Abstract

Node.js is an open source, cross-platform JavaScript runtime environment, which executes JavaScript code outside the browser. Node.js allows developers to write command-line tools using javascript, and allows server-side scripts that run scripts to generate dynamic web page content before the page is sent to the user's web browser. Node.js has an event-driven architecture and can be asynchronous I / O. These design choices aim to optimize the throughput and scalability of web applications with many input/output operations, as well as real-time web applications. In this chapter, the overarching architecture of Node.js is analyzed using different views and perspectives range from the stakeholders perspective to functional perspective and more. Lastly is a brief conclusion about what we have discovered about the architecture of Node.js.

Introduction

Node.js is a chrome-based JavaScript runtime platform, which can easily build fast and scalable network applications. Node.js uses an event-driven non-blocking I/O model, making it lightweight and efficient, and very suitable for data-intensive real-time applications running on distributed devices. Node.js application program is written in JavaScript and can run on OS x, Microsoft Windows and Linux when node.js runs. It also provides a rich library of various JavaScript modules, which greatly simplifies the process of developing web applications using Node. js.

Node.js was originally written by Ryan Dahl in 2009, about 13 years after the introduction of the first server-side JavaScript environment. The original version only supported Linux and

MacOS X. Its development and maintenance was led by Dahl and later sponsored by Joyent. Dahl criticized the limited possibilities of Apache HTTP server, the most popular Web server in 2009, for handling large numbers of concurrent connections (up to 10,000 or more) and the most common code creation (sequential programming) when code blocked the entire process or implied multiple execution stacks in the case of simultaneous connections, and that's why he wrote Node.js.

A key advantage of node.js is that developers find it easy to extend applications horizontally and vertically. By adding additional nodes to existing systems, applications can be scaled horizontally. In addition, Node. JS gives you the option to add additional resources to a single node during the vertical extension of the application. Therefore, it is highly scalable and offers a better choice than other JavaScript servers. Also, since the Node.js is providing the option of non-blocking I/O systems, it relatively helps you to process several requests concurrently. The system can handle the concurrent request handling efficiently better than others. The incoming requests get lined up and are executed quickly and systematically.

And the great popularity of node.js thus absorbs us to study about its architecture. We will first go through the stakeholders and the context view to have an understanding of what are involved in the architecture of the Node.js. Next, we will focus on some significant functionalities of Node.js in the function view. We end with the analysis of its performance and scalability which are embodied from its architecture.

Stakeholders view

In a large open source project like Nodejs, multiple types of stakeholders can be identified. Table 1 introduces the stakeholders involved in theNodejsproject.

Stakeholder class	Description
Acquirers	The Node.js project is sponsored by the Node.js Foundation. As the corporate steward of Node.js, co-founder of the Node.js Foundation, and one of its largest-scale production users, Joyent, a provider of modern, open cloud infrastructure as a service which is also a wholly-owned subsidiary of Samsung, is uniquely equipped to deliver the highest level of support for powerful application frameworks and APIs.
Communicators	The Community Committee is a top-level committee in the Node.js Foundation focused on community-facing efforts. The Community Committee reflects a formal role in the governance of the Node.js project. The formation of this group as a committee alongside the Technical Steering Committee (TSC) demonstrates that community-focused contributions are valued by the contributors of

	Node.js.
Development & Testing	The project is governed jointly by the Technical Steering Committee (TSC), who is responsible for high-level guidance of the project. TSC consists of key Collaborators who have demonstrated both technical expertise critical to the ongoing maintenance and evolution of the project and a long-term commitment to driving the project and community forward.
Suppliers	Nodejs is built upon Java and Chrome V8. The platform of Nodejs is based on Chrome JavaScript runtime. Its testing framework is Mocha, which can provide high quality of testing.
Support	Core Working Groups and Users are part of thesupport staff.This support is mostly provided throughthe Nodejs repository.
Users	Nodejs has a very broad user base which can be separated intothree categories: individual programmers, who use node.js to build their own pages, small projects needinginsights into their builds, and companies. What's more, as predominately developers,users can make change to this framework to their use, which can be considered as system administrators of their own project.
Competitors	Other parties delivering similar services are for example WebBrowser.
Maintainers	The in-house developers together with the Nodejs community ofusers, contribute to the maintenance of the system.Overall,Ryan Dahl and Ben Noordhuis seem to invest the mosteffort in maintaining the project.

