「ソースコード天国」実現に向けて

Toward realizing "Source Code Paradise"

大和 正武*

あらまし ソースコード読解の練達者個人あるいはチームが能力を自在に発揮できる環境「ソースコード天国」の実現を目指している．その第一歩として「読む可能性のあるソースコードを全て，前もってファイルシステムに整頓して配置しておき，ファイルシステムを利用者間で共有する」という方法 (ソースコード事前配置法) を考え，それを勤務先及び顧客部署に実装した．7 年間の利用を通して (1) ソースコードの入手待ち時間によって読解者の集中力が削がれることがなく，(2) 読解補助ツールを効率良く活用でき，(3) 共同作業者へ円滑に読解箇所を伝えることができる，という効果があることを確認した．

1　はじめに

勤務先の会社が提供する GNU/Linux ディストリビューション (Red Hat Enterprise Linux，省略して RHEL) について，顧客より受け付けた様々な技術的質問に回答する，という業務 (ソフトウェア調査業務) に従事している．「想定と異なる動作についてその原因を知りたい」あるいは「見たことの無いメッセージがシステムログに記録されていたのでその意味を説明して欲しい」というのが典型的質問である．顧客は RHEL の利用に豊富な経験を持ち (相対的に) 簡単な疑問やトラブルであれば自身で解決してしまう．筆者のところにやって来るのは，マニュアルなどのドキュメントを読んだり素朴な検索語を用いたインターネット検索では解決できなかった質問である．

対象ソフトウェアについて理解していなければ，そのような質問には回答できない．ドキュメントに記載されていないソフトウェアの詳細について現実的な時間内に理解するにはそのソースコード読む必要がある．オブジェクトコードの生成に用いたコンパイラにバグの疑いがあるといった特別な事情がある場合，あるいはソースコードを入手できない場合を除けばソースコードを読むべきである．RHEL はオープンソースソフトウェア (OSS) から構成されており，会社のロゴなど特殊な部分を除けば，RHEL 自身がオープンソースソフトウェアである．従って筆者のみならず顧客も全てのソースコードを読むことができる．

ソースコード読解が必要となるのはソフトウェア調査業務に限らない．ソフトウェアに関わる様々な業務の一部としてソフトウェア調査業務が含まれている．例として，自社開発したソフトウェアについて不具合の修正や脆弱性へ対応する保守業務，十分にはドキュメント化されていないサードパーティ製のライブラリやフレームワークを活用したソフトウェア開発業務が挙げられる．サードパーティ製のソフトウェアについて保守契約があれば調査を依頼できるが，重大な疑問に関しては調査結果を開発担当者自身で確認したくなるであろう．保守契約を結んでいない場合や，どこも保守サービスを提供していないソフトウェアを使う場合，どんな疑問であれ開発担当者自身で調べる必要がある．

ソフトウェア調査業務に従事して 7 年間，より精度の高い回答をより短い時間で提供するため 2 つの努力をしてきた．一つは回答作成に必要となる以上に積極的にソースコードを読み，ソースコード読解により練達することである．もう一つは練達した自分や同僚がどのようにして読解しているかを内省して，彼等がその能力を自在に発揮できる環境「ソースコード天国」を実現することである．

*Masatake YAMATO, レッドハット株式会社 (consulting-jp@redhat.com)

本稿で紹介する *sources* は，まだ「天国」と呼べるほどの環境には達していないが，同僚からの高い評価を得たシステムである．*sources* は RHEL の構成に深く依存しているが，その背景にある考え方 (ソースコード事前配置法) の応用先は RHEL に限らない．ソースコード読解に関心のある多くの人にとって有益な考え方のはずである．

「読む可能性のあるソースコードを全て，前もってファイルシステムに整頓して配置しておき，ファイルシステムを利用者間で共有する」というのがソースコード事前配置法である．ソースコード事前配置法を実践すれば，ソースコード読解の作業と時間から，目的のソースコードを探したり，ダウンロードしたりする時間と手間を排除できる．ローテクではあるが熟練の技や高度なツールといったものはソースコードの存在が前提であり，ソースコードが無ければ意味が無い．逆にソースコードが存在すると仮定できれば技もツールも真価を発揮できる．

