() {
        let code = "";
        code += "var _x = this._x;\n";
        return code;
    }
    create(options) {
        this.init(options);
        let fn;
        switch (this.options.type) {
            case 'sync':
                fn = new Function(
                    this.args(),
                    this.header() + this.content()
                )
                break;
            case 'async':
                fn = new Function(
                    this.args({after:'_callback'}),
+                    this.header()+this.content({ onDone:()=>" _callback();\n"})
                )
                break;
+            case 'promise':
+                let tapsContent = this.content({ onDone:()=>" _resolve();\n"});
+                let content = `return new Promise(function (_resolve, _reject) {
+                    ${tapsContent}
+                })`;
+                fn = new Function(
+                    this.args(),
+                    this.header()+content
+                )
+                break;
            default:
                break;
        }
        this.deinit();
        return fn;
    }
+    callTapsParallel({onDone}){
        let code = `var _counter = ${this.options.taps.length};\n`;
        code+=`
            var _done = function () {
+                ${onDone()}
            };
        `;
        for (let j =0;j< this.options.taps.length ; j++) {
            const content = this.callTap(j);
            code += content;
        }
        return code;
    }
    callTapsSeries() {
        if (this.options.taps.length
            return '';
        }
        let code = "";
        for (let j =0;j< this.options.taps.length ; j++) {
            const content = this.callTap(j);
            code += content;
        }
        return code;
    }
    callTap(tapIndex) {
        let code = "";
        code += `var _fn${tapIndex} = _x[${tapIndex}];\n`
        let tap = this.options.taps[tapIndex];
        switch (tap.type) {
            case 'sync':
                code += `_fn${tapIndex}(${this.args()});\n`;
                break;
            case 'async':
                code += `
                    _fn${tapIndex}(${this.args({after:`function (_err${tapIndex}) {
                        if (--_counter
                    }`})});
                `;
                break;
+           case 'promise':
+              code = `
+                  var _fn${tapIndex} = _x[${tapIndex}];
+                  var _promise${tapIndex} = _fn${tapIndex}(${this.args()});
+                  _promise${tapIndex}.then(
+                      function () {
+                       if (--_counter === 0) _done();
+                      }
+                  );
+              `;
            default:
                break;
        }
        return code;
    }
}
module.exports = HookCodeFactory;
```

7. interceptor

- 所有钩子都提供额外的拦截器API
    - call:(...args) => void当你的钩子触发之前,(就是call()之前),就会触发这个函数,你可以访问钩子的参数.多个钩子执行一次
    - tap: (tap: Tap) => void 每个钩子执行之前(多个钩子执行多个),就会触发这个函数
    - register:(tap: Tap) => Tap | undefined 每添加一个Tap都会触发 你interceptor上的register,你下一个拦截器的register 函数得到的参数 取决于你上一个register返回的值,所以你最好返回一个 tap 钩子.
- Context(上下文) 插件和拦截器都可以选择加入一个可选的 context对象, 这个可以被用于传递随意的值到队列中的插件和拦截器

7.1 使用

```
const {SyncHook} = require('tapable');
const syncHook = new SyncHook(["name","age"]);
syncHook.intercept({
    register:(tapInfo)=>{
        console.log(`拦截器1开始register`);
        return tapInfo;
    },
    tap:(tapInfo)=>{
        console.log(`拦截器1开始tap`);
    },
    call:(name,age)=>{
        console.log(`拦截器1开始call`,name,age);
    }
});
syncHook.intercept({
    register:(tapInfo)=>{
        console.log(`拦截器2开始register`);
        return tapInfo;
    },
    tap:(tapInfo)=>{
        console.log(`拦截器2开始tap`);
    },
    call:(name,age)=>{
        console.log(`拦截器2开始call`,name,age);
    }
});

syncHook.tap({name:'回调函数A'},(name,age)=>{
   console.log(`回调A`,name,age);
});

syncHook.tap({name:'回调函数B'},(name,age)=>{
    console.log('回调B',name,age);
});
debugger
syncHook.call('zhufeng',10);
```

```
(function anonymous(name, age) {
  var _x = this._x;
  var _taps = this.taps;

  var _interceptors = this.interceptors;
  _interceptors[0].call(name, age);
  _interceptors[1].call(name, age);

  var _tap0 = _taps[0];
  _interceptors[0].tap(_tap0);
  _interceptors[1].tap(_tap0);
  var _fn0 = _x[0];
  _fn0(name, age);

  var _tap1 = _taps[1];
  _interceptors[0].tap(_tap1);
  _interceptors[1].tap(_tap1);
  var _fn1 = _x[1];
  _fn1(name, age);
});
```

7.2 实现

7.2.1 Hook.js #

tapable\Hook.js