Table 1: Stakeholders of the Nodejs project

The stakeholders are visualized in Figure 1, their respective power and interest in the Gradleproject are shown in Figure 2.

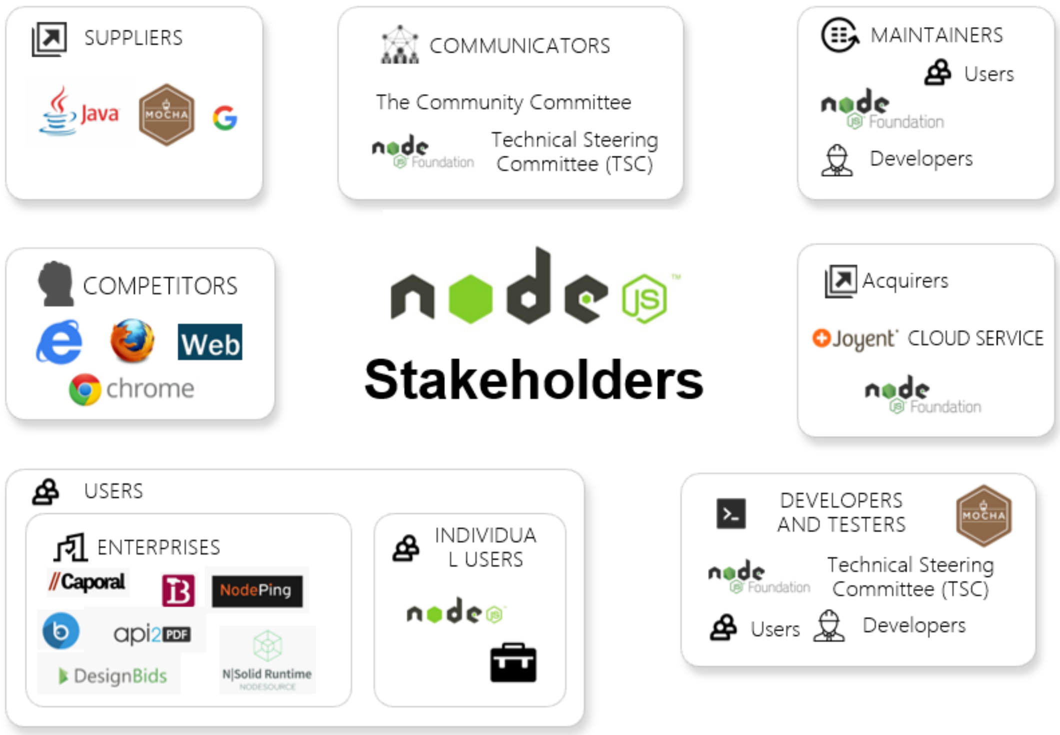


Figure 1:An overview of all the stakeholders involved in the Nodejs project

Power-Interest grid

In the stakeholder analysis, many different actors with different roles, power and interest have been analysed and visualised inFigure 2.

The main stakeholder of Nodejs is Joyent, which is a wholly-owned subsidiary of Samsung. In the late 2011, Node was funded by cloud computing service Joyent, and founder Ryan Dahl joined Joyent to take full responsibility for Node's development.

As Node.js Foundation is sponsoring this project, so the Node.js Foundation must be quite interested with its trade tendency, which is quite related with its financial interest. Also, with contributing money to this project, Node.js Foundation can make the trademark guidelines of the Node.js project, so Node.js Foundation is of great interest and also power in the project.

Since TSC members are responsible for top level technical community concerns. The role is mostly administrative and is responsible for admitting new Top Level Projects, Top Level Working Groups, and advocating for any needs in the technical side of the foundation to the Node.js Foundation Board of Directors. So developers and testers have much interest on nodejs project, but without too much power on making the future direction decision of nodejs.

