

# LyX におけるOnTheSpot様式の実装

横 田 宏 治

## 概要

学術分野の論文執筆に用いられる LyX において行った日中韓文字入力の新機構導入の概要である。本改訂においては、入力メソッドをOverTheSpot様式からOnTheSpot様式に変更して、LyX 本体との統合を図ったほか、入力の負担を軽減するような入力メソッドの自動切替機構の導入、その他不具合の修正を行った。

キーワード：LyX、L<sup>A</sup>T<sub>E</sub>X、インプットメソッド、日本語入力、UI

## 1 はじめに

LyX は学術分野を中心に論文等の執筆に用いられるオープンソースの文書プロセッサである。

同アプリケーションは、2007年の1.5版で多言語対応となったものの、入力メソッドを使用する言語の話者によるメンテナンスがしばらくなかったため、近年では不具合が見られたり、入力スタイルがやや時代遅れとなっていたりしていた。学生の卒業論文執筆に LyX を使わせる方針を立てたのを契機に、2023年末よりこれらの不具合の修正と、アドホックなOverTheSpot様式からモダンなOnTheSpot様式の入力スタイルへの変更を行ったので報告する。LyX はかなり複雑なシステムであり、開発を担うボランティアが新たに参入する上で、ハードルが高い面がある。本稿に記録を残すことで、将来 LyX の開発を考える人の閾を

下げることに役立てば望外の幸せである。

学生の卒業論文執筆に LyX を使わせようと計画したのは2023年後半のことであった。それまで一ユーザーとして、ほとんどの文章を LyX で書いていたはずであったが、LyX の日本語環境に徐々に生じていた不具合に気付くことはなかった。この不具合は、文字変換の候補ウィンドウが変換中の本文を覆い隠してしまったり、何を書いていたか確認できなくなってしまうという、かなり重篤なものであったが、慣れとは怖いもので、何時の頃からか使いづらくなったという感触はあったものの、とくに意識することはなかったのである。これに気付いたのは、学生に LyX の使い方のガイダンスを行う必要が出て、意識的に操作の確認を行ったおかげである。いずれにせよ、不要な混乱の元になるのは明白であったので、これをそのまま学生たちに使わせるわけには行かなかった。そこで、不具合の主要な部分をただちに修正して

表1 LyX 2.4.x版以降のおもな修正と新機能

修正・新機能
入力メソッドとの包括的な通信のサポート
OnTheSpot様式での文書編集
マルチストロークショートカットでの入力自動制御
数式モードでの入力自動制御
ダイアログでの入力自動制御
InputMethod/GuiInputMethodクラスの新設
タブレット等のデバイスでの将来的な入力の基礎の形成

LyX 2.4版に反映させたが、コードを見れば、日本語話者の誰かが、いずれ LyX と入力メソッドとの統合を図らなくてはならなくなるのは明らかであった。中国語、韓国語も入力メソッドを使用するものの、日本語ほど複雑な機構は有していないためである<sup>1)</sup>。かくして、上記の不具合の修正——これは入力メソッドと LyX アプリケーションとの座標系のずれから生じていた——に加えて、表1に挙げたような修正と新機能の追加を行うこととした。

表中にあるOnTheSpot様式での文字編集とは、日本語変換に際して表示される未確定文字列が、通常の確定文字列と同様に本文中に表示される表示様式のことである。つまり、本文中のAとBの間に新たに文字列を挿入しようとする場合、変換前の未確定文字列Cが挿入されるにしたがって、Bの部分がCと重なることなく後ろに押し出されていくような表示様式のことである。これに対して、LyX 2.4.x版までで採用されていた表示様式はOverTheSpot様式と

呼ばれ、未確定文字列CがBの上に重なって表示されるものである。この様式は古くはよく使用されていたが、かつては地の文字列と混乱しないように、未確定文字列がビビッドな彩色で区別されるのが通例であった。このような彩色が採られなくなった現在では、未確定文字列と地の文との区別が付かないだけでなく、お世辞にも美しい表示方法とは言いがたい。また、入力メソッドエディタが進歩して、ユーザの入力方法が必ずしも短い文節変換に限られなくなった現在では、未確定文字列が——変換作業中の文字ではないとは云え——地の文を覆い隠してしまう表示方法は、望ましいとは云えない。

このように、OnTheSpot様式の表示方法の方が、ユーザーに入力メソッドとアプリケーションの境界を感じさせない分、自然であると云える。とはいえ、OnTheSpot様式への移行の主要な意義は、後述するように、以上のような外見上の変更よりも、アプリケーションでの内部的な処理の変更にある。

