Linux SEのお仕事

トラブル三題噺
～終電・増殖・コミュニケーション

文：株式会社ビーブレイクシステムズ　鹿取裕樹

最近ではLinuxが業務システムに浸透しており、読者の皆さんの中にも、「お仕事」としてLinuxを使ったシステム開発に携わっておられる方もいると思われる。Linuxの適用範囲が基幹業務へも広がり、既存システムと連携したシステムをLinux上に構築したり、ERP（Enterprise Resource Planning）システムをLinux上で稼動させるという例も数多く聞かれる。

この連載では、筆者が経験したLinuxを用いた業務システム構築の実例を紹介し、その中で発生したトラブルとそれに対してどう対応したかを中心に紹介していきたい。

Linux採用の理由

SE no Oshigoto

Linuxを使用することのメリット／デメリットはさまざまな点が論じられているが、顧客にとって一番わかりやすいのが初期導入コストが低いことである。筆者が開発に携わったシステム（以下、本システムと呼ぶことにしよう）が構築されることになった理由も、この初期導入コストの低さを顧客が評価したためであった。

もちろん顧客も安さだけで単純に判断するわけではなく、OSとしての安定性や稼動するソフトウェアなどを総合的に判断する。これらの点でもLinuxのメリットは大きい。Linuxは低価格のサーバで安定稼動し、PostgreSQL、Apache、Tomcatといった信頼性の高いオープンソースソフトウェアを稼動させることができるからだ。

これに対し、顧客がLinuxの導入に感じるデメリットは、運用するスキルがないということである。Windowsに慣れている顧客にとって、やはりLinuxはとっつきにくく、不安に感じるものである。

本システムは、「IBMのeServer iSeriesサーバ上に4GL（第4世代言語）で構築されているBtoBシステムをLinux＋Javaで再構築する」というものである。「旧システムでは4GLの保守料が負担である」、「iSeriesサーバの空きディスク容量が少なくなったが、追加に多額のコストがかかる」という主にコスト面の不満が大きかった。加えて、基幹の会計／在庫システムと連動しており、外部に公開しているシステムであるため、信頼性も重視される。

そこでLinuxのコストの低さ、信頼性の高さというメリットが評価され、採用に至ったわけだ。Linuxの運用スキルへの顧客の不安は、開発会社に運用を委託することで解消された。

■システムの概要

簡単に本システムの概要を紹介しておこう（**図1**も参照のこと）。

本システムは商社のBtoBシステムで、図1の中心に描かれているPCサーバ上に構築されている。特徴としては、以下の3つが挙げられる。

1つめは多国語対応。買い手企業は日本、中国、アメリカの企業であるため、本システムでは日本語、中国語、英語の3カ国語に対応しており、画面表示からデータ登録までを3カ国語で行うことができる。

2つめは基幹システムとの連携である。基幹システムはIBMのiSeries上に構築されており、在庫管理、販売管理、会計の機能がある。本システムは、基幹システムの商品マスタの参照、在庫

の引当、受注伝票の登録といった基幹システムとの連携処理を行う。

3つめは提携先企業への受注代行である。提携先企業は中国の商社で、ここが扱う商品データを本システムに取り込み、本システムに登録された受注データを提携先企業に送信している。なお、両者の間はVPNで結ばれている。

■プロジェクトの流れ

開発プロセスとしてはウォーターフォール型が採用された。プログラム言語はJavaを用いていたが、マネジメントを担当する会社に反復型開発のノウハウがなかったため、従来どおりのプロセスがとられることになったのだ。ウォーターフォール型の開発プロセスでは、開発プロジェクトは**図2**のような流れとなる。

プロジェクトにはさまざまなリスクがあり、何が起こるかわからないものである。本プロジェクトも例外ではなく、ざっと挙げるだけでも次のようなトラブルが発生した。

- **iSeriesへの問い合わせSQLのパフォーマンスが悪い**
- **Javaプロセスが増え続け、システムが停止する**
- **承認された仕様の変更**
- **iSeriesとの連携APIの致命的なバグによるAPI変更**
- **在庫引当連携機能のバグによる在庫数のずれ**
- **提携先企業のシステム構築の遅れ**

