Isaac

[25s] Hi, my name is Isaac, [I'm Jay], [I'm Raph], [and I'm Stefan]. In this video, we'll be taking a look at three different software systems that provide cross-platform networking and web solutions for developers. For each of these systems, we'll analyze two significant design decisions and take a look at why they were made, as well as their implications on the system. The three software systems we'll be analyzing are Twisted, Node.js and Flask.

[52s] Our reference system is Twisted. Twisted is an event-driven networking engine built in Python. An event-driven engine involves a reactor that receives a stream of events from an event queue. Events, such as a device receiving a datagram or an object colliding with another object in a videogame, are binded to functions that perform relevant tasks, such as the system saving the contents of a datagram, or a character taking damage in a videogame. Event-driven programming differs from conventional imperative programming in that the main reactor loop is not in control of what happens in the program. Events decide what happens, and the main reactor simply calls functions to respond to the occurence of events. In this sense, an analogy can be drawn between the reactor, and the task scheduler on your device. Some popular applications that use Twisted are Twitch, The Apple Calendar, and the Ubuntu One file-hosting service.

[36s] Node.js is a JavaScript runtime environment that runs on Chrome's V8 engine and executes JavaScript code outside of a web browser. It can be used for both front-end and back-end services. Similar to Twisted, node uses an event-driven model that allows functions to handle events. Where these two systems differ, is that they use different programming languages, and provide different levels of flexibility among their utilities. At the end of the day, both services can accomplish the same tasks with some tweaking. Some popular applications that use Node.js are PayPal, Netflix, and LinkedIn.

[30s] Flask, unlike Twisted and Node.js, is a microframework for web development, and is written in Python. Flask differs from the other two systems in that it is designed to be lightweight and minimal, meaning it can be used from the get-go with no additional dependencies. Of course, like all good microframeworks, it is highly expandable with libraries and external tools for accomplishing a large variety of tasks that it does not natively support. Some popular applications that use Flask are Airbnb, Reddit, and Uber.

Jay

Twisted is an event-driven networking engine written in Python. In event-driven programming paradigm, program flow is determined by external events; characterized by an event loop and the use of callbacks to trigger actions when events happen. Twisted implements the reactor design pattern, which describes demultiplexing and dispatching events from multiple sources to their handlers in a single-threaded environment.

Twisted allows the program to make progress when it can without the use of additional threads.

Twisted hence counters the producer/consumer problem because the programmer doesn't have to worry about thread safety here which additionally makes it great for applications that require mutable data.

Twisted is an engine for producing scalable, cross-platform network servers and clients, designed to be very flexible, and it lets us write powerful clients, at the cost of a few additional layers in the way to writing said clients. Twisted comes with client and server implementations for all of its protocols, as well as utilities that make it easy to configure and deploy production-grade Twisted applications from the command line.

Why not use pre-existing applications?

Those server implementations have networking code written from scratch, typically in C, with application code coupled directly to the networking layer. This makes them very difficult to use as libraries.

The clients and servers in Twisted are written in Python using a consistent interface. This makes it is easy to write new clients and servers. Twisted's implementation makes it easy to share application logic between protocols.

Twisted allows for easy client-server code sharing.

Raph

Node.js is a JavaScript runtime environment that brings event-driven programming to web servers. To achieve this, a significant design decision was to build Node.js around an event-driven architecture. The main goal of this architecture is to allow Node to handle a large amount of requests asynchronously, through the use of an event-loop and a worker pool - a set of spawning background worker threads. Node's event loop is single-threaded, and accepts events triggered directly from the application level. When an event needs to be operated on asynchronously at a later time, the event-loop delegates this event to a new worker thread dedicated to this event. When the event triggers, the background worker thread can then operate on it (via callback chains) and finally notify the main event-loop that the event is ready to be dispatched to its correct handler at the application level.

(Showing M3 C&C diagram for Node.js here)

The event-driven architecture allows Node to avoid the Producer/consumer problem, where contention for resources blocks the processing of a system.

(Pt.2) Node.js greatly leverages the V8 JavaScript runtime engine, and thus inherits many of its capabilities. By deciding to build on top of V8, Node.js becomes lightweight and performant, as the large majority of apps and functionality is built natively in JavaScript. Therefore, Node largely relies on JavaScript and avoids having to rely on too many external dependencies outside of the language it is primarily focused on.

Node's battle-tested libraries can be easily attached on through its dependency package manager, npm. This integration allows Node to offer a large (And still growing) amount of resources to developers, who can tackle their development problems through the help of Node. With libraries built for Node.js, implementations - such as client-server models - can be easily realized by eager developers.

Stefan

As Isaac mentioned, Flask is a Micro-web framework. This framework is an abstract architecture that essentially moves the joy of solving problems to other vendors that have already done so. However, which solution to the problem is best? This answer is left to the user to decide on implementation.

When it comes to solving the Producer/Consumer problem, as mentioned, there is no built-in solution that explicitly solves it. Flask offers only a starting point. So, say the user wants to directly build a solution to their problem in Flask’s source code. Editing the source goes against the framework objective. Frameworks are tools that helps construction and therefore the process of construction should not modify the framework. Thus, Flask implements a highly dense, non-modular, and uninterpretable design to prevent people from changing the framework. As seen in the image, there is exceptionally high coupling.

Instead of letting users edit the framework, Flask offers extensive support to many non-native plugins. It is entirely possible to use Flask and add twisted or node.js as a plugin and therefore take their solution to handle the shared resources problem.

The next problem has a similar solution; if you need to implement more languages, simply find a library that supports different languages and protocols.

Although this seems like a very abstract framework that offers no solutions, that is their design objective. Flask lets the user make significant architectural decisions for the projects. It makes sure to support their choices and not get in the way.