表1の新機能中、マルチストロークショートカットと数式モードでの入力自動制御は、これらのモードでの入力メソッドのオン・オフを自動で行うものである。これは入力メソッドとの双方向通信を充実させたことによって可能となっている。これによって、ユーザは数式入力時やemacs型のキーバインディングを使用する

1) LyX のような多言語・マルチプラットフォームのアプリケーションでは、それぞれの言語・システムについてある程度のチェックが必要である（これはオープンソースでの開発には重荷でもある）。入力メソッド関連では、日本語以外にも中国語（繁体字・簡体字）と韓国語について、主要な入力メソッドエディタでのテストが必要となる。

際に、ややもすれば指のつりそうな入力メソッドのオン・オフ作業を行わなくて済むので、入力速度を向上させ、ユーザの入力負担感を減らすのに貢献する。

著者の属する明星大学経済学部においては、全学生に卒業論文の提出を卒業要件として課している。多くの学生にとっては、論理的で構造の整った長文を書くのは初めての経験であり、戸惑うことが多いようである。学生たちの不安の多くは、高すぎる自由度に起因するものとみられるため、外形的条件によるハードルを設定することで、比較的容易に課題をこなせるようになるようである。具体的には、①卒業論文のテーマと一定数の関連文献リストの提出、②各章の要約を伴う論文の章立ての設計、③実証分析の計画、④時期に応じた課題の成否と字数による中間評価とフィードバック、といった形での進行を採用している。論文構造の設計から中身へ進行させる上で、LyX をツールとして使用させるのは親和的であり、仕上げの最終段階で体裁上の余計な要素に指導の時間を取られずに済むことが期待できる。実際、要求されている整然さや客観性を理解させるのは——個人差はあるが——なかなか難しい。そのようなこともあって、LyX（および私の作成した卒業論文用レイアウト）とZotero、Better BibTeXを一体としてツールに用いることとしたのである。LyX をもって論文の構造を担保する一方、Zoteroに参考文献を収集させることをターゲットに置くことで、参考文献への言及を明白に位置づけることができる。学部生に対しては、社会に出てからよく使われるWordを使わせるべきとの考えもあるとは思われるが、LyX を使用して、スタイルベースの構造的な編集を行う癖を身につけさせることには価値があるものと思われる。Wordでもスタイルベースの編集は可能であるが、ユーザがそれを使用しなくては

意味がない。一方で、アプリケーションの個別のメニューを知ることは一朝一夕でも可能であろう。

## 2 LyX の構造

### 2.1 抽象クラスによるGUIの分離

LyX では、グラフィカル・ユーザ・インターフェース（GUI）から独立したコード（以下S層と呼ぶ）と、GUIで使用するQtライブラリ依存のコード（以下G層）とが、分離されており、C++の多態性（polymorphism）を活用したコーディングがなされている。ファイルシステム上、前者S層のコードを含むファイルはディレクトリ `src`（名前空間は `lyx`）に置かれ、G層のファイルは `src/frontends/qt`（名前空間は `lyx::frontend`）に置かれる。両者の橋渡しをするためのインタフェースクラスは `src/frontends`（名前空間は `lyx::frontend`）に置かれる（以下 I 層と呼ぶ）。新たな機能を追加するに当たっては、その機能が上記各層で完結するような特殊な場合を除き、GUIライブラリを抽象化した機能を設計してS層に実装し、I 層を通じてG層からアクセスする形にすることが必要である（図1）。G層に依存する部分はなるべく最小限にすることが望ましい。

LyX は、かつてはxformsをGUIとして用いていたが、バージョン1.3からQtをGUIとして採用したバージョンも並行してリリースするようになった。さらにバージョン1.5からはQtのみをGUIとして採用している。このような移行は、上記のような構造があるがゆえに可能になったといえる。コードのGUIライブラリ依存部分を分離することによるメリットとしては、特定の外部ライブラリがサポートしていないプラットフォームをサポートすることが比較的容易になること、標準的ではない外部ライブラリ

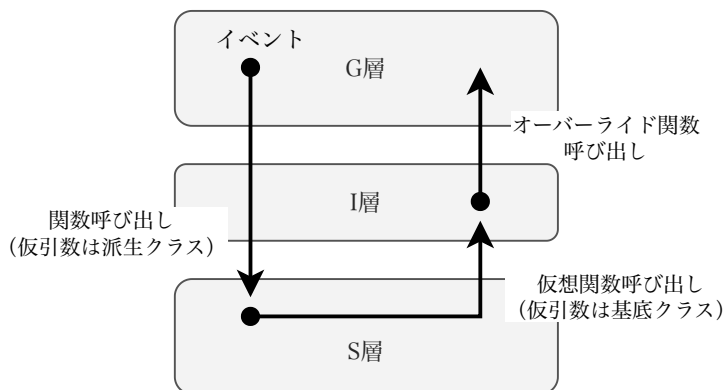


図1 各層とのやりとり

の存続が危ぶまれるような事態が仮に起こった場合にも、プロジェクトの維持が可能となること等が挙げられる。一方で、コードが複雑になることで、ボランティアに依存しているオープンソースプロジェクトへの参加の閾を上げてしまうデメリットもある。