以降，2 章ではソースコード事前配置法の詳細を説明する．3 章では筆者自身の *sources* 利用経験と同僚へのインタビューを元にソースコード事前配置法を実践すると何が良いのか，を説明する．4 章ではソースコード事前配置法を実践した *sources* におけるソースコードの配置を紹介する．5 章はまとめである．

2　ソースコード事前配置法

「読む可能性のあるソースコードを全て，前もってファイルシステムに整頓して配置しておき，ファイルシステムを利用者間で共有する」についてより詳しく説明する．

業務の内容から読解者が扱う可能性のあるソフトウェアのコンポーネントとコンポーネント毎に 1 つ以上あるバージョン群が決まる場合，それらのソースコード全てがソースコード事前配置法の対象となる．これが「読む可能性のあるソースコードを全て」の意味である．「前のバージョンでは問題無く動作していたのに，新しいバージョンを導入したら動かなくなった．なぜか.」といった疑問に対する調査に備えるため，同じソフトウェアであってもバージョンが異なれば，それぞれ別にソースコードを配置する．対象とするソースコードは顧客に提供したバージョンに限定する必要は無い．例えば開発中の最新ソースコードツリーも配置しておくと，開発現場での知見の一部を共有できる．

読解者がソースコードを必要としたときには既に配置されている，というのが「前もって」の意味である．これは現在入手できるソースコードを一度だけ配置すれば良いという意味ではない．業務の内容によって継続的に実施する必要がある．例えば，読解者として，あるソフトウェアの保守担当者を考える．読む可能性のあるソースコードには，過去顧客に出荷したバージョンのものだけでなく，将来出荷するであろうバージョンのものも含まれる．従って，対象ソフトウェアの新しいバージョンがリリースされ次第，すなわち保守の対象になり次第，新しいバージョンのソースコードを配置する．

「ファイルシステム」は必須の要件では無い．配置したソースコードに対してその場所をユニークに指し示す文字列 (ソースコードパス) を提供できれば良い．ただしソースコードパスは利用者が読んで意味のある文字列とする．ソースコードの sha-1 ハッシュ値を用いればユニークな文字列を生成できるが，意味のある文字列にはならない．

後述する「整頓」が適切になされていれば，ファイルシステム上の「パス」をソースコードパスとして提供できる．物理的な配置先を公開せずウェブインターフェイス経由でソースコードにアクセスを許す場合「URL」が「パス」のかわりとなるであろう．ファイルシステムが適切かどうかは読解者の読解にもちいるコンピュータ環境 (読解環境) による．筆者と同僚の読解環境は，RHEL 上で動作するシェルとテキストエディタなのでファイルシステムを選択した．ファイルシステムを選択した

場合の利点については3.2で述べる．一方ウェブインターフェイスの利点として，スマートフォンを含む多様な読解環境を選択できること，行単位でソースコードの場所を指し示すことができること，ハイパーリンクの埋め込み等により，他のデータをソースコードの表示に統合できる点が挙げられる．

「整頓して」という言葉は複数の要件を含意している．最も重要なのは，業務の内容から何が契機でソースコードが必要になるかということを把握し，その契機から目的のソースコードに迅速かつ自然に辿りつけるよう配置することである．例えば，ソフトウェア調査業務において質問中に調査対象の指定にコンポーネントバージョン使われることが多ければ，コンポーネント/バージョンに基づいて分類，配置する．バグトラッキングシステムのバグ識別子が使われることが多ければ，バグ識別子に基づいて分類，配置する．ただし，業務やが複数ある場合あるいは契機が一つに定まらない場合，同じソフトウェア/バージョンに対して複数のソースコードパスを与えると良い．ファイルシステムを用いる場合シンボリックリンクで複数のソースコードパスを与えることができる．重要度の劣る，互いに競合する可能性のある2つの要件が続く．読解補助ツールが生成するデータを配置できる余地を残すこととソースコードパスを短かくすることである．