Suppliers may have a little interest of the using outlook of nodejs, since it may enlarge its reputation of good quality.The two least powerful stakeholders are the competitors and the individual users. However, competitors are quite interest in this project, since they are rivals, who should keep a closed eye on each other.

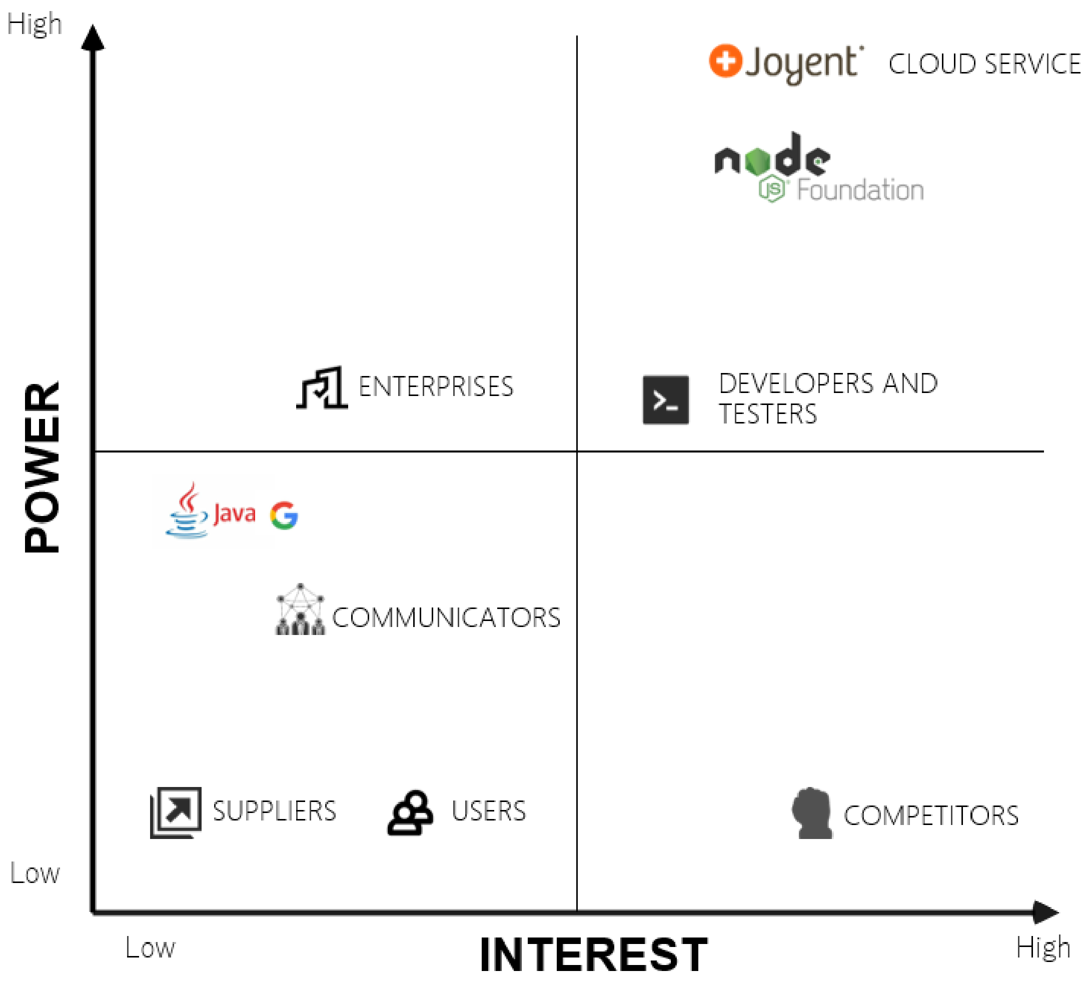


Figure 2: Power-Interest grid for the stakeholders of Nodejs

Context view

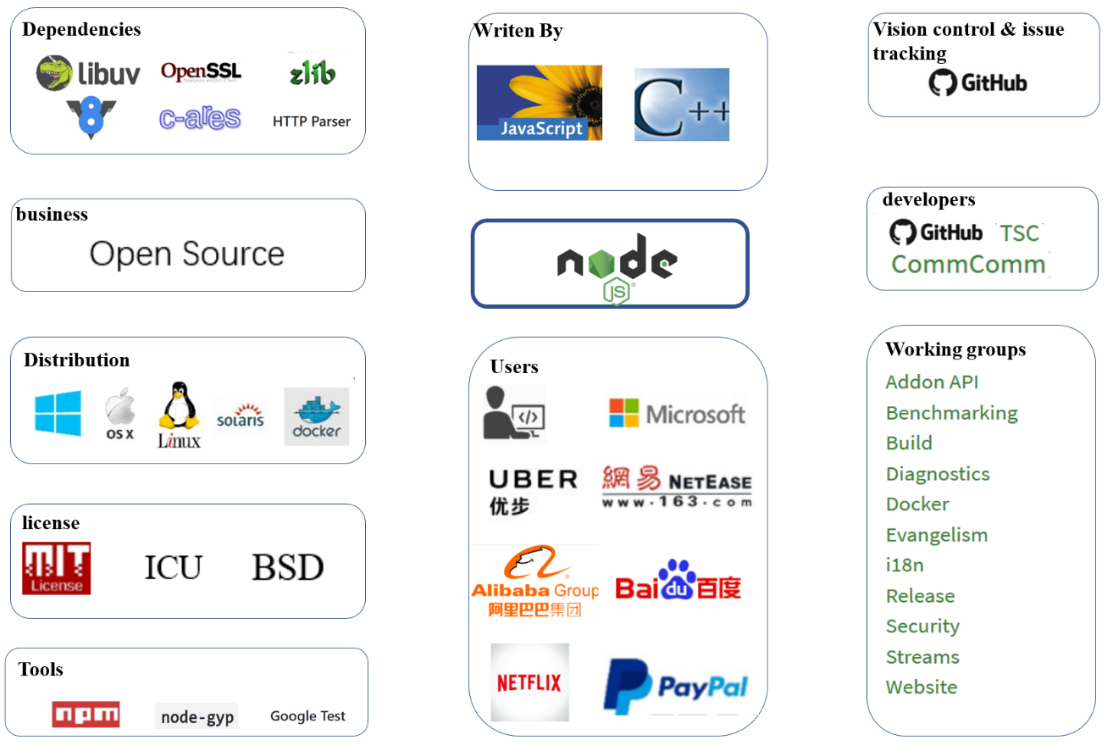
In this section we will discuss the context view describes relationships, dependencies, and interactions between the system and its environment—the people, systems, and external entities with which it interacts.

System scope and responsibilities:

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It was designed to build scalable network applications. Node.js is an event-driven I/O server-side JavaScript environment. Based on Google's V8 engine, it executes Javascript very quickly and performs very well. It is mainly meant to provide developers with the foundations for common server-side functionalities such as file system I/O operations, database access, computer networking.

Environment:

We have listed the environment of node.js which we discuss sequentially. An overview of all the entities and their relations to Node.js can be found in picture below.



Programming languages:

Node.js is almost written in JavaScript Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Meanwhile this engine is itself written in C++, so Node.js unavoidable interact with this engine are also written in C++.

Dependencies:

There are a number of libraries or products that Node.js explicitly depends upon. It offers

only the most basic necessities for a product.
libuv:A C library that is used to abstract non-blocking I/O operations to a consistent interface across all supported platforms. It provides mechanisms to handle file system, DNS, network, child processes, pipes, signal handling, polling and streaming. It also includes a thread pool for offloading work for some things that can't be done asynchronously at the operating system level.
c-ares:A C library for asynchronous DNS requests.
openssl: OpenSSL is used extensively in both the tls and crypto modules. It provides battle-tested implementations of many cryptographic functions that the modern web relies on for security.
http parser: HTTP parsing is handled by a lightweight C library called http-parser. It is designed to not make any syscalls or allocations, so it has a very small per-request memory footprint. In short, it parses HTTP requests and responses.
v8: The V8 library provides Node.js with a JavaScript engine, which Node.js controls via the V8 C++ API. V8 is maintained by Google, for use in Chrome.
zlib: a library used for (de)compression.