さて、LyX と入力メソッドの通信はGUIで行われるので、それに関連した中心機能はG層で実現される。一方でOnTheSpotスタイルを実装するためには、preeditと呼ばれる変換確定前の文字列を、通常の確定済み文字列と同様に、S層に入れ込むことが必要である<sup>2)</sup>。なぜならば、GUIの編集画面を描画する際には、描画対象となるテキストやオブジェクトをS層において画面描画に最適となるように再編してから（これはGUIライブラリとは独立に行えること

に注意）、描画処理を行うからである。現行のOverTheSpotスタイルでは、地のテキストの上にpreedit文字列を上書き描画（OverTheSpot）するだけで良いので、関数の呼び出しを除けば、コーディングはG層のみで完結し、S層を呼び出す必要はなかった。一方、OnTheSpotスタイルでは、各層にまたがるやりとりが必要となる。その際、各層との出入力に必要なとされる機能をライブラリ独立な形に抽象化して、そのインタフェース（宣言）をI層に仮想的に実装する必要がある。すなわち、I層における各関数の仮引数および返値はGUIライブラリから独立でなくてはならない。ここのやりとりでGUIライブラリを呼ばなくて済むような設計を行うことが大切である。

## 2.2 フロントエンドの構造

LyX は、モデル/ビュー/コントローラ（MVC）プログラミングパラダイム [3] に即した設計を採用しており、各主要クラスの対応は表2のようになっている。モデルを担うBufferクラスは、LyX ファイルの内容をメモリ上に表したものである。1つのBufferはオープンされた LyX ファイル1つに対応しているが、Bufferはファイルが画面表示されているかどうか

2) ただし、preedit文字列はモデルすなわちテキスト本体には入れずに、ビューすなわち描画ステージにおいて描画文字列にのみ挿入する（表2参照）。この点は、モデル本体に文字列を反映させる確定文字列とは取り扱いが異なる。確定文字列は、他のテキストと同様に、Rowクラス中でSTRING型のElementとして扱われるのに対し、preedit文字列は、英文/コマンド補完のときに用いられる自動補完文字列と同様に、VIRTUAL型のElementとすることで取り扱いの区別を行っている。

表2 フロントエンドにおける各クラスの関係

役割	クラス
モデル	Buffer
ビュー	WorkArea/GuiView
コントローラ	BufferView/Painter/Cursor

かにはかかわらず存在する。Buffer自体は、画面表示の状態に関する情報は持っていない。ビューは、実際のスクリーン領域に対応するWorkAreaクラスと、ウィンドウ全体に対応するGuiViewクラスが担う。コントローラは、BufferView/Painter/Cursorの各クラスが担当する。これらのクラスの間関係は、図2のようになっている。1つのWorkAreaは、1つのBufferViewを保有している。WorkAreaは、実際のスクリーン領域におけるキーボードやマウス操作によって起こる文字の挿入／削除を、BufferViewをツールとして使用してBufferが理解可能な形に変換する。また、WorkAreaは、BufferViewを使って、Bufferの一部にアクセスすると同時に、それを描画ルーチンに送って描画領域に表示させる。このとき、BufferViewは、BufferへのアクセスにCursorクラスを使用し、描画に際しては、直接の描画機構を担うPainterを使用してBuffer内の各インセットに描画させる。インセット (*inset*) は、LyX インタフェース内では、その表示のされ方から差込枠と訳出されているものであるが、LyX 内部では一群のオブジェクトを指す抽象的な存在であるので、ここではインセットとして言及する。LyX 文書内には、数式や脚註、書誌情報、フロートなど、特定の表示のされ方をするオブジェクトを挿入することができる。これらのオブジェクトは、すべてインセットとして取り扱われる。インセットは入れ子にすることができ、インセットが埋め込まれている本文も、内

部的にはインセットとして取り扱われる。描画は、各インセットが独立して担当し、インセットが入れ子になっている場合には、再帰的に実施されることに注意が必要である。

### 3 入力メソッドとの通信

入力メソッドとの通信は、GUIレベル(G層)で行われる。Qtにおいては、QInputMethodEventクラスとQInputMethodQueryEventクラスが主に関係する。前者はQtライブラリ側からLyX (あるいは一般にアプリケーション) への通信に用いられる。LyX 側からQtライブラリに通信したい場合には、LyX から入力メソッド関連の状態が変化したことを通知し、Qt側から LyX に対しクエリを発せしめ、それに対して LyX が返答を返す形態を取る。この動作には後者のQInputMethodQueryEventクラスが主に関与する。他には、入力メソッドへのアクセスを提供するQInputMethodクラスや、QWidgetクラス・QGuiApplicationクラスの一部の関数が関連する。