```
class Hook{
    constructor(args){
     if(!Array.isArray(args)) args=[];
     this.args = args;
     this.taps = [];
     this.call = CALL_DELEGATE;
     this.callAsync = CALL_ASYNC_DELEGATE;
     this.promise = PROMISE_DELEGATE;
+     this.interceptors = [];
    }
    tap(options,fn){
        this._tap("sync", options, fn);
    }
    tapAsync(options,fn){
        this._tap("async", options, fn);
    }
    tapPromise(options,fn){
        this._tap("promise", options, fn);
    }
    _tap(type, options, fn) {
        if(typeof options
            options={name:options};
        let tapInfo ={...options,type,fn};
+        tapInfo=this._runRegisterInterceptors(tapInfo);
        this._insert(tapInfo);
    }
+    _runRegisterInterceptors(tapInfo){
+        for(const interceptor of this.interceptors){
+            if(interceptor.register){
+             let newTapInfo = interceptor.register(tapInfo);
+             if(newTapInfo){
+                 tapInfo=newTapInfo;
+             }
+            }
+        }
+        return tapInfo;
+     }
+     intercept(interceptor){
+         this.interceptors.push(interceptor);
+     }
    _resetCompilation(){
        this.call = CALL_DELEGATE;
    }
    _insert(tapInfo){
        this._resetCompilation();
        this.taps.push(tapInfo);
    }
    compile(options) {
        throw new Error("Abstract: should be overridden");
    }
    _createCall(type){
        return this.compile({
            taps:this.taps,
            args:this.args,
+            interceptors:this.interceptors,
            type
        });
    }
}
const CALL_DELEGATE = function(...args) {
    this.call = this._createCall("sync");
    return this.call(...args);
};
const CALL_ASYNC_DELEGATE = function(...args) {
    this.callAsync = this._createCall("async");
    return this.callAsync(...args);
};
const PROMISE_DELEGATE = function(...args) {
    this.promise = this._createCall("promise");
    return this.promise(...args);
};
module.exports = Hook;
```

7.2.2 HookCodeFactory.js #

tapable\HookCodeFactory.js

```

class HookCodeFactory {
    setup(hookInstance, options) {
        hookInstance._x = options.taps.map(item => item.fn);
    }
    init(options) {
        this.options = options;
    }
    deinit() {
        this.options = null;
    }
    args(options = {}) {
        let { before, after } = options;
        let allArgs = this.options.args || [];
        if (before) allArgs = [before, ...allArgs];
        if (after) allArgs = [...allArgs, after];
        if (allArgs.length > 0)
            return allArgs.join(', ');
        return "";
    }
    header() {
        let code = "";
        code += "var _x = this._x;\n";
```

```
+        if(this.options.interceptors.length>0){
+            code += `var _taps = this.taps;\n`;
+            code += `var _interceptors = this.interceptors;\n`;
+        }
+        for(let k=0;k
+            const interceptor=this.options.interceptors[k];
+            if(interceptor.call)
+                code += `_interceptors[${k}].call(${this.args()});\n`;
+        }
        return code;
    }
    create(options) {
        this.init(options);
        let fn;
        switch (this.options.type) {
            case 'sync':
                fn = new Function(
                    this.args(),
                    this.header() + this.content()
                )
                break;
            case 'async':
                fn = new Function(
                    this.args({after:'_callback'}),
                    this.header()+this.content({ onDone:()=>" _callback();\n"})
                )
                break;
            case 'promise':
                 let tapsContent = this.content({ onDone:()=>" _resolve();\n"});
                 let content = `return new Promise(function (_resolve, _reject) {
                     ${tapsContent}
                 })`;
                 fn = new Function(
                     this.args(),
                     this.header()+content
                 )
                break;
            default:
                break;
        }
        this.deinit();
        return fn;
    }
    callTapsParallel({onDone}){
        let code = `var _counter = ${this.options.taps.length};\n`;
        code+=`
            var _done = function () {
                ${onDone()}
            };
        `;
        for (let j =0;j< this.options.taps.length ; j++) {
            const content = this.callTap(j);
            code += content;
        }
        return code;
    }
    callTapsSeries() {
        if (this.options.taps.length
            return '';
        }
        let code = "";
        for (let j =0;j< this.options.taps.length ; j++) {
            const content = this.callTap(j);
            code += content;
        }
        return code;
    }
    callTap(tapIndex) {
        let code = "";
+        if(this.options.interceptors.length>0){
+         code += `var _tap${tapIndex} = _taps[${tapIndex}];`;
+         for(let i=0;i
+            let interceptor = this.options.interceptors[i];
+            if(interceptor.tap){
+                code += `_interceptors[${i}].tap(_tap${tapIndex});`;
+            }
+         }
+        }

