# **第二章 相关技术概述**

## 2.1 Node.js

### 2.1.1 Node.js

Javascript最早是运行在浏览器端，然而浏览器只是提供了一个上下文。事实上，JavaScript是一门完整的语言，可以使用在不同的上下文中。在Node.js出现之前，javascript通常作为用户端程式设计语言使用，以javascript写出的程式常常在客户的浏览器上执行。Node.js实际上就是另外一种上下文，Node.js的出现使Javascript脱离浏览器环境，用于服务器编程，在后端运行javascript代码。要实现在后台运行JavaScript代码，代码需要先被解释然后正确的执行。Node.js是一个基于chrome V8引擎的javascript运行时环境，来解释和执行JavaScript代码。 V8引擎是基础BSD许可证的开源软件，速度非常快且专注于网络功能，在HTTP\DNS\TCP等方面更佳成熟。Node.js使用高效、轻量级的事件驱动、非阻塞I/O模型和非同步输入输出模型等技术来提高效能，可优化应用程式的传输量和规模。事件驱动这一特性，让开发者可以在不使用线程的情况下开发出一个能够承载高并发的服务器。此外，Node.js的生态系统NPM是目前最大的开源包管理系统，这些十分有用的模块可以简化很多重复的劳作。 Node.js允许通过Javascript和一系列模块来编写服务器端应用和网络相关的应用。核心模块包括文件系统I/O、网络、二进制数据流、加密算法、数据流等。因此，Node.js既是一个运行时环境，同时又是一个库。Node模块的API形式简单，降低了编程的复杂度。使用框架可以加速开发，常用的框架有Express.js、Socket.io和Connect等。

### 2.1.2 Express框架

Express是一种以最低程度规模依旧能保持灵活的Node.js Web应用程序框架,为Web和移动应用程序提供一系列强大的功能,例如各种HTTP使用程序方法和中间件,帮助开发者快速构建Web应用。Express框架允许我们通过设置中间件来响应HTTP 请求。中间件就是处理http请求的函数，其特点为：一个中间件处理完，再传递给下一个中间件。app实例在运行过程中，会调用一系列的中间件。并且Express.js定义了路由表用于执行不同的HTTP 请求动作，具体的说就是指定不同的访问路径，指定不同的处理方法。此外Express

.js支持多种模版，可以通过向模板传递参数来动态渲染HTML页面，渲染就是指将数据代入模版的过程。实际运用中，数据都是保存在数据库中的，这里未来便于理解。Express框架建立在Node.js内置的HTTP模块上,HTTP模块生成服务器的原理的关键是createServer方法,表示生成一个HTTP服务器实例。接受一个回调函数,该回调函数的参数,分别为代表HTTP请求和HTTP回应的request对象和response对象。Express框架的核心是对HTTP模块的再次包装。Express提供方法来区分HTTP方法（GET、POST、SET）和URL模式。虽然Express本身相当简约，但是开发人员已经创建了兼容的中间件软件包来解决几乎任何Web开发问题。例如已经有很多库致力于解决cookie、会话、用户登录、URL参数、数据、安全等。

## 2.2 测试

### 2.2.1 mocha.js

为了确保质量控制，测试驱动开发以及行为驱动开发就成为了重要的一个环节。使用测试驱动开发可以很大程度上简化编写代码的思考过程。Mocha.js是一个强大且成熟，适用于Node.js开发环境和浏览器的测试框架。与Jasmine以及其他完整的测试套件不同，Mocha可以被理解为一个任务运行器，他将重点侧重在整体结构，允许开发者使用模块化方法去测试Node.js应用程序；而并不执着于实际的断言，Mocha在运行测试用例过程中，当捕捉到错误时，依旧能够灵活的运行精确的报告。这有利于我们持续的对测试进行观察，与此同时在选择断言库方面也给予了我们很大选择的空间。于是，编写的测试的结构决定着怎样让测试知道代码运行是否失败以及测试结果的显示。代码的创建方式，就是使用多个describe块来展示库中的不同组件，然后在其中添加it块来指定一个具体的测试用例。描述一句测试用例是否正确，首先describe语句是可以嵌套的，多个describe嵌套的使用可以用于描述模块下的子模块的关系；真正的测试语句是在it函数的回调里面，info也是一句描述性的说明，callback里面的断言才真正决定这条测试是否正确。如果失败的话控制台会把log打印出来。一个it对应一个测试用例，里面可以有多条断言或一条断言。以下是一个测试用例的例子。