#### 3.1 InputMethod/GuiInputMethodクラスの新設

LyX 内部で主にpreeditの表示処理と入力メソッドとのコミュニケーションを行うためのクラスとして、G層にGuiInputMethodクラスを新設し、S層へのインタフェースとして抽象クラスのInputMethodクラスをI層に設ける。





*mode*) と、それらの表音文字列を目的とする表記に変換するための**仕上げ段階** (*completion mode*) がある<sup>4)</sup> が、Qt 6 の段階では、アプリケーション側としては両者を区別する必要はない。両段階の区別をユーザに伝えるために、*preedit*文字列には下線を引いたり背景に着色したりして区別をするのが一般的であるが、この仕事はQtライブラリの側が担当するので、アプリケーション側はその表記を正しく反映すれば、目的は達せられる (Qt 5 までの段階ではこの点が不十分であったので、アプリケーション側の対応が必要とされた)。Qtはこの表記方法をアプリケーション側に伝えるために、*QInputMethodEvent*コンストラクタに渡される属性として*QTextChar*クラスを渡すことで通信を行う。この通信形態は、GUIライブラリ分離型の構成を持つ LyX には一つの課題を課すことになる。すなわち、渡されたデータを LyX 内で処理するには、Qt特殊な*QTextChar*クラスで渡された属性を、S層で処理できる形に抽象化し、出力の際に再度復元できるようにしなくてはならないのである。この点については、第4.2節で詳述する。

*preedit*の表記属性は、他の属性と一緒にたに*QInputMethodEvent::Attribute*クラスの*QList*として渡される。*QList*の中身の規則性は保証されていないので、渡された*QList*の中身を走査して整理する作業が必要である。*preedit*の表記様式に関する*QList*要素は、*Attribute*が*TextFormat*型のものである (*Attribute*の型は *type* 属性として渡される)。この要素は*preedit*の特定部分の文字やバックグラウンドの装飾の情報を *value* 属性として持っており、*QTextCharFormat*クラスとして

与えられる。*preedit*の変換作業中には、*preedit*中の変換対象となる部分をハイライトして、まだ変換作業が終わっていない、あるいはすでに終わった部分と区別することになる。*preedit*文字列自体も周囲の確定文字列と区別されなくてはならないので、編集段階では周囲と異なる表記様式が少なくとも1種類必要であり、仕上げ段階では少なくとも2種類が必要となる (これらはQtが指定してくる)。仕上げ段階では、それらを*preedit*中の各セグメントに割り当てる必要がある。セグメントは、変換対象のセグメント・その前の非対象セグメント・その後の非対象セグメントが発生しうる。プラットフォームによっては、非対象セグメントをさらに文節毎に区切って指定してくる場合もある。*value*属性が持っているのは、これらの各セグメントをどのような装飾で表すかという情報である。また、そのセグメントの開始位置に関する情報は *start* 属性、そのセグメントの長さに関する情報は *length* 属性で与えられる。

なお、Qt 6.7時点でのレファレンス [2] によれば、同じセグメントに関する情報が重複することはないとされているが、MacOS上のQt 6ではシステムティックに重複するパターンが観察され、変換対象セグメントを指し示すために積極的に使用されていると解釈することもできる。この性質を利用したコーディングは為されているが、通信様式が変わっても問題はないように設計されている。

また、カーソル表示に関する情報が、*Cursor*型の*Attribute*として*QList*に含まれる。カーソル位置は *start* 属性として与えられ、*length* 属性はカーソルの表示 (*length* > 0) あるいは非表示 (*length* = 0) を指定する。また、*value* 属性としてカーソルの色を指定することができるが、使用されるのは稀であるようである。

また、*Attribute*型には、表3に示すように、

4) [2] *QInputMethodEvent* Class項目の“Detailed Description”参照。

表3 QInputMethodの各Attributeの属性

Attribute type属性	属性	摘要
QInputMethodEvent::TextFormat	start	適用されるpreeditセグメントの位置
	length	適用されるpreeditセグメントの長さ
	value	セグメントの書式を表す QTextCharFormat
QInputMethodEvent::Cursor	start	カーソルの位置
	length	カーソルの表示／非表示（0なら非表示）
	value	カーソルの色を表す QColor
QInputMethodEvent::Language	value	セグメントの言語を表す QLocale
QInputMethodEvent::Ruby	value	セグメントのルビを表す QString
QInputMethodEvent::Selection	start	選択する箇所の本文中の位置
	length	選択する箇所の長さ