読者の皆さんは「このようなトラブルが起きないようにするのがプロなのだ」と思われるだろうが、そうはいかないのもまた現実。今回は、上記のうち最初の3つのトラブルについてお話することにしよう。

トラブルその1

SE no Oshigoto

開発が進む中で1つのトラブルが発生した。それはiSeriesサーバ上のDBに問い合わせたSQLの結果が返ってくるのが非常に遅いというものである。一人で悶々と試行錯誤してみたが、どうにもうまくいかないのだ。

顧客からは検索結果が3秒以内で返ってくることを求められていた。しかし、そのSQLは数個のテーブルを結合するSELECT文だったが、なんと数分かかっても結果が返ってこないのであった。問題外の遅さである。しかも、これらの検索ができないと開発を先に進められないため、早期に修正を行う必要があった。そこで、私の担当範囲ではあったが全プロジェクトメンバーで対応にあたることとなった。

「できました！」　私の隣の新人プログラマーが大声をあげる。夜中の12時は回っていたであろう。メンバーはビクっと反応し、時計を見て「まだ終電がある」とホっとしたのも束の間、問題のものとは別のSQLを流していることに気づき、「このSQLじゃないぞ！！」と眼い目をこすりながら対応した。

翌日、DBの最適化を行うしかないと判断したプロジェクトリーダーは、顧客企業の担当者にDBベンダーにサポートを依頼しましょうと提案したが、「予算がない」とあっさり断られてし