        code += `var _fn${tapIndex} = _x[${tapIndex}];\n`
        let tap = this.options.taps[tapIndex];
        switch (tap.type) {
            case 'sync':
                code += `_fn${tapIndex}(${this.args()});\n`;
                break;
            case 'async':
                code += `
                    _fn${tapIndex}(${this.args({after:`function (_err${tapIndex}) {
                        if (--_counter
                    }`})});
                `;
                break;
            case 'promise':
               code = `
                   var _fn${tapIndex} = _x[${tapIndex}];
                   var _promise${tapIndex} = _fn${tapIndex}(${this.args()});
                   _promise${tapIndex}.then(
                       function () {
                        if (--_counter
                       }
                   );
               `;
            default:
                break;
        }
        return code;
    }
}
module.exports = HookCodeFactory;
```

8. HookMap

- A HookMap is a helper class for a Map with Hooks

8.1 HookMap

```
let {SyncHook,HookMap} = require('./tapable');
const keyedHookMap = new HookMap(()=>new SyncHook(["name"]));
keyedHookMap.for('key1').tap('plugin1',(name)=>{console.log(1,name);});
keyedHookMap.for('key1').tap('plugin2',(name)=>{console.log(2,name);});
const hook1 = keyedHookMap.get('key1');
hook1.call('zhufeng');
```

8.2 tapable\index.js

tapable\index.js

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

8.3 HookMap

```
class HookMap {
  constructor(factory) {
    this._map = new Map();
    this._factory = factory;
  }
  get(key) {
    return this._map.get(key);
  }
  tapAsync(key, options, fn) {
    return this.for(key).tapAsync(options, fn);
  }
  tapPromise(key, options, fn) {
    return this.for(key).tapPromise(options, fn);
  }
  for(key) {
    const hook = this.get(key);
    if (hook) return hook;
    let newHook = this._factory();
    this._map.set(key, newHook);
    return newHook;
  }
}
module.exports = HookMap;
```

9. stage

9.1 stage

```
let {SyncHook} = require('tapable');
let hook = new SyncHook(['name']);
debugger
hook.tap({name:'tap1',stage:1},(name)=>{
   console.log(1,name);
});
hook.tap({name:'tap3',stage:3},(name)=>{
   console.log(3,name);
});
hook.tap({name:'tap5',stage:5},(name)=>{
   console.log(4,name);
});
hook.tap({name:'tap2',stage:2},(name)=>{
   console.log(2,name);
});

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

9.2 tapable\Hook.js

tapable\Hook.js

```
class Hook{
    constructor(args){
     if(!Array.isArray(args)) args=[];
     this.args = args;
     this.taps = [];
     this.call = CALL_DELEGATE;
     this.callAsync = CALL_ASYNC_DELEGATE;
     this.promise = PROMISE_DELEGATE;
     this.interceptors = [];
    }
    tap(options,fn){
        this._tap("sync", options, fn);
    }
    tapAsync(options,fn){
        this._tap("async", options, fn);
    }
    tapPromise(options,fn){
        this._tap("promise", options, fn);
    }
    _tap(type, options, fn) {
        if(typeof options
            options={name:options};
        let tapInfo ={...options,type,fn};
        tapInfo=this._runRegisterInterceptors(tapInfo);
        this._insert(tapInfo);
    }
    _runRegisterInterceptors(tapInfo){
        for(const interceptor of this.interceptors){
            if(interceptor.register){
             let newTapInfo = interceptor.register(tapInfo);
             if(newTapInfo){
                 tapInfo=newTapInfo;
             }
            }
        }
        return tapInfo;
     }
     intercept(interceptor){
         this.interceptors.push(interceptor);
     }
    _resetCompilation(){
        this.call = CALL_DELEGATE;
    }
    _insert(tapInfo){
        this._resetCompilation();
+            let stage = 0;
+            if (typeof tapInfo.stage === "number") {
+                stage = tapInfo.stage;
+            }
+            let i = this.taps.length;
+            while (i > 0) {
+                i--;
+                const x = this.taps[i];
+                this.taps[i + 1] = x;
+                const xStage = x.stage || 0;
+                if (xStage > stage) {
+                    continue;
+                }
+                i++;
+                break;
+            }
+            this.taps[i] = tapInfo;
    }
    compile(options) {
        throw new Error("Abstract: should be overridden");
    }
    _createCall(type){
        return this.compile({
            taps:this.taps,
            args:this.args,
            interceptors:this.interceptors,
            type
        });
    }
}
const CALL_DELEGATE = function(...args) {
    this.call = this._createCall("sync");
    return this.call(...args);
};
const CALL_ASYNC_DELEGATE = function(...args) {
    this.callAsync = this._createCall("async");
    return this.callAsync(...args);
};
const PROMISE_DELEGATE = function(...args) {
    this.promise = this._createCall("promise");
    return this.promise(...args);
};
module.exports = Hook;
```

