优化的3个实际案例

机票前端开发

他们叫我**狼叔**

- 姓名：桑世龙
- 部门：罗辑思维&得到 首席前端架构师
- 简要介绍：嗷呜。。。

Node.js布道　StuQ明星讲师　被坑的CTO　晒娃狂魔

目录

一、用户订单token如何通过命令行获取

以前的鉴权代码

```
function mockToken() {
    return Promise.resolve({
        "ret": true,
        "data": {
            "mobile": 135[redacted]52,
            "orderno": "cta170[redacted]127448",
            "ordertoken": "a71dc1a9[redacted]2db68bdfa90ae1895d",
            "prenum": 86,
            "querytype": 3
        }
    });
}

// 验证非登陆状态用，提交时删除
function gettoken(){
    return usercenter.queryOrderToken('cta170209[redacted]448', '13521[redacted]2', '86')
        .then(function(data){
            resolve(data)
        }).catch(function (err) {
            console.log(err)
        })
}
```

写死，蛋疼

获取beta线上的，蛋疼

我们要的只是token...

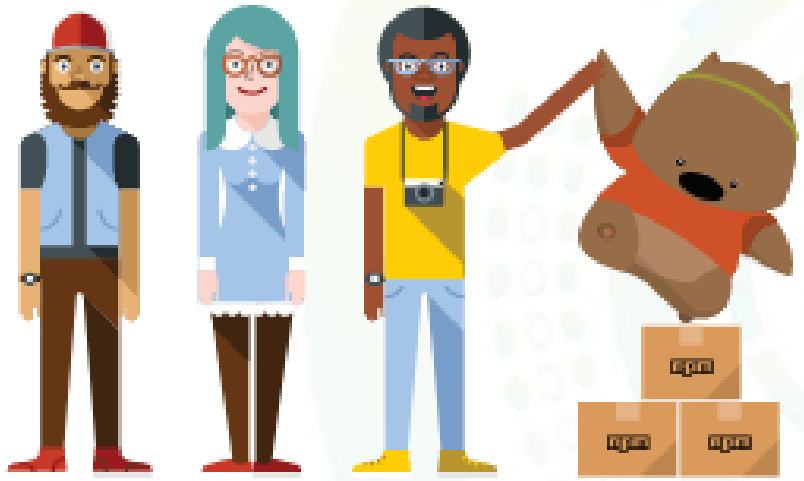
/node/order/detail?token=20d63dd23ce6486b8829cab3a759aa69

更好的做法

```
$ npm i -g @qnpm/qtoken
$ token cta1[redacted]12173434478
17310821905

orderToken:   c545d734ce7f4686bb
8520ab6b070[redacted]
```

这样做的好处

- 1）Npm模块简单易用
- 2）内部源，安装极快
- 3）私有模块，安全可靠
- 4）Cli支持更多参数
- 5）减少辅助开发的脏代码



Package.json配置文件

```
{
  "name": "@qnpm/qtoken",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "bin": {
    "token": "index.js"
  },
  "dependencies": {
    "request": "^2.81.0"
  }
}
```

参数处理

```
#!/usr/bin/env node

const argv = process.argv;
// argv.shift();
var orderno = process.argv[2],
    mobile = process.argv[3];

const file_path = __dirname;
const current_path = process.cwd();

const request = require('request')


var options = {
  proxy:"http://10      51.193",
  url: 'http://l-      .user.qunar.com/api/queryorder',
  method: 'POST',
  form: {
    orderno: orderno,
    mobile: mobile,
    prenum: '86',
    timeType:2,
    querytype:3
  }
};
```

核心代码

```
request(options, function (error, response, body) {

  if(error) {
    console.log('\n');
    console.log('\x1b[31m', 'error: ', error);
    console.log('\n');
  } else {
    var result = JSON.parse(body);
    console.log('\n');
    console.log('\x1b[33m%s\x1b[0m ', 'orderToken: ', result.data.ordertoken);
    console.log('\n');
  }
})
```