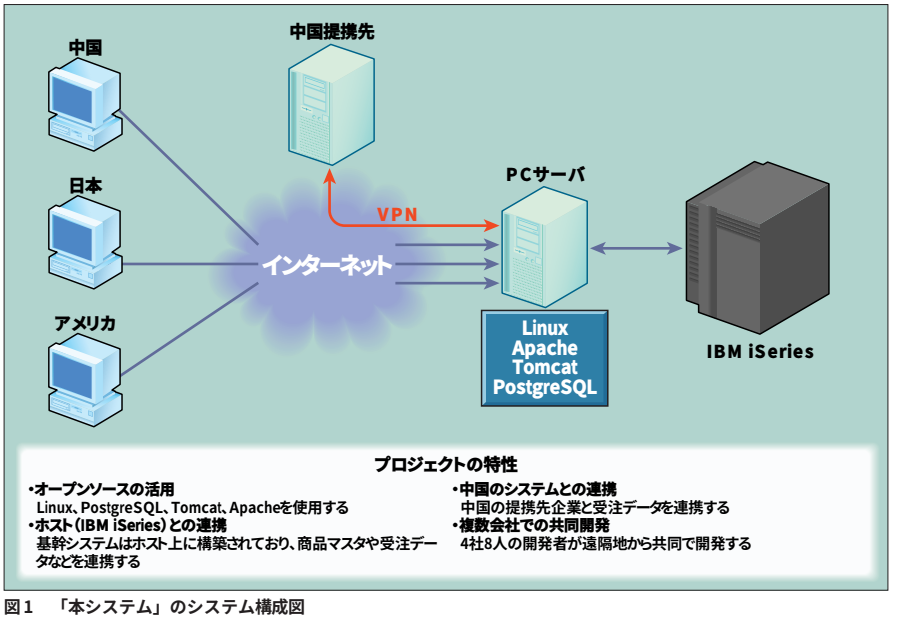

図1　「本システム」のシステム構成図

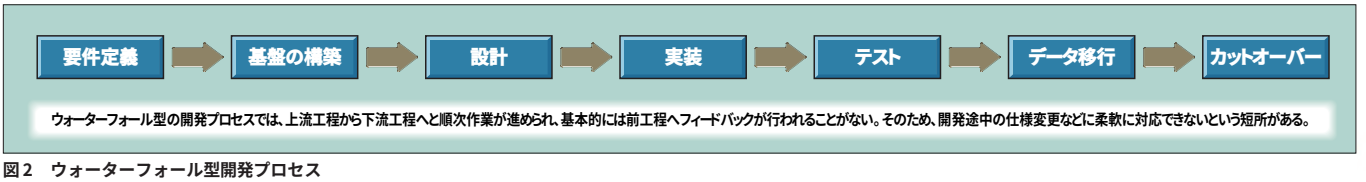

図2　ウォーターフォール型開発プロセス

まった。連日の徹夜で反論する気力もなく、その夜も地道なSQLの解析が続いた。

食事は、常にメンバー全員で定食の出る居酒屋に行っていたのだが、そのときの話題といえばプロジェクト進捗状況とSQLと帰宅時間だけだったと記憶している。

結局、SQLの書き方をいろいろと変更して試行錯誤したところ、結合の順序によって結果が返ってくる時間に違いがあるということがわかった。ドキュメントによると、DBシステム側でSQLを解析して結合の順序に関わらず処理してくれるはずなのだが、その機能がうまく働いていないようであった。

そこでレスポンスが遅いSQLを洗い出し、それぞれについて一番早い結合の順序を見つけるという地道な作業を繰り返した。多いものでは10個のテーブルを結合しており、その結合の順序を順番に入れ替えて検証するのには、かなりの時間がかかった。

この頃になると、皆、疲れているというより、「うすーく」一人笑いをして自分の世界に入っていた。

しかしながら、さらに悪いことに、検索画面には複数の条件を指定できるようになっていて、それらの条件間を「and」でつなぐか「or」でつなぐかをユーザーが選択できるようになっていた。このため、ある条件が指定されていてある条件が指定されていない場合や、各条件間を「and」でつなぐ場合と「or」でつなぐ場合など、非常に多くのパターンを検証しなくてはならなくなった。

「できたぞー」とプロジェクトリーダーが、皆の目を見ながら軽く叫ぶ。だれも声を出すことはできなかった。うつむきかげんに微笑むのが精一杯だった。

かなり原始的な方法をとることになったが、タイトな期間の中で数日を割いて検証し、なんとかどの検索処理でも3秒以内に検索結果を表示させることができるようになったのだ。「SQL万歳！」と心のなかで大きく叫んだ。

今考えてみると、

- **DBベンダーへのサポートを依頼することを顧客に対して強く交渉する**
- **DBベンダーへのサポート依頼が許可されない場合は、ユーザーの利便性を考え、検索スピードを上げるために検索方法を変えるように交渉する**

などの対応が考えられる。対症療法のために時間を費やすのではなく、機能の本質的な目的を見極めた対応をしていくべきであった。

トラブルその2

SE no Oshigoto

カットオーバーが近づいた頃、新たなトラブルが発生した。

それはLinux上でJavaのプロセスが増え続けるというものである。統合テストに入りシステムを連続稼動させ、多数のトランザクションを実行するようになってこのトラブルが明らかになった。

具体的には、システムを連続稼動させている状態でLinuxのpsコマンドを実行するとJavaのプロセスが大量に表示されていたのである。さらに、このJavaプロセスが一定数に達するとTomcatでエラーが発生し、システムが停止するということも判明した。

BtoBシステムは外部に公開され長期間連続稼動させるため、これは致命的な問題である。

トラブルが発生した際は、まずその現象を再現させ、次にその原因を分析する、という手順で対応する。

「再現すればすぐわかるでしょう」とプロジェクトリーダー。「そうですね」と私は答えた。いつもどおりのトラブル発生時のやり取りだ。しかし、トラブルを再現することはならず、さっそく調査に入った。

我々は当初、Javaのプロセスは Tomcatのスレッドに対応しているものであり、Javaのプロセスが増えるのはTomcatのスレッドが増え続けているためだと考えた。そのため、Java ServletやTomcatの仕様を読み、スレッドの数を制限することができないかを調査した。さらにネットでの検索を行い、Tomcatのメーリングリストのログや掲示板をあたった。

しかし、使用している環境ではすでにスレッドの最大値が設定してあり、掲示板やメーリングリストのログにもここで設定されている最大値以上にスレッドが増えるという問題は報告されていなかった。「ついてないなぁ」としか言えない状況だ。偶発的な出来事として片づけたいが、そうもいかないのだ。システムが落ちてしまうのだから。動かないコンピュータにしたくないと強く願った。