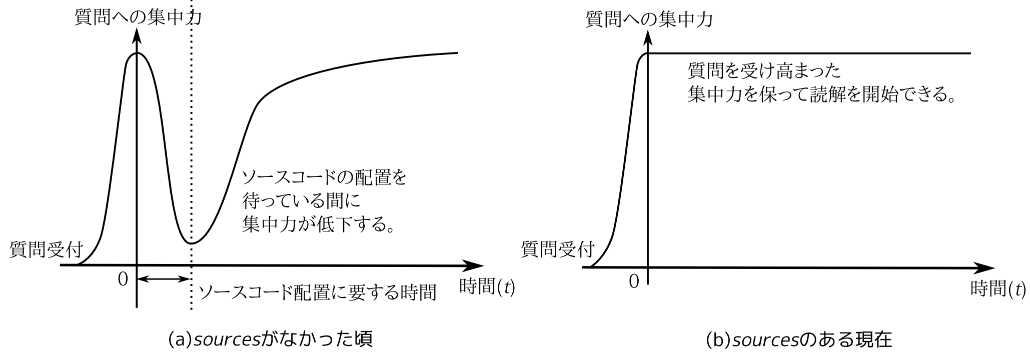
読解環境によってはソースコード読解に補助ツールを利用できる場合がある．例えばGNU Emacsは，手続きや変数，型の名前を与えるとその定義箇所を表示する機能 (タグジャンプ) がある．ただし補助ツールによっては，主処理が特別なデータ (ツールデータ) を必要とし，主処理とは独立に実行できる前処理がソースコードを入力としてデータを生成するよう構成されている．再びGNU Emacsを例に挙げると，タグジャンプのために名前と定義箇所の情報を記載したインデックスファイル (TAGS) を必要とする．TAGSの生成には付属のコマンドetagsを実行する．

通常，主処理に比べ前処理の方が時間を要するが，同じ入力に対してツールデータの作成は一度で良い．ソースコードの配置とあわせてツールデータの作成も実施しておけばツールを使う読解者の時間と手間を減らすことができる．ただし生成したツールデータの配置についても，ソースコード同様に整頓されている必要がある．また将来新しいツールを導入することも考慮してソースコードパスを設計する．

ソースコードパスが短かければ，ソースコードを指定のに伴う入力の手間が減る．一方，方法にはよるが，ツールデータを配置しようとするとソースコードパスが長くなる．読解環境を踏まえて，どちらの要件を優先するか考える必要がある．*sources*においては，読解者がソースコードパスの入力にファイル名補完機能を有するシェル，テキストエディタを利用していると仮定できたので，ソースコードパスが長くなってもツールデータを配置できる余地を大きくとることにした．

「配置しておき」という言葉は，ソースコードを入手して，必要であればそれを読解に適した形へ変換し，その後ツールデータを作成するという意味である．具体的な処理は，配置すべきソースコードがどういった形式で提供されていて，どこから，どのように入手可能かによって異なる． RHELのソースコードの場合については4章で説明する．

同じ業務に従事する人が複数人いれば，彼等の間でファイルシステムを「共有する」．共有方法はファイルシステムをホストする環境と，読解環境の組合せによる．共有にあたっては，誰の読解環境であっても同じソースコードが同じソースコードパスで参照できるようにする (ソースコードパスの標準化)．*sources*の場合，両方の環境がRHELであるためNetwork File System(NFS) を選択した．NFSの場合，NFSクライアントでマウントポイントを自由に選べてしまうので，NFSサーバーを設置しただけではソースコードパスを標準化できない．そこで利用者間で`/srv/sources`をマウントポイントとする，というポリシーを定めている．配置先あるいは共有方法によっては特別な取り決めが必要無いかもしれない．配置したソースコードをウェブインターフェイスで共有する場合，URLがそのまま標準化されたソースコードパスとなる．以降の説明では，利用者間で共有は適切になされているとして，ソース