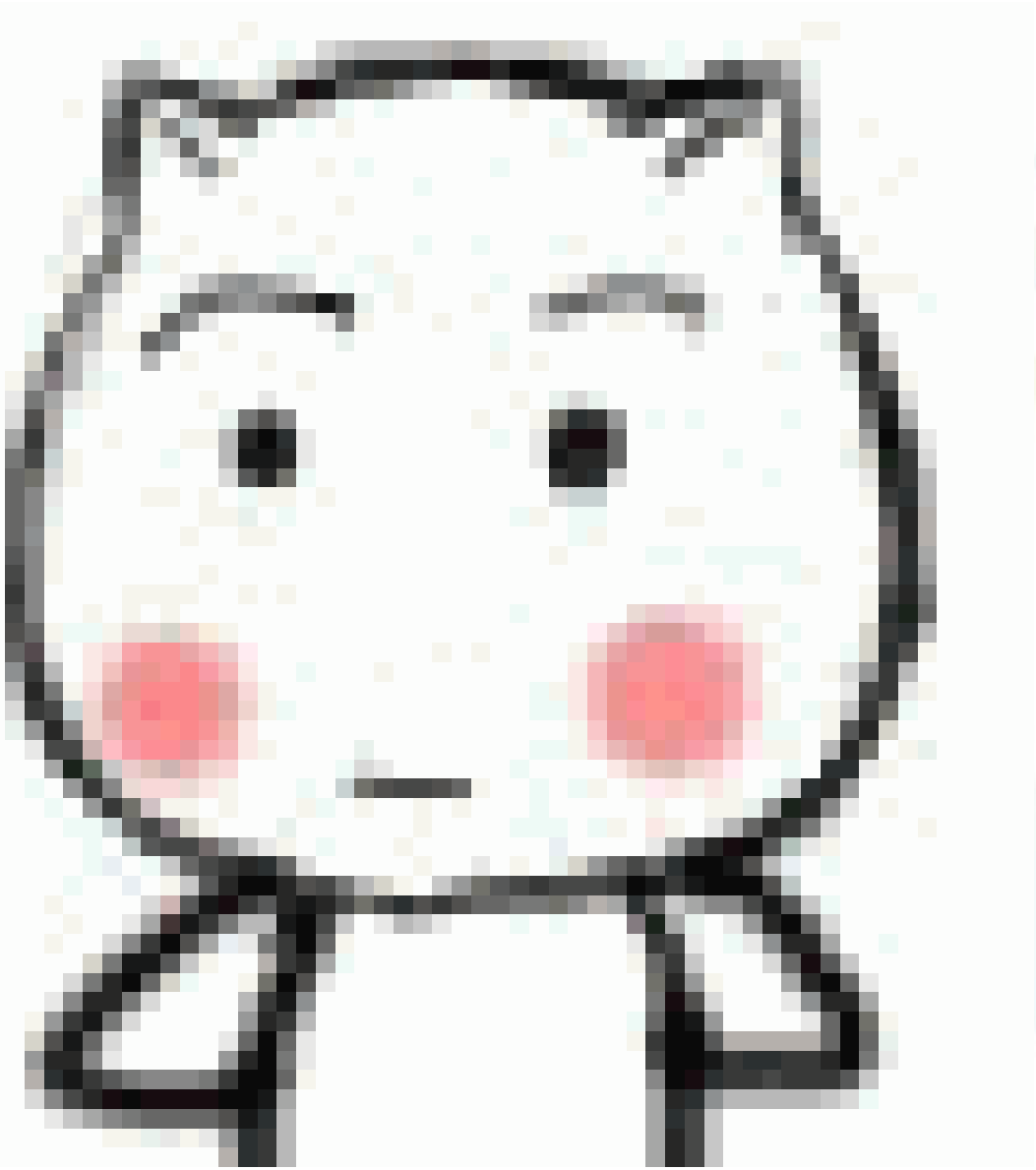
更多

- 这是beta环境可用，测试很方便
- 如果是线上的单呢？

参数解析

- https://github.com/tj/commander.js

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var program = require('commander');

program
  .version('0.1.0')
  .option('-p, --peppers', 'Add peppers')
  .option('-P, --pineapple', 'Add pineapple')
  .option('-b, --bbq-sauce', 'Add bbq sauce')
  .option('-c, --cheese [type]', 'Add the specified type of cheese [marble]', 'marble')
  .parse(process.argv);