他にLanguage・Ruby・Selectionがあるが、これらもほとんど使用されていないようである。

### 3.3 QInputMethodQueryイベント：アプリケーションから入力メソッドへ

QInputMethodクラスは、入力メソッドへのアクセスを提供するクラスである。同クラスのpublic slotであるupdate関数は、LyXのQInputMethodQueryの属性に変化が生じたことを入力メソッドに通知するのに使うことが

でき（属性の変化した対象を仮引数で指定する）、入力メソッドはこれを受けて、LyXにQWidget::inputMethodQueryを呼び出す形でクエリを発する。LyXはこの関数に返す形で入力メソッドに返答を渡す（図3①～③）。

QInputMethodQueryEventが問い合わせる内容としては、主に表4に挙げたようなものがある。特に、ImEnabledとImCursorRectangleが入力メソッドの挙動に関して主要な役割を果たす。ImCursorPositionとImSurroundingText

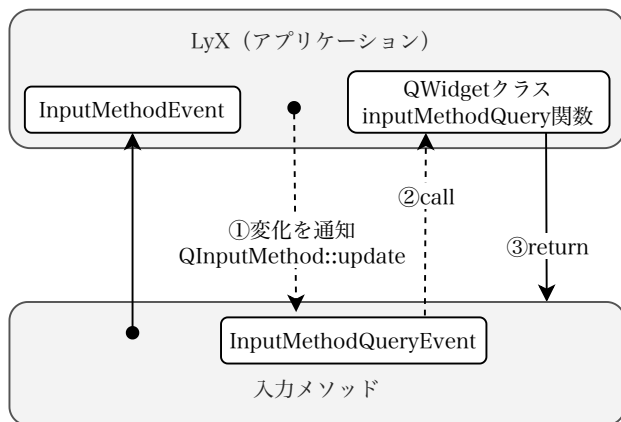


図3 入力メソッドとのやりとり



表 4 QInputMethodQueryの主要な値

定数	摘要
Qt::ImEnabled	ウィジェットが入力メソッドの入力を受け付けるか否か
Qt::ImHints	入力を期待する文字種のヒント
Qt::ImCursorRectangle	カーソル領域を被覆する長方形の座標
Qt::ImCursorPosition	カーソルの文書内における論理位置
Qt::ImSurroundingText	カーソルの周辺のテキスト

およびそれらに関連した値は、カーソル周囲のテキストを入力メソッドに渡して、変換効率の向上等に寄与するものと推測されるが、代表的な入力メソッドのいくつかに対して試験した限りにおいては、顕著な効果は観察されない<sup>5)</sup>。

**Qt::ImEnabled** Qt::ImEnabledは入力メソッドの入力を受け付けるかどうかを制御する。今回の実装では、これを用いて、数式内の数式モード時およびマルチキーシークエンスのショートカット入力時の入力メソッドの自動オフの制御を新たに加えた。この新機能がユーザの入力作業の労力軽減に与える影響はかなり大きいものと見込まれる。とくに入力速度の向上に貢献するemacs系のキーバインディングにおいて効果が顕著である。入力メソッドの切替は多くの場合2つのキーの同時打鍵であり、けっして打ちやすい組み合わせではないものもある。日本語入力中に、emacsキーコマンドを発行しようとする、キーコマンドの前後にこの同時打鍵を挟まなくてはならなかったが、今回の措置でこれが不要となった。また、一部ダイ

アログ上で入力メソッドヒントを利用した入力制御を導入した。

#### Qt::ImCursorRectangle

Qt::ImCursorRectangleは、入力メソッドが仕上げ段階に表示する変換候補ウィンドウの表示位置に関連する。Qt::ImCursorRectangleに対して、アプリケーションのウィジェットの座標系におけるカーソル位置を与えると、入力メソッドはそれを入力メソッド自身が持つ座標系に変換して表示を行う。ここで注意しなくてはならないのは、ウィジェットと入力メソッドアイテムの座標系の相対位置は、いったんウィンドウやアプリケーションの切替を行うと、一般的にずれてしまうということである。したがって、元々持っていたウィジェットと入力メソッドアイテムとの相対位置関係を、ウィンドウ等の切替を行った後に復元する必要がある。これらの作業は、QInputMethodクラスの `setInputItemTransform` 関数や `inputItemTransform` 関数を用いて行う。

5) 入力メソッドから当該のクエリが発行されることがあることは確認される。このクエリに対して、当該段落のテキストに関連する情報とともに返す機構は、効果のほどは定かではないが、いちおう新たに実装した。この処理はユーザが観察可能な影響は及ぼさないものの、それなりのコストが掛かるので、ベネフィットとのバランスは考慮すべきである。