(a)*sources*がなかった頃　　(b)*sources*のある現在

図 1　質問に対する集中力の時間的推移

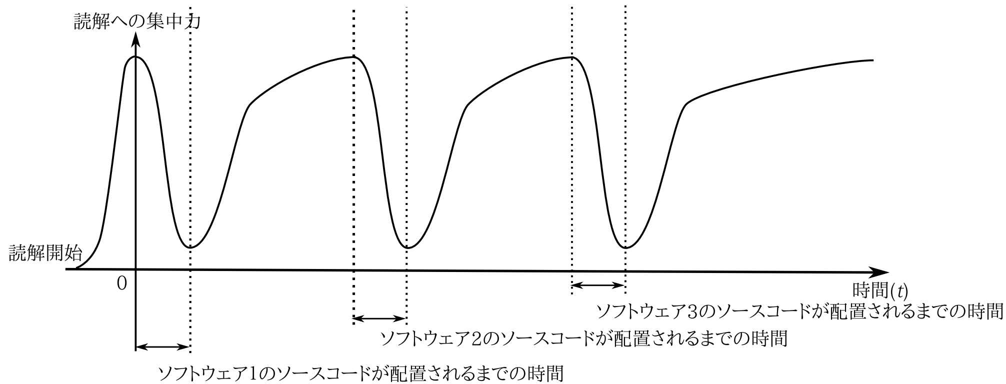


図 2　複数のソフトウェアに跨がる読解中に起こり得る集中力の減衰

コードパスと標準化されたソースコードパス を区別しない.

3　ソースコード事前配置法の効果

sources の同僚利用者 2 名に不定期に繰り返し行った非形式的なインタビューの内容と筆者自身が利用して得た感想を元にソースコード事前配置法の効果を述べる. 2 名の利用者はソースコード読解に関して筆者と同程度以上の経験を持つ.

3.1　配置時間の短縮と集中力の維持

適切な整頓ルールのものでソースコード事前配置法を実施すれば，従来，読解者が必要に応じて実行していたソースコードの入手と読解に適した形へ変換するのに要していた時間 (配置時間) をほぼ 0 に短縮できる．この短縮によりどの程度業務が改善されたかどうかは読解時間全体に対して配置時間が占めるオーバーヘッドで測ることができる.

sources が対象とする RHEL に含まれるソフトウェアの場合，配置時間は 1 分から 5 分程度である．時間の差はソースコードのサイズに由来する．稀にメンテナンス期間に重なると配置に数時間かかるケースもある．たとえ簡単な質問であったとしても調査にソースコードを使う場合，目的の情報を得るのに 30 分以下の読解で済むというケースは少ない．短縮できた時間だけを見ると *sources* による業務の改善は限定的である.

しかし短縮できた時間以外の効果として，利用者は質問あるいは読解に対して高

い集中力をもって取り組めるようになったことを挙げている．ソフトウェア調査業務の従事者は，質問を受け付けてからその内容の理解を進めつつ，並行して回答に向けた方策を練り始める: 回答にはソースコードを調べる必要があるか，あるとすればどのソフトウェアのソースコードを調べるべきか，読解を開始するソースコード箇所をどのような検索語で選出するか，といったことである．方策を練る過程で「質問への集中」をする．

読解を開始すると，読解者は様々な項目を記憶しようとする．それには例えば手続きの名前，変数の型やそれらの間の関係などが含まれる．それらを憶えていないと回答どころかソースコード読解を先に進めることができない．全て憶えることはできないので，限りある記憶容量を節約するため質問に関わる重要そうな項目だけを限定して憶えるのが良さそうである．しかし何が重要かということは読解してみないとわからないので限定するのが難しい．結局,「読解に集中」した状態，すなわち，その時点で重要だと判断した項目を記憶力の許す限り多く記憶することになる．

質問を受けてから時間の経過ともに質問への集中力がどのように推移するかのイメージを図 1 に示す．ソースコードの配置に要する時間以外の因子は無視した．図は客観的な情報に基づく厳密なものではなく，利用者の一人が主観で描いたものである．ただし図の意味するところについて筆者も他の利用者も同意している．質問の理解を終えいよいよ方策を実行に移そうとしたとき (図中 $t = 0$) に目的のソースコードが無いと，方策の実行を一旦中断してソースコードの配置を待つことになる．このとき集中力が減衰する (図 1(a))．ソースコードがすでに配置済みであれば，すぐに方策を実行できる．集中力を維持してソースコード読解に取り組める (図 1(b))．

質問によっては複数のソフトウェアを調査する必要がある．例えば複数のライブラリがリンクされているアプリケーションや，2 つのバージョンの間の差異を対象とする場合である．筆者の元へ来る質問では，多くの問題において Linux カーネルと 1 つ以上のユーザプロセスとの相互作用が存在する．読解の途中で別のソフトウェアを参照したくなったとき，ソースコードが配置されていなければ，その度に読解の集中力が減衰する恐れがある (図 2)．配置を待っている間に，それまで憶えてた項目を忘れてしまうのである．利用者の一人は集中力の低下を恐れて，配置待ちの間に直前まで読んでいたソースコードを繰り返し読み直していた．ソースコードがすでに配置済みであれば，読解への集中力の低下を避けることができる．

ツールの利用時にも同じことが起りえる．ツールを利用しようとして，それが前処理を必要とする場合，一旦読解を中断して前処理用のコマンドを起動する．この間に読解への集中力の低下する．これを避けるため *sources* では，ソースコード配置時にあわせて前処理を実施する (4 章参照).