そこで、ほかに原因があると考え、システムの機能を1つ1つ実行しながらpsコマンドの結果を監視し、Javaプロセスが残る機能を洗い出しにかかった。

その結果、ホストへの接続がクローズされていないという原因を探り当てることができた。

このケースではネットの情報が直接

にはトラブル解決につながらなかったが、オープンソースのメリットである公開された情報の多さには非常に助けられた。しかし、言語によってこのメリットを享受できるかどうかに大きな差がある。日本語と英語とでは情報量に非常に大きな差があるのである。オープンソースソフトウェアを利用する場合、その活用は英語の読み書きができるか否かが大きく影響することを実感した。

トラブルその3

SE no Oshigoto

要件定義は今も昔もユーザー、開発者の双方を悩ます問題である。本システムでも要求定義が上手くいったとはいえず、一度は顧客から承認を得て判を押してもらった仕様だが、案の定、変更要求が出た。

「こっちがお金払っているのだから、やって下さいよ」、しかも対応するのは当たり前で「これが直らないと金を払わない」とまで言われたのである。よくあることだ。

すでに承認され確定した仕様なのだから、理屈の上では変更を受け入れる必要はないのだが、現実的にはそうもいかない。しかたなく、変更要求に優先度をつけ、スケジュール／工数を考慮し、できる限りの対応をするということになった。

今回のケースをあとで分析してみると、以下の原因が考えられる。

1. **顧客に対してシステム構築のプロセスを十分説明できておらず、仕様を変更することのリスクが理解されていなかった**
2. **顧客は仕様確定の段階では自分達がやりたいことを完全に描ききれていなかった**
3. **顧客が言うがままに仕様を決めており、開発者側から有効な提案ができていなかった**

このプロジェクトはキックオフが十分に行われずに開始されてしまった。そのため、構築のプロセスが顧客に十分に理解されておらず、仕様確定の重要性や確定された仕様を変更することのリスクが認知されていなかった。

プロジェクトの早い段階である仕様確定時点では、顧客はシステムが実際にどうなるかをイメージできていない。したがって、あとで変えることができるという認識がある。また、そもそも仕様確定の段階ですべてを決めるというのは不可能なのである。

このプロジェクトではこの問題に対応し、システムが実際にどうなるかというイメージをはっきりさせるために、仕様をHTMLで表して実際の画面の動きに合わせて説明した。しかし、その仕様はほとんど顧客の要望のままに作られたもので、ユーザビリティや運用までを網羅した全体的な観点での検討が十分になされていなかった。

顧客は、開発者がシステム構築のノウハウを生かし、顧客自身が考えられる仕様以上のものを作りあげるはずだ、というプロとしての付加価値を期待している。これに応えていくために、こうした全体的な観点での検討と提案をしていかなくてはならないと感じている。また、仕様の変更は必ず起きるという前提に立ち、大きな仕様変更が起きないような工夫をする必要がある。

具体的には全体に関わる部分、リスクの大きな部分を先に開発し、顧客に確認してもらい確定させていくという反復型開発プロセスを取り入れていくことも検討すべきであろう。

一番苦労するのは人間

SE no Oshigoto

技術的なトラブルも、もちろん厄介なものだが、相手がコンピュータであるだけに理屈どおりに対処すれば解決は可能である。人間相手にはそうはいかない。

プロジェクトにはお客様やプロジェクトメンバーといった直接的に関係する人のほかに、お客様のお客様、お客様の上司、自社の上司、営業担当者などの間接的に関係する人がいる。それぞれの思惑が異なり、何かを伝えるときにも、自分では思っていなかったように受け取られることがある。

このようなコミュニケーション上の齟齬が要件定義におけるユーザーと開発者との間、要件定義担当者と実装担当者との間、プロジェクトマネージャーと開発者との間、ユーザーとその上司との間で少しずつ積み重なって最終的には大きな乖離となることがある。

コミュニケーションを助けるために、UMLなどさまざまなツールがあるが、その使い方も含めて、やはり最終的にはコミュニケーションを行う人間しだいということになる。

ソフトウェア開発は、製造業と比較して前近代的と言われている。もちろん製造業と条件が異なる点は多いが、見習うべき点も多いと感じる。

筆者は、開発に携わる人々は工学的なアプローチをとるなどして、効率的な開発を行え、ユーザー／開発者の双方が幸せになれる状況となるよう努力していくべきだと考えている。