# Table of Contents

# Introduction

# 目录

# 常用方法

## queryString

## bodyParser

# CommonJS

> 所有代码都运行在模块作用域，不会污染全局作用域。 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，必须清除缓存。 模块加载的顺序，按照其在代码中出现的顺序。

# 模块上下文

1. 在一个模块中，存在一个自由变量 require 函数
    - require 函数接受一个模块标识符
    - require 函数返回外部模块导出的 API
    - 如果存在依赖循环，那么外部模块在被它的传递依赖(transitive dependencies)所 require 的时候可能并没有执行完成，这种情况下 require 返回的对象必须至少包含此外部模块在调用 require 函数之前就已经准备完毕的输出。
    - 如果无法返回所请求的模块，则 require 必须抛出一个错误。
    - require 有一个只读的、不可删除的 main 属性。main 相当于程序根目录的 module。如果设置了该属性，则其必须和根目录的 module 指向相同的对象。
    - require 功能可能具有 paths 属性，即到顶级模块目录的路径的从高到低的路径字符串的优先数组。
        - paths 属性不会存在于沙盒中。
        - 在所有模块中 paths 的 attribute 均指向相同的值。
        - paths 是无法被替换的。
        - 当 paths 的 attribute 存在时，修改 paths 的内容可能会导致模块无法被正确的搜索到。
        - 当 paths 的 attribute 存在时，它可能只包含了部分 path,当模块加载器在使用这些路径之前或者之后，去检查其它的路径。
        - 当 paths 的 attribute 存在时,它是模块加载器使 paths 规范化、标准化的依据。
2. 在模块中，有一个名为 exports 的自由变量，该变量在模块执行时可以向其添加 API 的对象
    - 模块必须使用 exports 对象作为唯一的导出方法。
3. 在模块中，必须有一个自由变量 module，即一个对象
    - module 对象有一个只读的、不可删除的 id 属性，它是 module 的顶层 id。当执行 require(module.id)时，可以通过该 id 找到对应的 module 并返回 module exports 出的对象。

- 当创建一个 module 对象时，该对象可以有一个 URI 属性。该属性指向对应的模块源文件。该 URI 不存在于沙盒中。

# 模块标示符

- 模块标示符是一个由正斜杠分隔的”terms“组成的字符串。
- 一个 term 的必须为驼峰样式标识符，或者为”.“或”..“。
- 模块标识符可以省略文件名的后缀。比如”.js“
- 模块标识符可以是相对路径(relative)或者绝对路径(top-level)。如果模块标识符的开头是”.“或者”..“则此模块标识符为相对路径。
- 绝对路径必须是模块所在命名空间的根。
- 相对路径必须是相对于当前 require 的模块。