console.log('you ordered a pizza with:');
if (program.peppers) console.log('  - peppers');
if (program.pineapple) console.log('  - pineapple');
if (program.bbqSauce) console.log('  - bbq');
console.log('  - %s cheese', program.cheese);
```



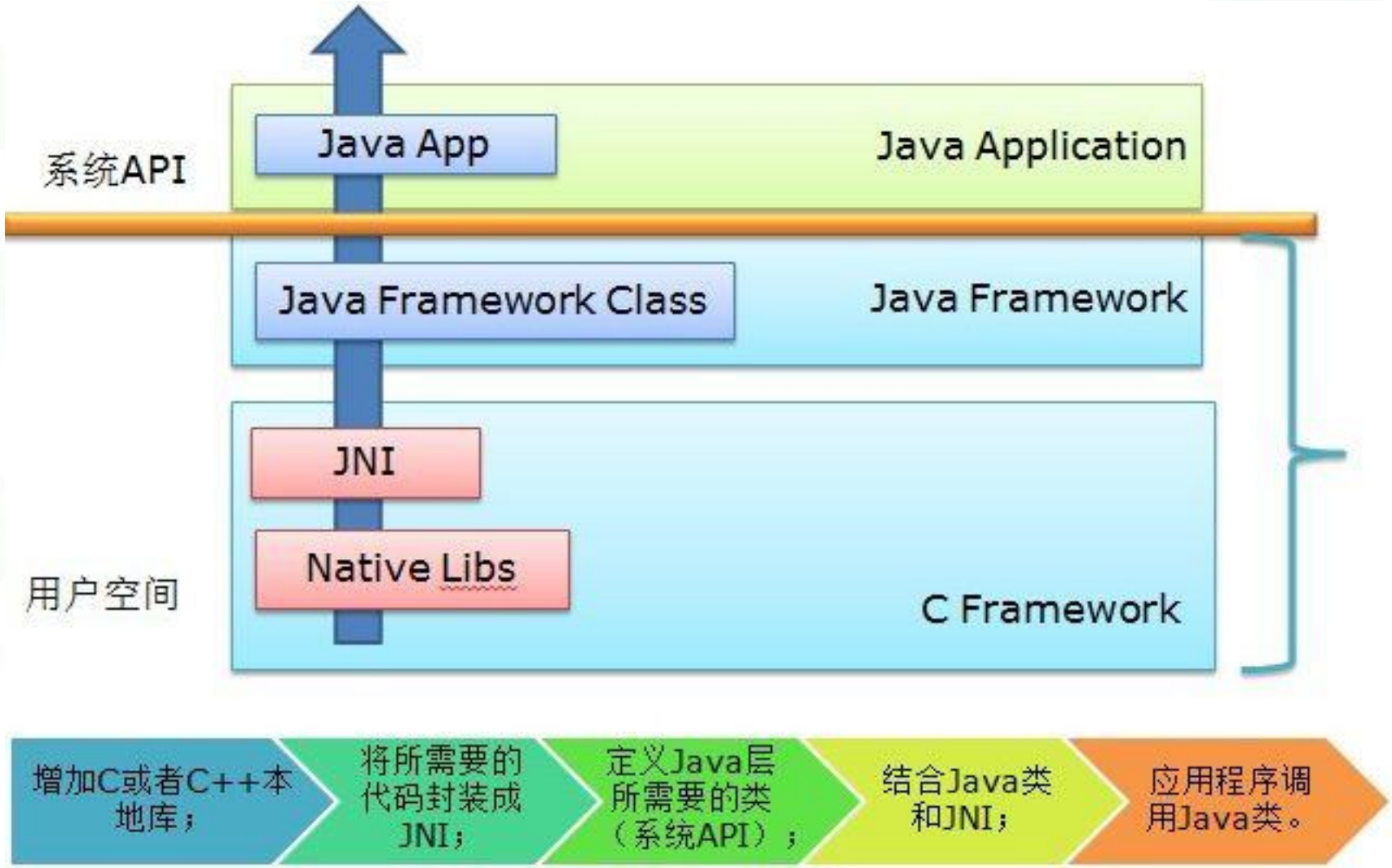
二、 node-java使用场景和原理

node-java是非常好的使用Node.js调用java代码的模块，可以无缝集成各种已有的Java世界的成熟稳定的模块，开发快速，是一种非常不错的解决方案。

NAN

- Native Abstraction for Node (NAN) makes it easier to write extensions
- Hides breaking changes in the V8 API
  - Your extension will support many versions!
- Functions and macros for common tasks
- https://github.com/nodejs/nan

JNI


系统API
Java App
Java Application
Java Framework Class
Java Framework
JNI
Native Libs
用户空间
C Framework
增加C或者C++本地库；
将所需要的代码封装成JNI；
定义Java层所需要的类（系统API）；
结合Java类和JNI；
应用程序调用Java类。

核心原理

> nan(Node.js调用c/c++)

> JNI (java调用c/c++)

> java反射机制

node-java

- 1）直接导入java的jar包，提供node可用api
- 2）自己通过java包一层，然后再提供node可用api

公司已有非常成熟的java版本的redis客户端，封装了各种安全相关机制，如果在Node.js项目里想使用redis，那么node-java会是最合适的选择，比自己使用node或nan写更好的

确保JDK支持JNI

在java8（主要是oracle的java sdk默认不支持JNI）里需要手动开启JNI设置，编辑 `/Library/Java/JavaVirtualMachines/<version>.jdk/Contents/Info.plist` 并把 `JNI` 设置为 `JVMCapabilities` 的选项:

