全局对象

全局变量 global

在 Node.js 环境中，也有唯一的全局对象，但不叫 window，而叫 global，这个对象的属性和方法也和浏览器环境的 window 不同。

进程对象 process

这说明传入 process.nextTick()的函数不是立刻执行，而是要等到下一次事件循环。

Node.js 进程本身的事件就由 process 对象来处理。

http 模块

应用程序并不直接和 HTTP 协议打交道，而是操作 `http` 模块提供的 `request` 和 `response` 对象。

`request` 对象封装了 HTTP 请求，我们调用 `request` 对象的属性和方法就可以拿到所有 HTTP 请求的信息；

`response` 对象封装了 HTTP 响应，我们操作 `response` 对象的方法，就可以把 HTTP 响应返回给浏览器。

```
var http = require('http');
// 创建 http server,并传入回调函数
var server = http.createServer(function(req, resp) {
    console.log(req, resp);
    // 得到请求的 method, url
    console.log(req.method + ':' + req.url);
    // 将状态码 200 和响应类型 写入响应头
    resp.writeHead(200, {'Content-Type' : 'text/html'});
```

```
    // 将 HTTP 响应的 HTML 内容写入 response
    resp.end('<h1>Hello World</h1>')
});
server.listen(8080);
console.log('Server is running at http://127.0.0.1.8080');
```

url 模块

通过 `parse()` 将一个字符串解析为一个 `Url` 对象： 列出对象一些基本属性

```
search: '?name=zzj',
query: 'name=zzj',
pathname: '/01_1http.js',
path: '/01_1http.js?name=zzj',
href: '/01_1http.js?name=zzj'
```

path 模块

`path` 模块，它可以方便地构造目录：

```
var path = require('path');
// 解析当前目录的绝对路径
var workDir = path.resolve('.');
console.log(workDir);
// 组合完整路径
var filePath = path.join(workDir, 'pub', 'index.html');
```

Node.js 回调函数

Node.js 异步编程的直接体现就是回调。

异步编程依托于回调来实现，但不能说使用了回调后程序就异步化了。

这样在执行代码时就没有阻塞或等待文件 I/O 操作。这就大大提高了 Node.js 的性能，可以处理大量的并发请求。

Node.js 事件循环

Node.js 基本上所有的事件机制都是用设计模式中观察者模式实现。

Node.js 单线程类似进入一个 while(true)的事件循环，直到没有事件观察者退出，每个异步事件都生成一个事件观察者，如果有事件发生就调用该回调函数.

事件驱动程序

Node.js 使用事件驱动模型，当 web server 接收到请求，就把它关闭然后进行处理，然后去服务下一个 web 请求。

当这个请求完成，它被放回处理队列，当到达队列开头，这个结果被返回给用户。

因为 webserver 一直接受请求而不等待任何读写操作。（这也称之为非阻塞式 IO 或者事件驱动 IO）

Node.js EventEmitter

Node.js 所有的异步 I/O 操作在完成时都会发送一个事件到事件队列。

Node.js 里面的许多对象都会分发事件：一个 net.Server 对象会在每次有新连接时触发一个事件， 一个 fs.readStream 对象会在文件被打开的时候触发一个事件。 所有这些产生事件的对象都是 events.EventEmitter 的实例。

一个事件可有多个监听

Node.js Buffer

JavaScript 语言自身只有字符串数据类型，没有二进制数据类型。

Buffer 类是随 Node 内核一起发布的核心库。Buffer 库为 Node.js 带来了一种存储原始数据的方法，可以让 Node.js 处理二进制数据，每当需要在 Node.js 中处理 I/O 操作中移动的数据时，就有可能使用 Buffer 库。原始数据存储在 Buffer 类的实例中。一个 Buffer 类似于一个整数数组，但它对应于 V8 堆内存之外的一块原始内存。

建议使用 Buffer.from() 接口去创建 Buffer 对象。而不是 new Buffer()构造函数

taf 接口调用服务实例

1. 创建项目文件安装依赖

```
npm install @taf/taf-rpc

npm install nodemon -D
```

2. 编辑 Hello.jce 文件

定义入参

```
struct HelloWorldReq
    {
        0 optional  string              data;           // 入参
    };
```

定义返回参数

```
    struct HelloWorldRsp
    {
        0 optional  int                 iRet;           // 返回码
        1 optional  string              message;        // 返回信息
```

```
        };
```

定义接口并设置传入参数，和 返回值

```
    interface Hello
        {
            int test();

            // 接口调用方法名
            int helloWorld(HelloWorldReq stReq, out HelloWorldRsp stRsp)
;
        };
```

3. 安装@up/oem-cli - oem 通用脚手架，可在本地编译 jce

```
    up jce2node --client   MyTest.jce 生成 xxxProxy.js，开发者引入该文件之后，可以直接调用服务端的服务 RPC：远程过程调用

    up jce2node --server MyTest.jce 生成 xx.jc 和 xxImp.js 完成 xxImp.js 具体函数

    up jce2node MyTest.jce
```

在服务端使用 `up jce2node --server Hello.jce`

在 client 客户端中 使用 `up jce2node Hello.jce --client`

```
PS D:\zijianzhang\whup\study\tafTest\HelloServer> up jce2node Hello.jce --client
success
done!
```

