翻訳トライアル実践講座　第10回

＜問題１＞　全文を訳して提出してください。

How should CIOs respond to the rise of Shadow IT?

Cloud computing now enables a seamless digital working life both inside and outside of the office. Working from home, sharing and storing data, remote capabilities and Bring Your Own Device (BYOD) have all been facilitated by the accessibility and ease of connectivity to the cloud. In fact, some reports have even suggested that cloud technologies are making physical offices obsolete altogether.

However, with the average employee now very comfortable with the process of acquiring and using cloud services, the traditional role of the IT department – as a selector and purchaser of workplace technology – is coming under threat.

IT purchased without the permission, control or knowledge of an organisation's IT department is known as 'Shadow IT'. A decade ago, Shadow IT spending was limited to basic computer accessories or boxed software. But in today's market, vast sums are being spent by unauthorised employees on cloud-based software and services.

For example, it is increasingly common for individuals or teams to purchase their own cloud storage solutions or direct mail platforms to replace existing internal tools. In many ways, this is only natural. For a digitally savvy employee in, say, HR or marketing, it will be second nature to get the tool they need with just a few clicks (and perhaps a company credit card). And, you might ask, if these tools are helping to drive productivity, why does it even matter?

The answer is that, as an isolated incident, it might not. However as a wider trend, Shadow IT can quickly become a significant issue for businesses. It's been predicted that these types of purchases make up a colossal 40 percent of IT spending – a worryingly high figure considering all of this IT activity is happening outside of an IT department's control.

After all, with any purchase, buying individually rather than in bulk results in higher costs. For cloud computing this is very much the case, especially if the cloud services purchased are supplementing existing solutions provided by the company.

Furthermore, using cloud services that have not been approved by IT can put business data at risk. By using third party cloud solutions without reading the small print, employees can't be sure about what is happening to their data once it leaves their sight.

＜問題 2＞　全文を訳して提出してください。

10 TIPS FOR CODING WITH NODE.JS #3: HOW TO KNOW WHEN (NOT) TO THROW

In JavaScript, throwing is a destructive action. It's the sledgehammer of error propagation.

In this post, we'll examine scenarios where throwing is appropriate, when not to throw, alternatives to throwing, and best practices for exception handling. This article is quite detailed, you might want to grab a cup of heated liquid mixed with some kind of organic material (such as beans or leaves) and settle into a comfortable chair first.

Types of errors

Let's group errors into two broad categories: developer errors and operational errors. Developer errors are bugs in the software: a mistyped variable name, incorrect input, syntax errors and so forth.

Developer errors also include error states that occur from mishandling operational errors. Operational errors are generally predictable issues that occur beyond the control of the program. For instance, network errors or invalid user input.

Native throwing

In many cases, the JavaScript engine will throw when it comes across a developer error. These errors should not be caught, the app needs to fail. In development this allows us to immediately weed out bugs and in production developers can be alerted whilst the process is restarted to retain maximum uptime.

When to throw operation errors

Most operational errors are manageable and should be expected.

For example a dropped connection can be logged and retried, or in some cases replaced with cached content. There’s rarely grounds for throwing due to an operational error.

But there is a subclass of operational errors that could merit a throw. This class of errors is to do with environment misconfiguration.

For instance if a connection is dropping because authentication details are incorrect, we could throw to exit the process (after logging the error).

Almost all configuration happens during initialization, so if we’re throwing post-initialization it could be an over-aggressive approach to an error.

以上