Tools:

Besides the dependencies, there are also many parts that makes Node.js more powerful. They just add flowers to embroidery but these are not parts that Node.js must have. They act as additional features.
npm: Node.js is all about modularity, and with that comes the need for a quality package manager; for this purpose, npm was made. With npm comes the largest selection of community-created packages of any programming ecosystem, which makes building Node.js apps quick and easy. It offers access to a multitude of open source libraries.
gyp: a build system to build those parts of Node.js and its dependencies that require compilation. The build system is handled by gyp, a python-based project generator copied from V8. It can generate project files for use with build systems across many platforms.
gtest:A unit testing suite for C and C++ code. It allows testing C/C++ without needing an existing node executable to bootstrap from.

Business:

Node.js is open-source, everyone can access its source code, design documents, and content.

Distribution:

Node.js is available on Windows, Mac, Linux, SunOS and Docker. It is open-source, everyone can access its source code, design documents, and content which can be accessed from webpage or packages everywhere. The packagers are responsible for packaging the Node.js code base themselves, so Node.js stresses that any issues people run into should be reported to them. If it turns out to be an issue with Node.js itself, those packagers will contact Node.js to notify them accordingly.

Users:

The users of Node.js can be divided into two subcategories, that is the individual and enterprise. The individual users uses Node.js for study or fun, such as students. They are not intended to make money.
Enterprise are the users who do use Node.js as a tool in their company to help improve their product. Some major companies use Node.js, such as Netflix,PayPal,Uber,Microsoft,Netease, baidu, Alibaba and so on.

Version control & Issue tracking

Node.js is actively being developed on GitHub using Git as its version control system. The same system is also used to track issues, report bugs and discuss features.

License:

The Node license closely follows the ICU, BSD and MIT license.

Developers:

The nodejs/node core GitHub repository is maintained by the Collaborators who are added by the Technical Steering Committee (TSC) on an ongoing basis. Individuals making significant and valuable contributions are made Collaborators and given commit-access to the project. These individuals are identified by the TSC and their nomination is discussed with the existing Collaborators. The project is governed jointly by the Technical Steering Committee (TSC) which is responsible for high-level guidance of the project, and the Community Committee (CommComm) which is responsible for guiding and extending the Node.js community.

Working groups:

Addon API: responsible for maintaining the NAN project and corresponding nan package in npm. The NAN project makes available an abstraction layer for native add-on authors for Node.js, assisting in the writing of code that is compatible with many actively used versions of Node.js, V8 and libuv.
Bench marking:gain consensus on an agreed set of benchmarks that can be used to:track and evangelize performance gains made between Node.js releases and avoid performance regressions between releases.
Diagnostics:The Diagnostics Working Group's purpose is to surface a set of comprehensive, documented, and extensible diagnostic interfaces for use by Node.js tools and JavaScript VMs.
Docker:The Docker Working Group's purpose is to build, maintain, and improve official Docker images for the Node.js project

Evangelism:The Evangelism Working Group promotes the accomplishments of Node.js and lets the community know how they can get involved.

Release:The Release Working Group manages the release process for Node.js.

Security:the Security Working Group manages all aspects and processes linked to Node.js security.

i18n: The i18n Working Groups handle more than just translations. They are endpoints for community members to collaborate with each other in their language of choice.

Streams:The Streams Working Group is dedicated to the support and improvement of the Streams API as used in Node.js and the npm ecosystem.

WebSite: The Website Working Group's purpose is to build and maintain a public website for the Node.js project.