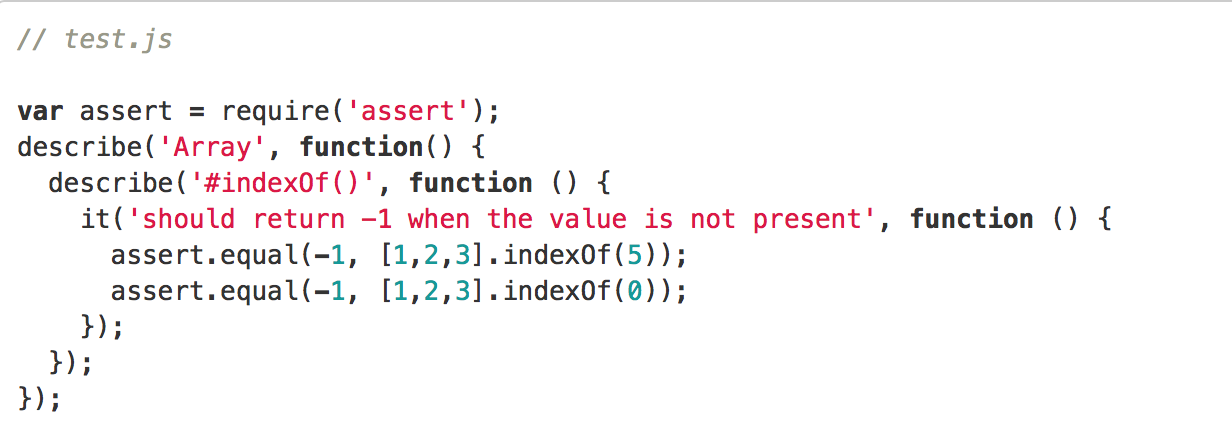


图2.1 Mocha.js测试例子

### 2.2.2 chai.js 断言库

chai.js是一个十分优秀的assert库的代替选择，是一套适用于测试驱动开发和行为驱动开发的断言库。可以在Node和浏览器环境运行,可以高效的和任何javascript测试框架搭配使用。Chai.js提供了三种断言风格来分别适合于BDD和TDD。Expect/should API对应BDD风格；Assert API对应TDD风格。BDD风格主要包括两个方法，分别是expect和should。都可以通过链接的方式构造断言，但主要区分在于断言的初始构造。Should的风格非常接近于自然语言。Expect和should的区别在于：expect需要的只是对象的引用，而should需要函数先执行。Expect接口提供的函数可以作为链接断言的起点，并且适用于Node和所有浏览器环境，而should接口继承自Object.prototype，提供的仅仅是一个getter作为链接断言的起点，并且不能在ie浏览器上工作。因为Mocha没有内置的Assertion,Mock和Stub功能,所以在本应用中,我们用Chai来为框架提供断言,用Sion为其提供Mock和Stub功能。