## 4 OnTheSpotの実装

### 4.1 イベント発生からの流れ

**Key pressイベントとinput methodイベント**  
ユーザが入力メソッドを使用してpreedit文字列の入力を始めると、一般のキー押下と同様に `key press` イベントが発生するが、これは特段

の処理がされることなく、新たなinput method イベントに引き継がれる<sup>6)</sup>。これはイベントハンドラGuiWorkArea::inputMethodEvent関数(QWidgetへのオーバーライド関数)によってGuiInputMethod::processPreedit関数に引き継がれ、主に以下のような操作を行う。

- ① **Input item geometryの再設定** アプリケーションやウィンドウの切り替え等によって、入力メソッドとアプリケーションの座標系の間にずれが生じる。両座標系の相対的な関係を保持しておき、補正するのはアプリケーション側の責任である。
- ② **確定文字列のBufferへの挿入** 確定部分はモデルであるBufferに反映させる。
- ③ **Preedit表示様式の情報取得** 編集段階・仕上げ段階がわかるような文字装飾が一般的に為される。仕上げ段階では、変換対象セグメントがわかるような文字装飾の指定が為される。受け取った情報に対してカスタマイズするような仕様とすることもできるが、現段階ではQt側が十分な装飾をしている限りにおいて(Qt5以前では不十分なことがある)、その指定をそのまま受け入れている。
- ④ **仮想キャレット(カーソル)のピクセル座標での表示位置の計算** この座標は、input methodからのクエリに対してQt::ImCursorRectangleとして渡され、変換候補ウィンドウの表示位置を特定するのに用いられる。適切な座標の計算はLyX側で行う。
- ⑤ **Qt::ImQueryInputに変化が生じたことをQInputMethodに通知** これによってinput

methodからのクエリを誘発し、情報を渡す。

- ⑥ **BufferView::processUpdateFlagsを実行**  
描画プロセスに移行する。

ここまでで行われるのは、preedit描画の前準備である。実際の描画は、paintイベント発生からの一連の流れによって行われる。preeditの描画は、文字入力以外にもウィンドウサイズの変更等、WorkAreaの画面表示の変更を必要とするような事象によって引き起こされるが、これらの処理は、paintイベント発生の段階から開始される。

#### Update flagの処理

BufferView::processUpdateFlags 関数がUpdate::Forceフラグを伴って呼ばれると、一般にWorkArea上では、画面上の改行位置の再計算や、インセットの再配置が必要となる。以下の順で処理を行う。

1. 段落の再構成
2. GuiWorkArea::scheduleRedrawの呼び出し
3. GuiWorkArea::viewport()->updateを呼び出して、GuiWorkArea::paintEvent()の呼び出しをスケジュール<sup>7)</sup>

LyXでは、本文を論理上は段落(Paragraphクラス)単位で管理しており、描画時に各行(Rowクラス)に分解しているが、paintイベントが起これば、S層においてTextMetrics::redoParagraphとTextMetrics::tokenizeParagraphが、対象となる段落をいったん分解して、Rowの再構成を行う。Rowは構造体Elementのベクトルとして

6) たとえば、MacOSの場合は(Qt source)/qtbase/src/plugins/platforms/cocoa/qnsview\_keys.mmにおける-handleKeyEvent:nseventを参照。

7) 実際のpaintイベントの呼び出しは、Qtがメインイベントループに戻ったときに為される。これにより、repaint()を直接呼び出すよりもフリッカーが少なくなるように最適化される。

構成されており、各Elementが、画面上の文字列やインセット、仮要素、空白、余白に対応している。これらは、それぞれSTRING型、INSET型、VIRTUAL型、SPACE型、MARGINSPACE型のElementとして取り扱われる。仮要素は、モデルには入り込まずビューだけで処理される要素であり、OnTheSpot様式の採用に伴って、preeditは仮要素のサブクラスであるPREEDIT型として、TextMetrics::tokenizeParagraphによる段落再構成の段階で挿入することとする。仮要素には、他に自動補完文字列がある。

**Paintイベント** Paintイベント発生以後のpreedit処理関連を中心にしたフローチャートを、付録Aに掲げる。BufferViewは各インセットの描画処理は、それぞれのインセットに委任して再帰的に処理される。ただし、MathInsetはやや取り扱いが異なるので、特殊な処理が必要である<sup>8)</sup>。数式内では一般に入力できる文字が定められているので、入力メソッドが介入すべきところは基本的にないが、MathInsetの中に再帰的に入れたテキスト関連インセット<sup>9)</sup>には対応しなくてはならない。