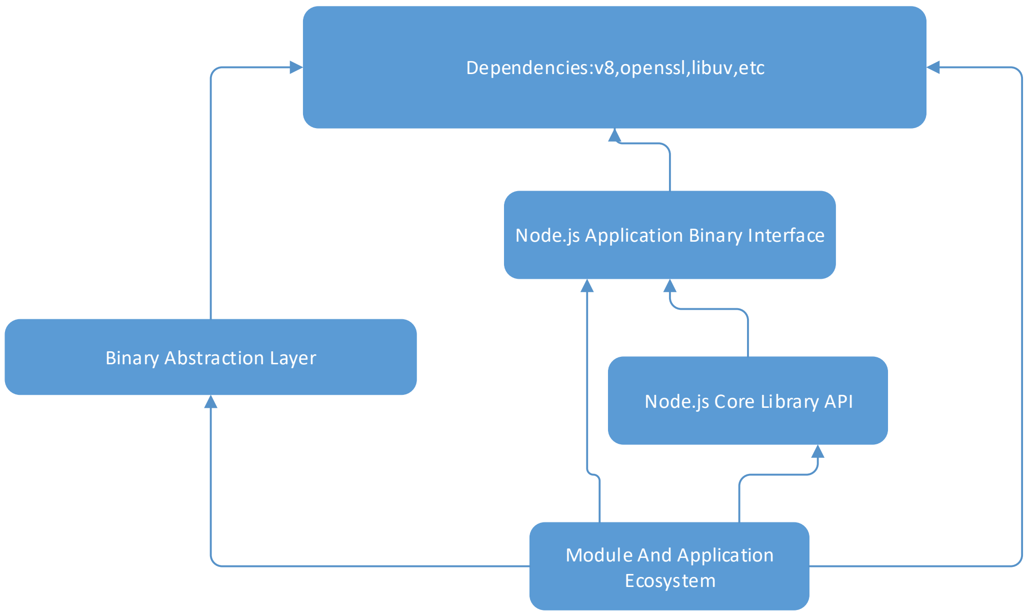
Build:TheBuild Working Group's purpose is to create and maintain a distributed automation infrastructure.de.js community.

Development View

In the development view part, it will focus on how the architecture of node.js enable itself to develop continuously and the aspects of development developers will care. Development views combine the different parts including building, testing, maintaining and improving the software and show it to the relevant stakeholders.

Component organization

Basically, node.js is not only a product but also a platform offering service for other developers to build various applications. As a result, the basic level in the architecture have influences in developing software by using node.js.


Dependencies:v8,openssl,libuv,etc
Node.js Application Binary Interface
Binary Abstraction Layer
Node.js Core Library API
Module And Application Ecosystem
Figure: Module Organization

- In the Dependencies, we need to figure out the basic nature of node.js.Node.js is a platform built on the Chrome JavaScript runtime.Node.js is an event-driven I/O server-side JavaScript environment, based on Google's V8 engine, which executes JavaScript very fast and performs very well.
- The Application Binary Interface (referred to as the ABI) is a new part of Node.js, which is built in recent years.
- The Node.js Core Library includes plenty of JavaScript files that help developers build software easier. It offers all kinds of function for users to build in various usages like network connections, server building and so on. In case of some APIs change without the developers' notice and operations, part of the API are hided and marked as "internal function".
- The Module and Application Ecosystem contains all applications built by using Node.js. The figure shows that developers can connect to all of the Node.js' level in the process of developing in theory. Mostly, the applications just connect to the Node.js Core Library in the real development.

Design Patterns

We emphasized how simple patterns such as factory can greatly improve the flexibility of our code and how with Proxy, Decorator, and Adapter we can manipulate, extend, and adapt the interface of existing objects. Strategy, State, and Template, instead, have shown us how to split a bigger algorithm into its static and variable parts, allowing us to improve the code reuse and extensibility of our components. By learning the Middleware pattern, we are now able to process our data using a simple, extensible, and elegant paradigm. Finally, the Command pattern provided us with a simple abstraction to make any operation more flexible and powerful.//引用 Node.js Design Patterns, 2nd Edition: Chapter 6: Design Patterns: Summary.