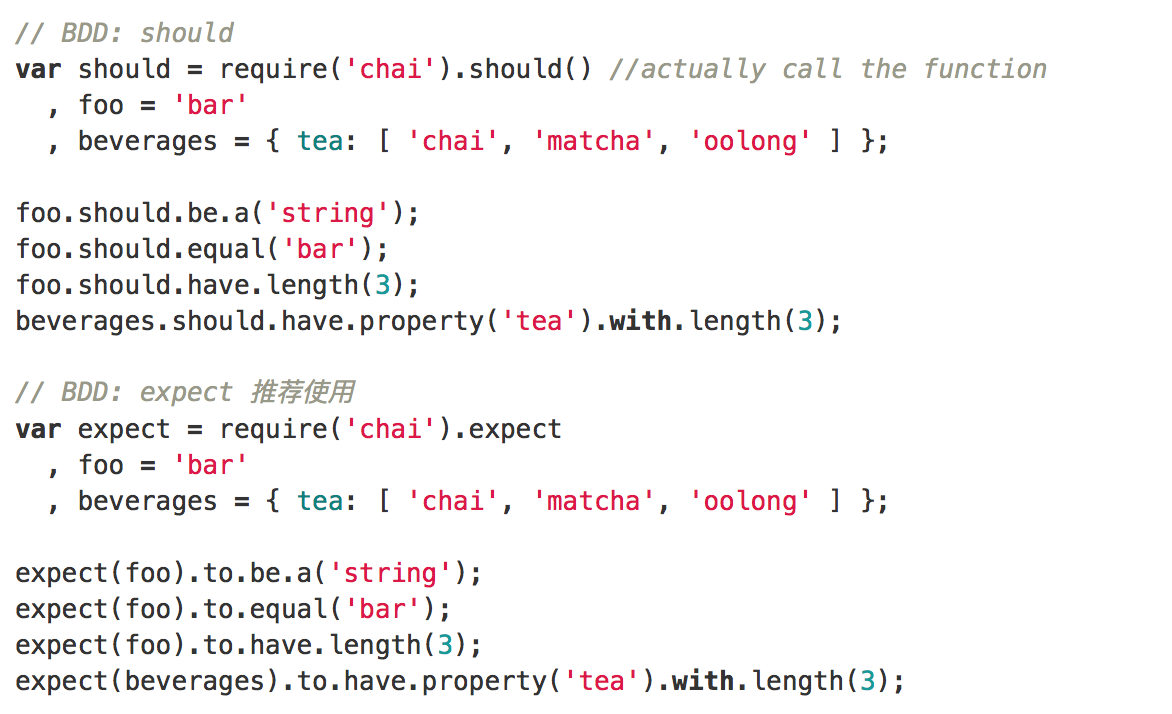


图2.2 Chai.js中should与expect的对比

## 2.3版本控制技术

### 2.3.1 Github技术

Git是分布式版本控制系统,客户端比知识提取最新版本的快照,而是将整个代码仓库镜像复制下来。Git能够对文件版本控制和多人协作开发,拥有强大的分支特性,所以能都灵活地以不同的工作流协同开发。如果任何协同工作用的服务器发生故障了,也可以用任何一个代码仓库来恢复。即使在协作服务器宕机期间依旧可以提交代码到本地仓库,当协作服务器正常工作后,再将本地仓库同步到远程仓库。 当团队中某个成员完成某个功能室,通过拉取代码请求（pull request）操作来通知其他团队成员,其他团队成员能够review code后在合并代码。在Git中,有三种文件状态：已修改（modified）,已缓存（ staged）,已提交（committed）, 几乎所有操作都是在本地执行,直接记录快照而不是差异比较,这一点也保证了Git能够时刻保持数据完整的特性。

## 2.4 NPM 包管理器

在Node 中使用NPM(Node Package Manager)，即Node包管理器。NPM提供了一个公共的注册服务,包含了大家发布的所有包，并提供了一个命令行工具，用来下载、安装和管理第三方。这意味着NPM包管理器允许你在一个沙盒里下载和使用模块,而不用担心污染全局的包环境。NPM包含三部分：存放第三方包的代码库、管理本地已经安装包的机制、定义包依赖关系的标准。

Node的操作主要有两种模式：全局和本地,这两种模式会影响包存放的目录结构以及加载包时的顺序。本地模式是NPM的默认模式,在这个模式下,NPM只工作在工作目录下,不会造成系统范围的修改。本地模式下,NPM安装的所有东西都在当前工作目录而不会影响任何全局的设置。这种机制让开发者可以一个个的设置应用程序的依赖模块以及他们的版本,这意味着开发者可以有依赖同一个模块不同版本的两个应用且彼此之前不会产生冲突。又因此,本地模式优先级总是高于全局模式。而至于全局模式则适合那些将会被很多程序使用并且总是被全局加载的公共模块，比如命令行工具这些不会被应用程序直接使用的模块。

在开发过程中，针对那些我们所依赖的第三方模块，开发者可以使用package.json文件定义该应用程序的依赖关系。具体的说，当你把项目源码从开发环境部署到生产环境，可以通过调用npm install来一次性安装所有依赖包。NPM会通过package.json内指定的依赖关系来自动完成依赖模块的下载安装。

## 2.5 Commonjs模块规范

Commonjs规范是为了解决javascript的作用域问题而定义的模块形式，可以使每个模块在自身的命名空间中执行。Commonjs的核心思想是允许模块通过require方法来同步加载所要依赖的其他模块，然后通过exports或者module.exports来导出需要暴露的接口。

具体来说，Node应用由模块组成,采用commonjs模块规范，每个文件就是一个模块，有自己的作用域。Node内部提供一个module构建函数，所有模块都是module的实例。Commonjs规范规定,每个模块内部，都要一个module对象，代表当前模块。Node为每个模块一个exports变量，指向module.exports。module.exports表示当前模块对外输出的接口，其他文件加载模块，其实是读取该模块的module.export属性。Require方法用于加载模块。

Commonjs模块规范有如下特点：1.所有代码都运行在模块作用，不会污染全局作用域。2.模块可以多次加载，但是只会在第一次加载时运行一次，然后运行结果就被缓存了。以后再加载就直接读取缓存结果，要想让模块再次运行，必须清楚缓存。3.模块按照其在代码中出现的顺序加载，Commonjs规范加载模块是同步的，只有加载完成才能执行之后的操作。但是同时Commonjs规范有一定的缺陷。例如：同步的模块加载方式并不适合在浏览器环境中，因为同步意味只阻塞加载，也就意味着应用并不能非阻塞的并行加载多个模块，然而浏览器资源是异步记载的。但是这对于服务器不是一个问题,因为所有的模块都存放在本地硬盘，可以同步加载完成，等待时间就是硬盘的读取时间，所以在本应用中，我们仍然选择使用Commonjs模块规范。