4. 创建编写入口启动文件 server.js，本地配置文件 local.conf，实现 HelloImp.js 中的接口， 编写客户端文件，调用服务代码 client.js

5. 修改 package.json 启动脚本

6. npm start 启动服务

```
PS D:\zijianzhang\whup\study\tafTest\HelloServer> npm start

> HelloServer@1.0.0 start D:\zijianzhang\whup\study\tafTest\HelloServer
> nodemon server.js

[nodemon] 2.0.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
LXSC.HelloServer.HelloObj
```

7. node client.js 允许客户端

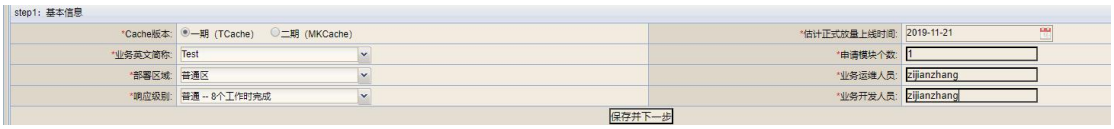
```
PS D:\zijianzhang\whup\study\tafTest\HelloServer> node client.js
### helloWorld ok ### { iRet: 0, message: 'Hello World' }
```

DCache 起步

DCache 安装

1. 进入 taf 管理平台，点击 运维工具-> 缓存部署 -> Cache 安装

2. 找到安装 DCache 和 DB 的业务，申请一个模块，其他默认填写保存下一步



3. 取一个有标识性的模块名称，不热备，接口调用为批量+单次，确定进入下一步

4. 允许淘汰，允许落地 DB，允许 Onlykey,DB 可读，落地方式为 key+value 直接落地

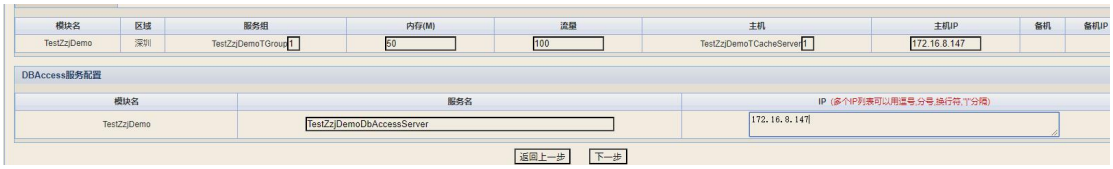
5. 主索引字段类型为 varchar(255),数据的 JCE 类型为 string,DB 字段类型为 mediumblob，可选

6. 点下一步，进入 DB 实例填写，

7. 选择数据库 ip 并填写用户名，密码，端口号默认 3306

8. 下一步->建库-》通过->进入服务组配置->配置一个服务组

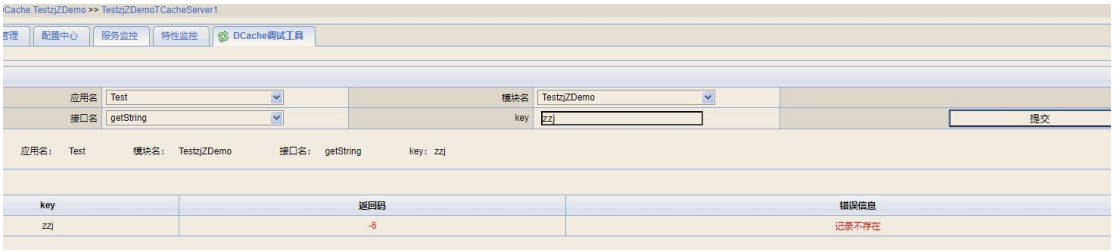
9. 填写服务器主机 ip,内存默认分配 50M



10. 下一步->默认提交->路由校验

11. 安装

12. 在缓存管理里找到对应的 Test 模块的 Cache 和 DBAccess 管理，发布对应的服务，发布后可以调试



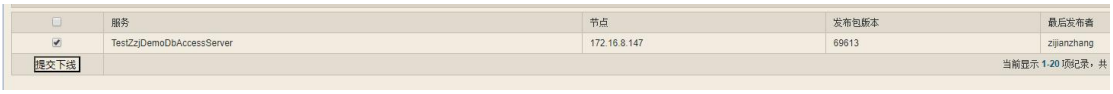
DCache 下线

Cache 下线

运维工具->缓存部署->非 Cache 下线-> 找到对应的服务提交下线

dbAccessServer 下线

运维工具->缓存部署->非 Cache 下线-> 找到对应的服务提交下线



node.js.pdf

高阶函数

高阶函数则是可以把函数作为参数，或是将函数最为返回值的函数

偏函数：

偏函数是固定一个函数的一个或多个参数，然后返回一个新函数

```
// 入参函数
function add(a, b, c) {
    return a + b + c
}
// 生产偏函数
function partial(fn, a) {
    return function(b, c) {
        return fn(a, b, c)
    }
}
// 传入 入参函数和固定参数，并接收返回的新函数
const parAdd = partial(add, 66);
console.log(parAdd(1, 2)); //利用返回的新函数 再传入剩余的参数
```