```
<key>JVMCapabilities</key>
<array>
    ...
    <string>JNI</string>
</array>
```

创建ArrayListTest.java，注意文件名要和类名一致

```
import java.util.ArrayList;

public class ArrayListTest {

    public static void main(String[] args) {
        // 创建ArrayList
        ArrayList<String> list = new ArrayList<String>();

        list.add("1");
        list.add("2");
        list.add("3");
        list.add("4");
        list.add(3, "5");

        System.out.println(list.toString());
    }
}
```

java是通过jvm跨平台的，所以编译过程如下

- 第一步：javac会将.java文件编译成.class字节码文件
- 第二步：java调用jvm来执行.class字节码

编译并执行（这里没有引入package）

```
$ javac ArrayListTest.java  && java ArrayListTest
[1, 2, 3, 5, 4]
```

那么，在node-java里如何使用ArrayList呢？这里给出了4种方法

方法	描述	是否同步
java.newInstanceSync	根据类来新建实例	Y
java.newInstance	根据类来新建实例	N
java.newInstancePromise	根据类来新建实例	Y
java.import	导出类	Y

示例：

```
const java = require("java")
const ArrayList = java.import('java.util.ArrayList')

let list = new ArrayList()
list.addSync("1")
list.addSync("2")
list.addSync("3")
list.addSync("4")
list.addSync(3, "5")

console.log(list.toStringSync())
```

引用jar包

```
#!/usr/bin/env node

var java = require("java");
java.classpath.push("lucene-core-6.0.0.jar");
java.classpath.push("lucene-analyzers-common-6.0.0.jar");
java.classpath.push("lucene-queryparser-6.0.0.jar");
```

异常处理

a）同步代码，使用try/catch

```
try {
  java.methodThatThrowsExceptionSync();
} catch(ex) {
  console.log(ex.cause.getMessageSync());
}
```

b）异步代码

比如Promise，可以使用对应的catch()或then()方法处理