10. before

10.1 before.js

```
let {SyncHook} = require('tapable');
let hook = new SyncHook(['name']);
debugger
hook.tap({name:'tap1'},(name)=>{
   console.log(1,name);
});
hook.tap({name:'tap3'},(name)=>{
   console.log(3,name);
});
hook.tap({name:'tap5'},(name)=>{
   console.log(4,name);
});
hook.tap({name:'tap2',before:['tap3','tap5']},(name)=>{
   console.log(2,name);
});

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

10.2 Hook.js

```
class Hook{
    constructor(args){
     if(!Array.isArray(args)) args=[];
     this.args = args;
     this.taps = [];
     this.call = CALL_DELEGATE;
     this.callAsync = CALL_ASYNC_DELEGATE;
     this.promise = PROMISE_DELEGATE;
     this.interceptors = [];
    }
    tap(options,fn){
        this._tap("sync", options, fn);
    }
    tapAsync(options,fn){
        this._tap("async", options, fn);
    }
    tapPromise(options,fn){
        this._tap("promise", options, fn);
    }
    _tap(type, options, fn) {
        if(typeof options
            options={name:options};
        let tapInfo ={...options,type,fn};
        tapInfo=this._runRegisterInterceptors(tapInfo);
        this._insert(tapInfo);
    }
    _runRegisterInterceptors(tapInfo){
        for(const interceptor of this.interceptors){
            if(interceptor.register){
             let newTapInfo = interceptor.register(tapInfo);
             if(newTapInfo){
                 tapInfo=newTapInfo;
             }
            }
        }
        return tapInfo;
     }
     intercept(interceptor){
         this.interceptors.push(interceptor);
     }
    _resetCompilation(){
        this.call = CALL_DELEGATE;
    }
    _insert(tapInfo){
        this._resetCompilation();
+       let before;
+            if (typeof tapInfo.before === "string") {
+                before = new Set([tapInfo.before]);
+            } else if (Array.isArray(tapInfo.before)) {
+                before = new Set(tapInfo.before);
+            }
        let stage = 0;
        if (typeof tapInfo.stage
            stage = tapInfo.stage;
        }
        let i = this.taps.length;
        while (i > 0) {
            i--;
            const x = this.taps[i];
            this.taps[i + 1] = x;
      const xStage = x.stage || 0;
+     if (before) {
+                if (before.has(x.name)) {
+                    before.delete(x.name);
+                    continue;
+                }
+                if (before.size > 0) {
+                    continue;
+                }
            }
            if (xStage > stage) {
                continue;
            }
            i++;
            break;
        }
        this.taps[i] = tapInfo;
    }
    compile(options) {
        throw new Error("Abstract: should be overridden");
    }
    _createCall(type){
        return this.compile({
            taps:this.taps,
            args:this.args,
            interceptors:this.interceptors,
            type
        });
    }
}
const CALL_DELEGATE = function(...args) {
    this.call = this._createCall("sync");
    return this.call(...args);
};
const CALL_ASYNC_DELEGATE = function(...args) {
    this.callAsync = this._createCall("async");
    return this.callAsync(...args);
};
const PROMISE_DELEGATE = function(...args) {
    this.promise = this._createCall("promise");
    return this.promise(...args);
};
module.exports = Hook;
```