Factory

It is a generic interface for creating objects. In fact, invoking a factory, instead of directly creating a new object from a prototype using the new operator or *Object.create()*, is so much more convenient and flexible in several respects. It can also be used as an encapsulation mechanism, thanks to closures.

Revealing constructor

The revealing constructor pattern is a relatively new pattern that is gaining traction in the Node.js community and in JavaScript, especially because it's used within some core libraries such as *Promise*.

Proxy

A proxy is an object that controls access to another object, called a subject. The proxy and the subject have an identical interface and this allows us to transparently swap one for the other; in fact, the alternative name for this pattern is surrogate. A proxy intercepts all or some of the operations that are meant to be executed on the subject, augmenting or complementing their behavior.

Decorator

Decorator is a structural pattern that consists of dynamically augmenting the behavior of an existing object. It's different from classical inheritance, because the behavior is not added to all the objects of the same class, but only to the instances that are explicitly decorated. Implementation-wise, it is very similar to the Proxy pattern, but instead of enhancing or modifying the behavior of the existing interface of an object, it augments it with new functionalities, as described in the following figure:

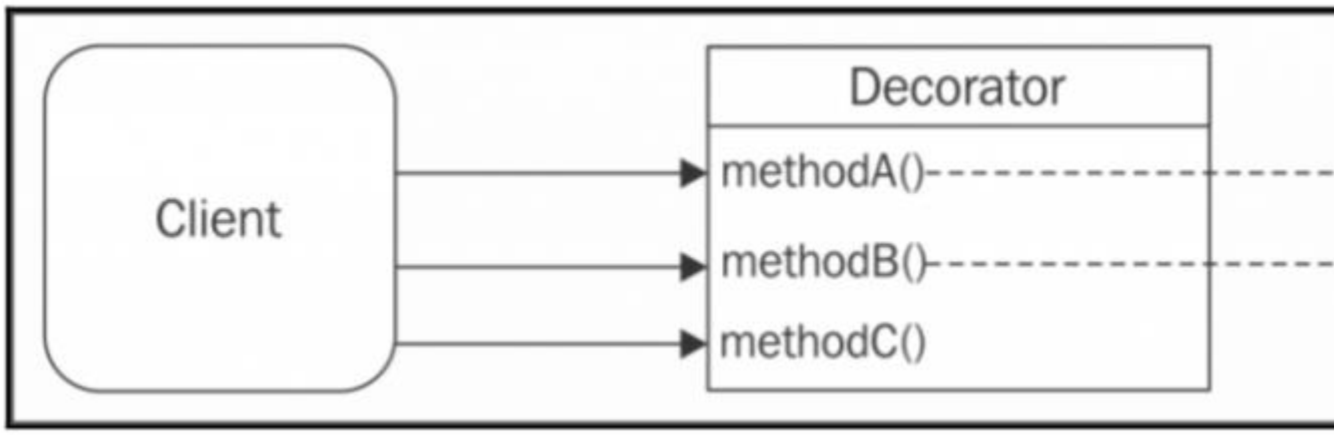


Figure: decorator//引用 Node.jsDesign Patterns, 2nd Edition: Chapter 6: Design Patterns: Decorator

Adapter

The Adapter pattern allows us to access the functionality of an object using a different interface. As the name suggests, it adapts an object so that it can be used by components expecting a different interface.The following diagram clarifies the situation:

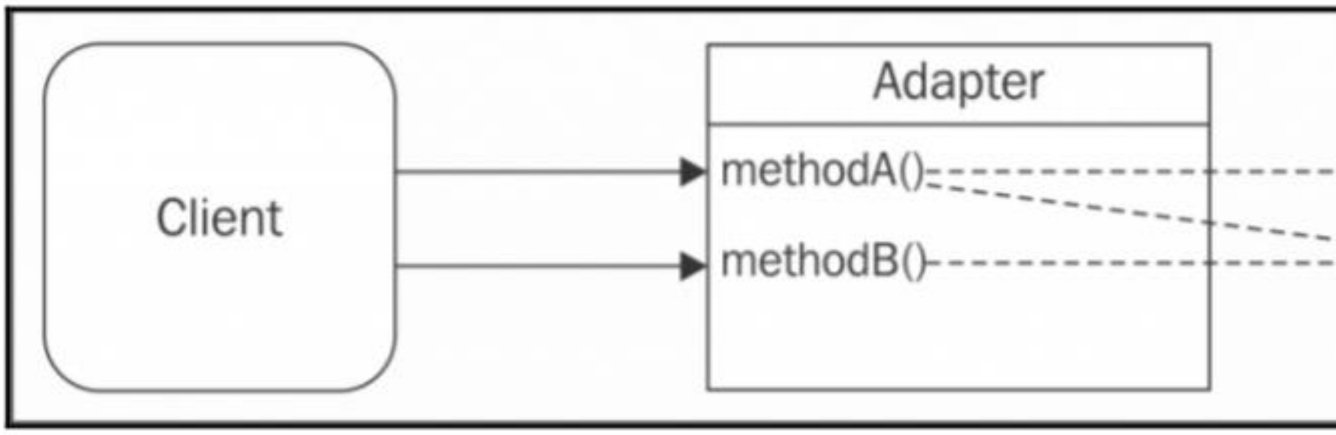


Figure: decorator//引用 Node.jsDesign Patterns, 2nd Edition: Chapter 6: Design Patterns: Adapter.

Strategy

The Strategy pattern enables an object, called the Context, to support variations in its logic by extracting the variable parts into separate, interchangeable objects called Strategies. The context implements the common logic of a family of algorithms, while a strategy implements the mutable parts, allowing the context to adapt its behavior depending on different factors such as an input value, a system configuration, or user preferences. The strategies are usually part of a family of solutions and all of them implement the same interface, which is the one that is expected by the context.

State

State is a variation of the Strategy pattern where the strategy changes depending on the state of the context. A strategy can be selected based on different variables such as user preferences, a configuration parameter, and the input provided, and once this selection is done, the strategy stays unchanged for the rest of the lifespan of the context. Instead, in the State pattern, the strategy (also called state in this circumstance) is dynamic and can change during the lifetime of the context, thus allowing its behavior to adapt depending on its internal state

Template

It also has a lot in common with the Strategy pattern. Template consists of defining an abstract pseudo class that represents the skeleton of an algorithm, where some of its steps are left undefined. Subclasses can then fill the gaps in the algorithm by implementing the missing steps, called template methods. The intent of this pattern is to make it possible to define a family of classes that are all variations of a similar algorithm.

Middleware

One of the most distinctive patterns in Node.js is definitely middleware. Unfortunately, it's also one of the most confusing for the inexperienced, especially for developers coming from the enterprise programming world. The reason for the disorientation is probably connected to the meaning of the term middleware, which in enterprise architecture jargon represents the various software suites that help to abstract lower-level mechanisms such as OS APIs, network communications, memory management, and so on, allowing the developer to focus only on the business case of the application.

Command

Another design pattern with huge importance in Node.js is Command. In its most generic definition, we can consider a Command as any object that encapsulates all the information necessary to perform an action at a later time. So, instead of invoking a method or a function directly, we create an object representing the intention to perform such an invocation; it will then be the responsibility of another component to materialize the intent, transforming it into an actual action. Traditionally, this pattern is built around four major components

CodelineOrganization

This part is a simple overview of the source code structure of Node.js, which significantly simplify the development.The structure of Node.js' code is as follows:

Directory	Description
benchmark	It contains the code and data for benchmarking and measuring the performance ofdifferent Node.jsimplementations. The benchmarks are classified into 25 directories depending on the subsystem they benchmark. It also includes a miscellaneous directory for benchmarks thatdo not clearly fit in one of the predefined categories.
deps	It contains the source code of the third party component that Node.js depends on.
doc	It contains all the documentation for Node.js.
lib	It contains the JavaScript modules used in Node.js.
src	It contains the bindings that expose the C/
test	It consists of the code used to test Node.js.
tools	It contains additional tools that are useful for development with Node.js.