3.2 Unix ツール群との高い親和性

ソフトウェア調査業務においてソースコードを読解する場合，目的のソースコード全体を読むということは稀である．ツールを利用して選出した箇所を読んで，そこで得た知見を元にツールを起動して再度選出する，ということを繰り返す．適切な選出によりコストの高い精読の範囲を狭めることができるため，読解環境にはツールが，読解者にはそのツールの使い熟しが求められる．特に複雑な条件で選出するには，一つのツールの出力を別のツールに入力するといった方法で複数のツールを組合せたツール (カスタムツール) を作成する必要がある．

読解環境である RHEL には Unix 開発環境に由来するテキストファイルとファイルシステムを処理の対象とする様々なツール (Unix ツール群) が充実している．これらのツールは「ファイルシステムに整頓して配置されたソースコード」と親和性が高い．*sources* の利用者は日常的に RHEL を使っており Unix ツール群に関して豊富なノウハウを持っている．*sources* があればそのノウハウをそのままソースコード読解に流用できる．ソースコード読解で頻繁に利用される Unix ツールには，ファイル間の差分を表示する diff コマンド，指定したディレクトリの配下を巡回し条件

に合致するファイルを列挙する find，正規表現検索する grep がある．またカスタムツールを記述するために標準的なシェルが活用できる． *sources* の利用には特殊なツールなどは必要ないため，利用者の導入コストは極めて低い．

典型的な選出の条件とツール呼び出しを紹介し，そのツール呼び出しに対して *sources* のソースコード整頓ルールがどのように役立っているかを示す．

文字列検索

PATH で指し示されるディレクトリ以下にあるソースコード群から文字列 (STRING) を検索したいことが頻繁にある．このとき以下のコマンドラインを用いる．

```
find PATH -type f | xargs grep STRING
```

詳細は 4.3 にて述べるが，*sources* ではソフトウェアのコンポーネント，そのバージョン毎ににディレクトリを分割してソースコードを配置している．そのため配置箇所の指定がそのまま検索範囲の指定として直接コマンドライン中で利用できる．例えばソフトウェア S のバージョン V について STRING を検索したい場合，コマンドラインは以下のようになる[1]:

```
find /srv/sources/S/V -type f | xargs grep STRING
```

複数のソフトウェアにまたがってソースコードを検索したい場合，そのソフトウェアのソースコードパス (PATH1， PATH2，...) をシェルのループ構文 for に与えれば良い:

```
for p in PATH1 PATH2 ...; do
        find p -type f | xargs grep STRING
done
```

バージョン間の比較

ソフトウェア S の異なる 2 つのバージョン (V1，V2，...) の間でどのようは変更が導入されたのか確認したいことがある．このとき以下のコマンドラインを用いる:

```
diff -r S/V1 S/V2
```

同じバージョンのソフトウェアであってもバージョン毎に異なるディレクトリに配置してあるため，配置箇所の指定がそのまま比較対象の指定として直接コマンドライン中で利用できる．

3.3 読解箇所の円滑な伝達

sources の利用者の間でマウントポイントを標準化したため，読解環境によらず同一のソースコードパスが使える．ソースコードパスはそのままメールに記載したり，あるいは口頭で読み上げることができるため，同僚間や顧客とのコミュニケーションがスムーズになった．

ソフトウェア調査業務の過程で，特定のソフトウェアに詳しい同僚に支援を求めることがある．例えばそのソフトウェアの特定のソースコード箇所について解説してもらいたい，といった場合である．*sources* が無い場合，支援依頼者は解説して欲しいソースコード箇所を指し示すのに不便があった．解説して欲しいソースコードが同僚の読解環境に無い場合，目的のソースコードの配置，すなわち入手及び変換を同僚に依頼する必要がある．配置が完了してようやく解説して欲しい箇所を指し示すことができる．たとえ同僚の読解環境に配置済みであったとしても，支援依頼者が直接的にそれを指し示すことができるとは限らない．読解者の方針によってソースコードを読解環境中のどこに配置するかというのが異なるためである: ソフトウェアの種類毎にディレクトリを作成するという方針もあれば，顧客からの質問毎にディレクトリを作成し，その調査に必要となったソースコードをそこに配置す