```
java.newInstancePromise("java.util.ArrayList")
    .then(function(list) { return list.addPromise("item1"); })
    .then(function(list) { return list.addPromise("item2"); })
    .catch(function(err) { /* handle error */ });
```

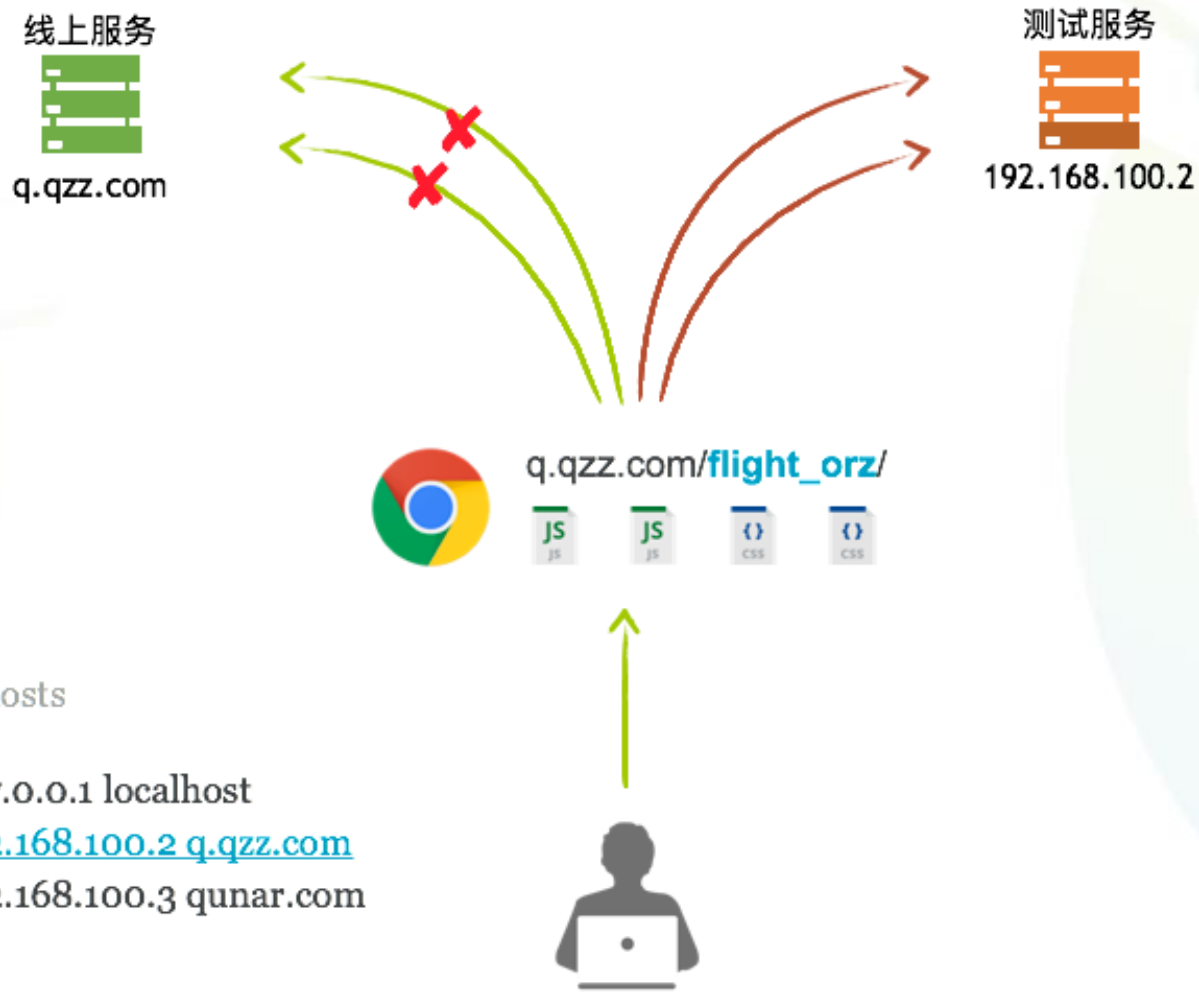
Node.js扩展对比

语言	稳定实现	新的实现
c/c++	nan	N-api
java	Node-java	暂无
rust	neon	node-api

三、为什么一定要用multihost和nginx？

重新思考一下我们的开发方式

前端环境复杂的只能host



后端呢？

hosts

127.0.0.1 localhost
192.168.100.2 q.qzz.com
192.168.100.3 qunar.com

Nginx