フローチャートAの「paintTextの処理」の項にあるとおり、ElementがSTRINGまたはVIRTUAL型の場合は、paintStringAndSel関数を用いてテキストの描画が為されるのに対して、ElementがINSET型の場合は、paintInset関数に入る。すなわち、インセットの描画は、各インセットがPainterクラスを用いてそれぞれに行う。結合子⑦からのフローに示されているとおり、数式以外のインセットに関しては、ここからの流れは再帰的な段落処理となる。し

たがって、数式以外のインセットにカーソルがある場合のpreedit文字列は、上記の「**Update flagの処理**」の項で仮要素として挿入しておけば、画面上に描画されることがわかる。

一方、数式インセットの場合、各インセットは、Insetクラスの派生クラスの種類によって分類される。数式内のテキストはMathRow内に含まれるInsetMathCharクラスの要素として扱われ、InsetMathHullクラス内のInsetMathGridクラスのセルに含まれるMathDataの一部となる。フローチャート上は、結合子⑦からのインセット型の判定において、InsetMathHullクラスと判定された場合の処理が行われる。詳細は結合子⑨からの流れとなるが、ここで、本文のテキストに対してRowクラスが行ったのと類似の操作が、MathRowクラスに対して行われる（数式インセット内で再帰的にインセットの処理が為されることに注意）。数式内テキストを入力メソッド対応にするために、そのMathRow内に含まれるInsetMathCharクラスから呼び出される各ツールをそれぞれ入力メソッド対応にし、新設した入力メソッド対応のGuiPainter::text関数が、QPainter::drawText関数を用いて描画する仕組みとする。

## 4.2 仮要素のデザイン

OnTheSpot様式の実装に際しては、構造体Elementのメンバ変数として、下表の2つの変数を新たに追加するものとする（lyx::Row.h）。

変数型	変数名	既定値
InputMethod*	im	nullptr
pos_type	char_format_index	

8) 現在、数式はインセットの一形態として扱われているが、元々は独立のオブジェクトであった。

9) たとえば、L<sup>A</sup>T<sub>E</sub>X コマンドの\mboxや\textなどに対応するものが関係する。

1行目は、運用上はInputMethod（I層）の派生クラスであるGuiInputMethodクラス（G層）のアドレスである。個別の描画関数では

BufferViewクラスに関する情報を持っておらず、したがってInputMethodクラスにアクセスできないので、このメンバ変数は、preedit文字列を描画する際の参照用に使用される。また、preedit文字列以外はこのメンバ変数を使用することはないので、preedit文字列以外のElementは、`im==nullptr`となるようにして、preedit文字列の判別用に使用する。2行目はpreedit中の各セグメントのIDを表す`pos_type`型すなわち`unsigned int`型変数の変数であり、そのセグメントの表示様式を特定化するのに用いられる。

ここでの`char_format_index`の実装のしかたが、G/I/S三層構造におけるポイントである。QInputMethodEventは、各セグメントの表示様式を指定するのに、QtライブラリのクラスであるQTextCharFormatクラスを使用する。描画機構ではこの情報をできるだけ尊重して使用することが推奨されている（Qt5およびQt6のQInputMethodEvent Classの参考文献[2]）<sup>10)</sup>。理想的には、QTextCharFormatをそのまま描画に使用したいので、`char_format_index`に該当する変数としてQTextCharFormat型を指定したい。しかしながら、構造体Elementの属するRowクラスはS層に属するので、Qtライブラリ依存のQTextCharFormat型は使用してはならない。この矛盾をどうやって解決すれば良いだろうか。

寓話として、あるテーマパーク『G』で、お

土産を両手いっぱいを持った状態で、ジェットコースター『S』に乗らなくてはならなくなった状況を考えてみれば良い（帰ろうと思ったのに子供にせがまれたか、はたまたパートナーのわがままに付き合わされたかという状況でしょう）。お土産を抱えたままジェットコースターに乗ることは禁止されている。ただし、こういうときのためにジェットコースター乗り場には、手荷物用ロッカールームが備え付けられており、各ロッカーにはキー`int`が付いている。ジェットコースター『S』には、`int`は持ち込んで良いことになっているので、それを持ったままライドを楽しめばよい。ライド中に誰かのお土産と交換したくなったときは、`int`を交換すれば良い（乗車中は危ないのでプラットフォームに帰ってきてからにしましょう）。帰ってきたときに、`int`で鍵を開ければ、無事テーマパークGのお土産を回収することができる。あるいはライド中に取り決めた交換の約束を現物レベルで実行することができる。