[1]実際の *sources* におけるパスの指定はもう少し複雑である．ここでは説明のために簡略化した．

るという方針もある．

sources があればこのような不便は無い．解説して欲しいソースコード箇所のソースコードパスを貼り付けてメールで送れば良い．メールを受けとった同僚はメール中のソースコードパスで指定されたソースコードを開くだけで良い．

`/srv/sources` にマウントするよう依頼して，*sources* の一部分を複製してファイルシステムイメージにまとめた者を顧客に提供している．質問を送る前に顧客自身がすでにソースコード読解を進めていることがある．そのような質問の場合，文面にソースコードパスを記載すれば，その箇所を引き継いで読解を始めることができる．回答の裏付けにソースコード箇所を示す場合にも，そのソースコードパスを記載することで顧客はソースコードソースを見ながら回答内容を確認できる．

ソースコードパスが有効なのは他者とのコミュニケーションのときだけでなかった．自身のソースコード読解を振り返るのにも役に立った．ソースコード読解の過程で理解したことをソースコードパスとともに記録しておけば，時間を経て再度のそのソースコード読解が必要となったとき，二次資料として参照できる．資料中のソースコードパスは，理解の根拠がどのソースコードに由来するのか確認したいときに役立つ．

4　RHEL を対象としたソースコード事前配置法の実践

ソースコード事前配置法の実践例として *sources* で採用したソースコードパスについて述べる．ソースコードパスの設計は対象とするソフトウェアの構成とソースコードの配布形態に依存するので，ソースコードパスの説明に先立ち RHEL ではそれらがどのようになっているか説明する．

4.1　RHEL の概要

RHEL は企業ユースを想定した GNU/Linux ディストリビューションの一つである．1000 以上の OSS から構成されている．それぞれの OSS は非営利団体，企業，個人など異なる開発元 (アップストリーム) で開発が進められている．どこでソフトウェアを公開するか，リリースサイクル，互換性をどの程度重視するかといった開発方針はアップストリームによって異なる．顧客がそれらの差異に煩わされること無く OSS を利用できるよう，RHEL がそれを吸収する．

RHEL には 24 から 36ヶ月に一度を目安とするメジャーリリースと 6 か月に一度を目安とするマイナーリリースがありそれぞれに番号が割り当てられている．1 つのメジャーリリースに対して複数回[2]のマイナーリリースが伴う．メジャーリリースでは，アップストリームから新しいバージョンを取り入れて積極的に機能強化を図ることを互換性に優先する．しかし一度メジャーリリースを発表した後は，そのマイナーリリースを繰り返す間，互換性を壊すような変更をしない，という方針で開発，保守されている．マイナーリリースには主に不具合の修正や脆弱性への対応が含まれる．本稿では説明のため rhel$M$um という表記でメジャーリリース番号 M，マイナーリリース番号 m の RHEL のリリース (RHEL リリース) を指し示すことにする．メジャーリリースそのものを指し示す場合 $m = 0$ とする．最新のメジャーリリースは 6 であるが，4，5 のメジャーリリースについても並行して提供されており，顧客は目的に応じて選択して利用できる．そのため複数のメジャーリリース，マイナーリリースがソフトウェア調査業務の対象となる．

RHEL を構成する OSS は rpm パッケージ (以下，パッケージ) と呼ばれるアーカイブ形式で個別に梱包されて出荷される．パッケージにはパッケージ名，パッケージバージョン名，パッケージリリース名という 3 つのメタ情報が含まれる．基本的にこの 3 つの情報からパッケージが一意に定まる．パッケージバージョン名とパッケージリリース名には説明を要する: パッケージバージョン名がアップストリーム

[2] rhel3 の場合 9 回だった

表 1　RHEL リリース名とパッケージバージョン名，パッケージリリース名の例 (対象 kernel パッケージ)

