技术进展调查报告

——Node网页框架Express vs Koa vs Hapi

**1背景**

Node是基于Google的V8引擎打造了基于事件循环实现的异步I/O框架，而V8引擎提供的远超其它脚本语言的性能及异步I/O的特点正是Node成功的极其重要的因素。它是目前最火热的后端技术，在开源社区Github上，Node高居第二。我们小组创建的是teambuilder网站，本着对新技术的热爱和学习，我们选择了Node作为网页后端。

选择了Node，接下来就是要寻找一份合适的Web开发框架，有如php之Laravel、CodeIgniter，Python之Django、Tornado以及Ruby之Rails。借助各种社区推荐，我们发现了Express，Koa，Hapi这三款比较热门的基于Node的网页框架。

**2 框架的背景**

**2.1 Express**

Express无疑是当前Node中最流行的Web应用框架，它几乎成为了大多数Node Web应用程序的基本的依赖，甚至于在介绍Node的书籍上，Express的影子也是处处可见。Express 是一个小巧且灵活的 Node.js Web应用框架，它有一套健壮的特性，可用于开发单页、多页和混合Web应用。根据其github上的README.md，这个框架被描述成：疯一般快速(而简洁)的服务端Javascript Web开发框架，基于Node.js和V8 JavaScript引擎。

**2.2 Koa**

相对Express，Koa这个框架有着更新的历史。于2013年8月17日，Express的作者TJ Holowaychuk提交了Koa的第一次commit，他描述Koa为“表现力强劲的Node.js中间件，通过co使用generators使得编写web应用程序和REST API更加丝般顺滑”。其中generators是ECMAScript 6中的新技术，也即JavaScript的下一个版本，而co[http://github.com/tj/co]是基于Node的一个ECMAScript 6的实现。

**2.3 Hapi**

Hapi是三个中最与众不同的一个，它基于一个观念：配置优于编码，业务逻辑必须和传输层进行分离。起初Hapi是基于Express构建的，后来才发展成自己的框架。

**3 社区热门度**

以下是这几个流行框架的社区数据：

**Metric** Express.js Koa.js Hapi.js

Github Stars 16,158 4,846 3,283

Contributors 163 49 95

Packages that depend on 3,828 99 102

StackOverflow Questions 11,419 72 82

由上可见，正如刚刚所说，Express的热门度远远超过另外两个，而Koa与Hapi更像是新兴的。StackOverflow是国外一个十分著名的提问社区，Express有问题量大大超过另两个，正是代表着它的使用者的数目。Github Stars代表着其项目受关注的程度，相对于StackOverflow Questions，三者的Github Stars相差没那么大，除去Express出现的年代比另两个早的原因，其实他们受关注的程度已经接近，这也从某种程度说明了Koa及Hapi的发展潜力。

**4 HelloWorld**

HelloWorld是程序开发中最经典的入门例子，现在分别用三个框架来实现一个最基本的网页服务，对比它们三个的区别

**4.1 Express**

var express = require('express');

var app = express();

app.get('/', function(req, res) {

res.send(''welcome to Teambuilder);

});

var server = app.listen(80, function() {

console.log('Express is listening to http://localhost');

});

Express使用get()函数来捕获’GET /’请求然后调用一个回调函数，这个回调函数会被传入req和res两个对象。这个例子当中我们只利用了res的res.send()来返回整个页面的字符串。Express有很多内置的方法可以用来进行路由控制。get, post, put, head, delete等等这些方法都是Express支持的最常用的方法。

**4.2 Koa**

var koa = require('koa');

var app = koa();

app.use(function \*() {

this.body = 'teambuilder';

});

var server = app.listen(80, function() {

console.log('Koa is listening to http://localhost');

});

Koa与Express有稍微的不同，但这却是一个很大的不同点。因为它用了ES6的generators。所有带有\*前缀的函数都表示这个函数会返回一个generate对象。 generator对象会同步地yield出数据，个人对ES6的学习尚浅，还不能明白生成器的用法。在Koa中，this这个上下文其实就是对node的request和response对象的封装。以上代码中，function\*捕捉了所有的路由，并将this的body属性赋值为一个字符串。

**4.3 Hapi**

var Hapi = require('hapi');

var server = new Hapi.Server(80);

server.route({

method: 'GET',

path: '/',

handler: function(request, reply) {

reply('teambuilder');

}

});

server.start(function() {

console.log('Hapi is listening to http://localhost');

});

正如之前所介绍的，Hapi崇尚于配置式代码，以上的路由就是一个典型的例子，代码把路由的要素都写在一个javascript对象字面量中，modhod指方法，path是url，handle就是捕捉到它后会调用的回调函数，在这里，Hapi为回调函数提供了request和reply对象，相当于Express中的req和res，以及Koa中的this。

可见三者的路由都使用了相似的接口。

**5 优缺点比较**

**5.1 Express**

**5.1.1 优点**

Express作为三者中出现最早的一个，它不仅拥有最大的社区，并且是所有Node Web程序框架当中最大的一个。在经过多年的发展及广大社区的维护和反馈，它已经发展得十分成熟，支持多种template engines，比较受欢迎的有ejs，jade和haml等，很好地解决诸如路由、缓存、中间件、cookie、session、template 之类的 web 典型问题，利于Web Developer快速构建Web应用。

**5.1.2 缺点**

Express自称为高度可配置，这有其中的好处，但也带来了一定的坏处。Express的特性自然而然地引来了各式各样的中间件，而对于最开始接触Node的程序员来说，这不一定是好事，他们可能无法轻易理解中间件的含义，而在开发过程中处处碰壁，也可能因为社区提供的众多的中间件库而迷失其中，忘记本源。另外，Express没有内置错误处理，开发者须要不断手动处理各种异常，这是一个乏味的工作。还有一个就是，Express体积很大。

**5.2 Koa**

**5.2.1 优点**

不像Express，Koa的体积小多了。使用了generators的它在写起中间件时要比另两个框架轻松得多，而且它并没有内置的中间件，我们可以自由地定义或选择与我们的Web程序所适应的中间件(当然，这也不容易)。Koa十分简洁，会让人感觉到代码是如此的轻巧、漂亮。

**5.2.2 缺点**

优秀自然会对应着不足，比如Koa正处于活跃的开发完善阶段，不比于Express的成熟阶段，在很多使用环境上还可能出现小的问题。它的社区尚未发展到比较大的程度，或许有些问题不一定能轻易找到答案。Koa使用的ES6也并没有在Node最新的稳定版中得到支持。

**5.3 Hapi**

**5.3.1 优点**

Hapi的最大优点是配置优于编码的概念，这在大型的团队项目开发中，会让编码变得十分具有可维护性，因为统一化地配置可以让代码更加整齐、易理解。而且也不少大型企业已经在实际生产中使用它，故在成熟性上其要比Koa要高上许多。

**5.3.2 缺点**

Hapi很适合用来做大型复杂项目，但对于一些不是简单小型的项目来说，配置化的代码会使它们变得太死板，而缺少灵活性。另外就是Hapi的样例好少，很多都是不开源的，学习成本较高。

**6 总结**

经过以上比较，结合看下我们的teambuilder项目的特点。它不是大型的，我们也是需要在比较短的时间内把系统搭建起来。而当选择一个框架时，我们需要一步一步的学习，有时也需要在网上参考一些例子。这些特点，让我们决定选择Express，它也许有些不好的地方，不过正如前面所比较的，他拥有庞大的社区群体，可以参照的问题也是很多。这样能减少我们在开发路上的磕碰，更快更好地开发出我们的目标产品。