```
location /flight_orz/ {
    proxy_pass http://192.168.100.2;
}
```

维护很多套配置

```
# Nginx

location /flight_orz/ {
    proxy_pass http://192.168.100.2;
}
```

```
# Nginx

location /flight_orz/ {
    proxy_pass http://192.168.100.2;
}
```

疼？

multi-host原理

- Node启动一个代理服务器
- 通过/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --proxy-server=http://127.0.0.1:9393打开浏览器
- 在代理服务器里，改写req.host

后端

- request
- q-request

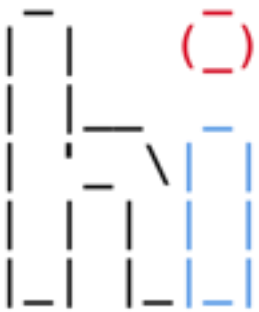
```
process.env.http_proxy = 'http://127.0.0.1:9393'
```

所有的q-request发送的api都会走本地的multi-host代理
问题来了：如果有的是本地，有的是服务器呢？

代理 + rewrite

我们需要一个带有rewrite功能的代理

All in one

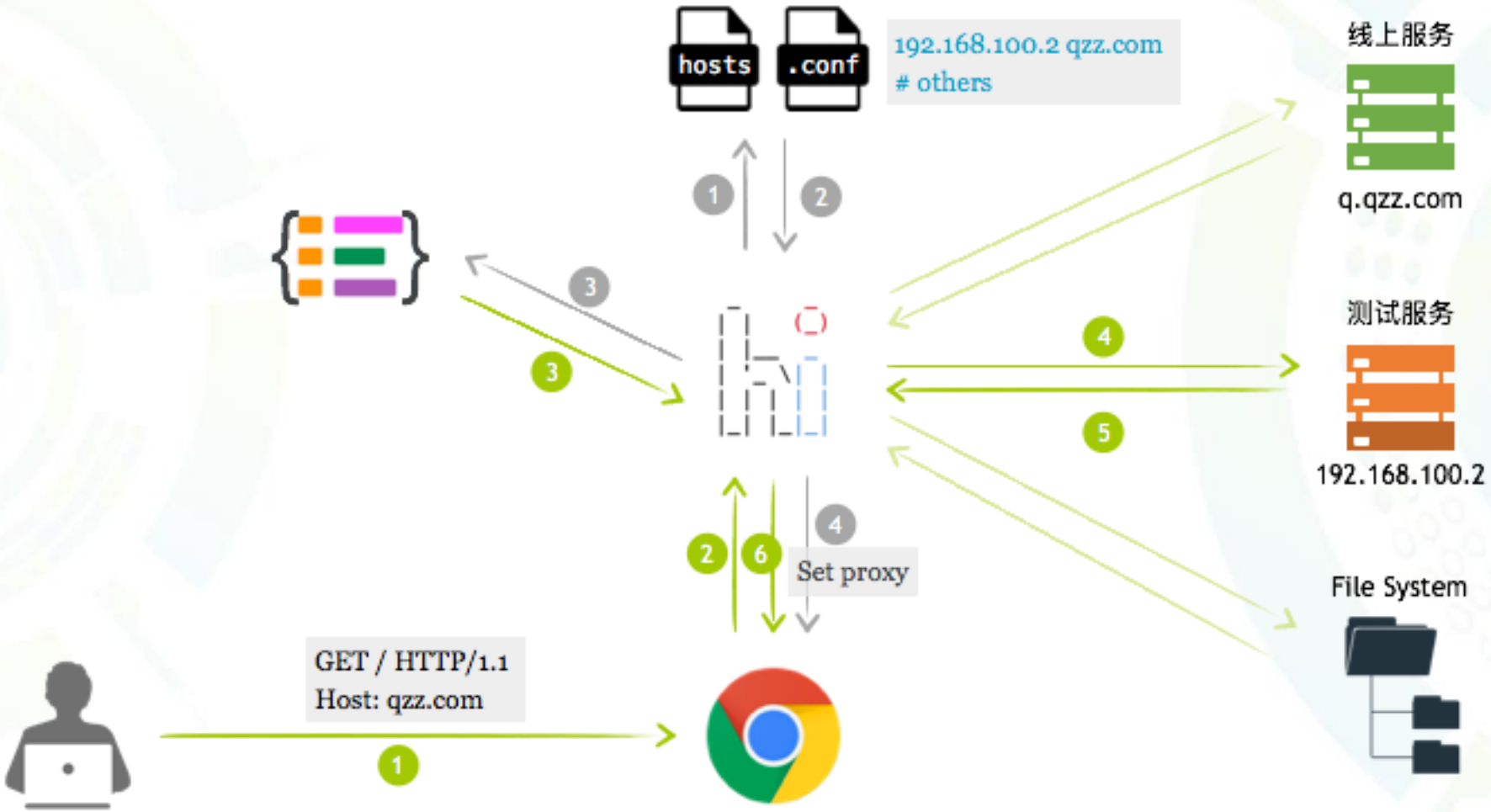


Rewrite规则配置

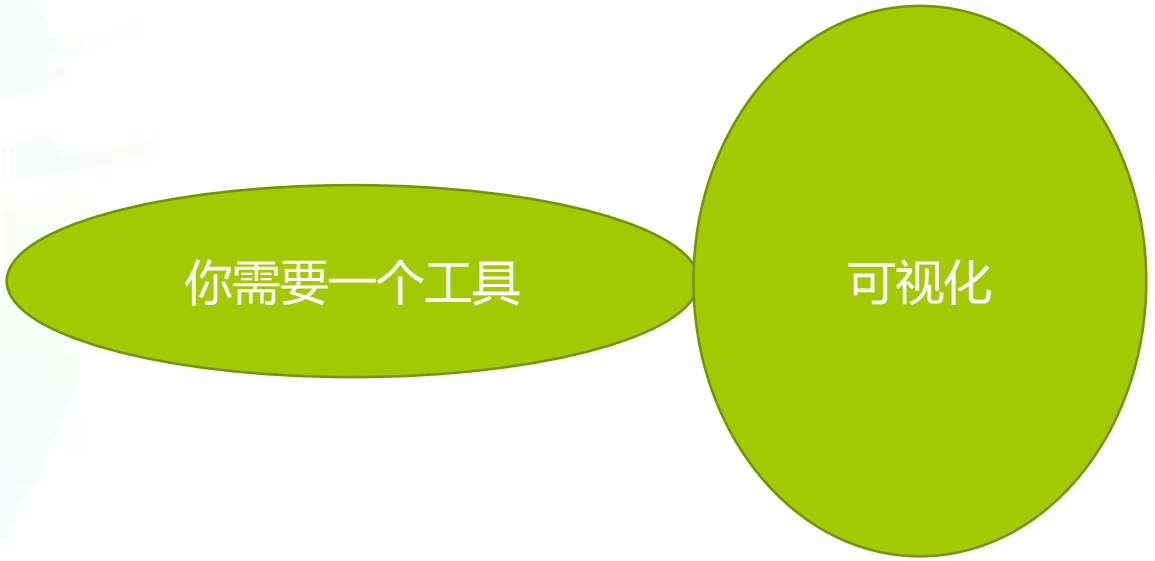
```
# Rewrite Rules

set $local 127.0.0.1:8800;

domain www.example.com {
   location / {
      proxy_pass http://$local/;
      set_header  Access-Control-Allow-Origin *;
      set_header Server hiproxy;
   }

   location ~ /\/path\/(.*) {
      proxy_pass http://example.io/$1;
      proxy_set_header Host example.io;
      # other directives
   }
}
```

- ☑ 语法跟Nginx配置语法类似
- ☑ 支持正则表达式 / 分组
- ☑ 支持代理请求头设置
- ☑ 支持返回头设置

hiproxy代理原理



更多



总结回顾

- 1）用户订单token如何通过命令行获取
- 2）node-java使用场景和原理
- 3）重新思考一下我们的开发方式，为什么一定要用multihost和nginx？

我们应该有更好的开发方式

更多乐趣，自己发现吧

1）初衷，server端，不想成了前端开发的基础设施
2）命令行辅助工具，甚至可以是运维
3）移动端：cordova，pc端：nw.js和electron
4）组件化，构建，代理
5）架构，前后端分离、api proxy
6）性能优化、反爬虫与爬虫
7）全栈最便捷之路

Q&A

狼叔说：少抱怨，多思考，未来更美好

http://i5ting.com

i5ting@126.com