RHEL リリース名	パッケージバージョン名	パッケージリリース名
rhel5u0	2.6.18	8.el5
rhel5u1	2.6.18	53.el5
rhel5u2	2.6.18	92.el5
rhel6u0	2.6.32	71.el6
rhel6u1	2.6.32	131.0.15.el6

にて与えられるのに対して，パッケージリリース名は (RHEL の開発保守元である) レッドハットが与える．基本的にメジャーリリース時にパッケージバージョン名が決まる．以降マイナーリリースを繰り返す間は，対象ソフトウェアに変更を施す度にパッケージリリース名を変更する．表 1 に RHEL リリース名とパッケージバージョン名，パッケージリリース名の例を示す．対象は rhel5，rhel6 に含まれる kernel パッケージである．

メジャーリリースあるいはマイナーリリースのタイミングで多数のパッケージが公開される．しかし重大な不具合や深刻な脆弱性が発見された場合，マイナーリリースを待たずに修正がパッケージとして公開されることがある．これを非同期エラータと呼ぶ．発見される問題は週に高々数個程度であるが，複数のメジャーリリースが並行して保守されているため，修正をそれぞれのメジャーリリースに反映するとパッケージの数としては数十個となる．そこで *sources* では，cron を用いて毎日，新しく公開されたソースコードが無いかを調べ，あればそれを配置する，という処理を実行している．

4.2　ソースパッケージ

rpm パッケージは src.rpm パッケージ (ソースパッケージ) を入力として rpmbuild コマンドによって生成される．src.rpm パッケージは ftp.redhat.com にて公開されている．

ソースパッケージは 3 つのファイル群から成る:

オリジナルソースコード　アップストリームで公開されたソースコード．tar コマンドなどによって複数のソースコードをまとめたアーカイブファイルになっていることもある．

パッチファイル　レッドハットが作成したオリジナルソースコードに対する変更を記載したファイル．これもソースコードの一種と見做す．

spec ファイル　パッケージに関するメタ情報を記載したファイル．主に以下の情報が記載されている．

- パッケージ名，パッケージバージョン名，パッケージリリース名
- パッチファイルのリスト
- rpmbuild によって解釈されるビルドの手順
- ビルドの結果生成されたファイルのインストール先
- レッドハットでの修正履歴

rpmbuild は，オリジナルソースコードがアーカイブ化されていればそれを展開して，patch コマンドを呼び出してパッチファイルに記載された変更をオリジナルソースコードに反映させた後，spec ファイルに記載された手順に従いコンパイラ等を呼び出してパッケージに含めるべきファイルをビルドする．最後にビルドされたファイルを rpm 形式にアーカイブしてパッケージを作る．

```
    /srv/sources/sources/C/N/V-R
    (例) /srv/sources/sources/k/kernel/2.6.18-8.el5
```

N: パッケージ名，C: N の最初の一文字，V: パッケージバージョン名，R: パッケージリリース名

図 3　パッケージバージョン名，パッケージリリース名とパッケージ名によるパッケージ特定ルール

```
    /srv/sources/dists/D/packages/C/N
    (例) /srv/sources/dists/rhel5u0/packages/k/kernel
```

D: RHEL リリース名，N: パッケージ名，C: N の最初の一文字

図 4　RHEL リリース名とパッケージ名によるパッケージ特定ルール

4.3　ソースコードパスの設計

パッケージ特定ルールとパッケージ局所ルールの 2 つのルールを組合せてソースコードパス が決まるように設計した．パッケージ特定ルールはその名の通りパッケージ毎のディレクトリを特定する役割りを持つ．パッケージ特定ルールで特定されるパッケージ毎のディレクトリについて，その配下のファイルの配置についてはパッケージ局所ルールが決める．

パッケージ特定ルール

パッケージを指定するには，パッケージバージョン名，パッケージリリース名とパッケージ名の 3 つ組を使う方法と，RHEL リリース名とパッケージ名の 2 つ組を使う方法がある．表 1 によれば,「rhel5u0 の kernel パッケージ」と「2.6.18 の 8.el5 の kernel」は同じパッケージを 2 つの方法で表記したものになる．質問中には調査対象に関してパッケージ指定がある場合，2 つの方法のいずれかが使われることが多い．

質問中のパッケージの表記がどちらの方法であっても迷うことなく目的のディレクトリを特定できるよう 2 つのルールを設けた． 3 つ組に由来するルールを主とした．2 つ組に由来するルールは，3 つ組に由来するルールで特定されるディレクトリへのシンボリックリンクを用いて実現した．図 3 は，パッケージバージョン名，パッケージリリース名とパッケージ名の 3 つ組に対するルールである．一つのディレクトリ配下のファイルが多過ぎると ls コマンドの出力が画面に収まりきらず見にくくなる．そこでパッケージ名 N の上位に N の最初の一文字 C を名前とするディレクトリを設けた．

図 4 は RHEL リリース名とパッケージ名の 2 つ組に対するルールである．`/srv/sources/dists/D/C/N` としなかったのは，全文検索ツールのインデックスファイルなど，RHEL リリース全体を対象とするツールデータを格納する余地を残すためである．ツールの名称を T としたとき，ツールデータを `/srv/sources/dists/D/plugins/T` 以下に配置する．現状では，ツールデータとして hyperestraier [2] のインデックスファイルを配置している．

パッケージ局所ルール

ソースパッケージに含まれるファイルを対象として，表 2 に示すルールに従い各パッケージのディレクトリにソースコード及びツールデータを配置した．`pre-build` に配置したパッチファイル適用済みソースコードは，配置時にオリジナルソースコードにパッチファイルを適用して合成する．合成処理には rpmbuild コマンドを使用する．ツールデータについて現行では etags, ctags, cscope の各ツールが生成するイ

表 2　パッケージ局所ルールと例

ソースパッケージ中の情報	ルール
オリジナルソースコード	vanilla/以下に配置する
パッチファイル	archives/以下に配置する
パッチファイル適用済みソースコード	pre-build/以下に配置する
spec ファイル	specs.spec
ツールデータ	plugins/T/以下に配置する

T: 読解補助ツールの名前

```
/srv/sources/sources/k/kernel/2.6.18-8.el5/archives/git-geode.patch
/srv/sources/dists/rhel5u0/packages/k/kernel/plugins/etags/TAGS
/srv/sources/sources/g/gcc/4.4.5-6.el6/archives/gcc44-hack.patch
/srv/sources/dists/rhel5u0/packages/k/kernel/specs.spec
/srv/sources/dists/rhel6u1/packages/s/sendmail/plugins/ctags/tags
```

図 5　*sources* におけるソースコードパスの例

ンデックスファイルを配置している．

パッケージ特定ルールとパッケージ局所ルールを使い特定されるソースコードパス及びツールデータの配置の例を図 5 に示す．

5　まとめ

7 年間のソフトウェア調査業務において有効だったソースコード事前配置法について説明した．RHEL のソースコードを対象にソースコード事前配置法の要となるソースコードパスの設計について述べた．

今後の課題は 2 つある．一つは *sources* 上で動作する読解補助ツール群を充実させることである．今まで質問毎に必要に応じて作成していたカスタムツールについて整理し，汎用化の可能性を探る．またソースコード読解に関する学術的な研究成果を調査し，*sources* 上で動作するツールへ落し込む．

もう一つは，より多くの RHEL の利用者に *sources* あるいは同等のシステムへのアクセスを提供することである． 具体的に 3 つの開発作業を想定している: (1) ウェブインターフェイスを追加する， (2) 複製したファイルシステムを配布する，(3)*sources* の設置及び運用に使用しているプログラム群 *srpmix* [1] を，利用者サイトで mo 実行できるよう文章化する．(3) について，プログラム群そのものは OSS として入手可能であるが，ドキュメント化が不十分なため利用にはソースコード読解が必須である[3]．

謝辞　現行の *sources* のソースコードパスを設計にご協力頂いたグーグル株式会社樽石将人氏に感謝する．*sources* の開発に理解を示し機会を提供して下さったレッドハット株式会社グローバルサービス本部長高宮敏幸氏に感謝する．利用方法に関するフィードバック及び *sources* の実装方法の助言についてレッドハット株式会社森若和雄氏，同佐藤暁氏に感謝する．

参考文献

[1] 大和 正武， 樽石 将人: Scripts to expand src.rpm, *http://sourceforge.net/projects/srpmix*
[2] 平林幹雄: 全文検索システム Hyper Estraier, *http://fallabs.com/hyperestraier/*

[3] /srv/sources/source/s/srpmix に配置されている．