このように、S層に持ち込めないオブジェクトについては、G層に保管しておき、S層ではテーブルを参照する形で操作を行えば良い。G/I/S型の構造では、ビューでの操作は、G層に始まりG層に帰ってくる（図1参照）。G層に帰ってきた際に、S層で取り決めた操作を実行すれば良い。実際には、ロッカールームはG層であるGuiInputMethodクラスに`std::vector<QTextCharFormat>char_formats_`として設置する。S層に持ち込み禁止のQTextCharFormatはここに置いておく。ロッカーキーはこのベクトルの添え字である（`pos_type = unsigned int`型）。S層に属するElementを取り扱う際には、この添え字で以て参照する。

#### 4.3 数式内テキストに関する註釈

**数式の構造** 数式はInsetクラスの派生であ

10) なお、LyX 2.4までのpreedit表示様式は、LyX側で変換対象セグメントとそれ以外のセグメントを判定し、前者を実下線、後者を破下線で表示していた。表示様式はLyX側で固定させていたので、入力メソッド側からの様式指定を考慮する必要はなかった。Qt5までの様式指定は、参考文献での言及にも関わらず、不十分な点が見られたので、このような仕様になったのではないかと推測される。

るInsetMathクラスで取り扱われる。すなわち、数式もまたインセットである<sup>11)</sup>。InsetMathクラスからは、さらにInsetMathCharクラスやInsetMathFracクラスなど、数式の構成要素に対応する各クラスが派生する。実際の数式の構成においては、これらの数式構成要素クラスは、つねに多重構造を持つインセット群として構成される。とくに、各数式構成要素クラスはカーソル位置等の情報を持たないので、つねに(InsetMathクラスの派生クラスの一である)InsetMathNestsクラスオブジェクトの中に収められる。また、数式インセット群の多重構造の最も外部には、つねにInsetMathHullが置かれ、FormulaInsetやFormulaMacroInsetのようなそれを包含する一般インセットとのインタフェースとしての役割を担う<sup>12)</sup>。

数式内に通常の文字列を挿入する際、 $\text{\LaTeX}$ においては`\text{...}`コマンドや`\mbox{...}`コマンドが用いられる。LyX においては、これらの対応物は数式インセットの中の内部インセットとして扱われる。

**インセットとカーソル** 数式内テキストは、数式インセットの内部インセットとして表現される。この入れ子になったインセット内部の文字列にアクセスするためには、その位置情報が必要である。これには、DocIteratorクラスとその派生であるCursorクラスが使用される。一般に、LyX における文書構造は、入れ子構造を許容する一連の多重インセットの列として表現される。Bufferクラスで表現される文書それ自身も、DocIteratorあるいはCursorクラス内においての取り扱いは、インセットである。

インセットには、テキストモード(texted)と数式モード(mathed)がある。

DocIteratorクラスは、文書中のカーソル位置を表し、カーソルが多重インセット内にあるとき、各インセットレベルのカーソル位置を表すCursorSliceのベクトルを保持する。CursorSliceクラスは、特定のインセットレベルにおける位置情報を持ち、これらは下記のような関数でアクセスすることが出来る<sup>13)</sup>。

`pos()` テキストの場合、現段落中のカーソル位置を返す。数式の場合は、数式中のカーソル位置を返す。

`pit()` テキストの場合、一連の段落のうち、現段落のインデックスを返す。

`inset()` 現在カーソルが位置するインセット(カーソルを内包するインセット)を返す。ただし、InsetTabularクラスの場合は、InsetTableCellクラスで表されるセルインセットではなく、表本体を返す。

`idx()` インセットが表や行列の場合に、カーソルを含むセルを返す。その他のインセットの場合はつねに0を返す。

これらの関数はDocIteratorクラスも持っているが、DocIteratorクラスの場合は、内部スライスの情報を返す。外部スライスの情報は、`top()`関数を使用することでアクセスできる。また、選択範囲の始点を示すアンカーが、CursorData派生クラスのDocIteratorオブジェクトとして取得できる。

**数式と数式内テキストの構築** MathRowコンストラクタ中では、Elementsに次の順でエレメントが入れられ、空白の調整が為される。

11) ただし、数式構成要素のインセットは、内部処理において通常のインセットとは別扱いを受け、また数式外では使用できないので、まったく同じ訳ではない。

12) InsetMath.h L62-72コメント [1] 参照。

13) Cursor.h L12-51コメント [1] 参照。



1. MathClassがMC\_OPEN型のDUMMY要素
2. MathData引数 (addToMathRow (\*this, mi) 中で行われる)
3. MathClassがMC\_CLOSE型のDUMMY要素

インセットや文字の挿入は、DUMMY (MC\_OPEN) → INSET → ... → INSET → DUMMY (MC\_CLOSE) の順に行われる。文字1つはINSET 1つに対応する。インセットが入れ子になっている時は、入れ子がINSET (MARKER) で表され、その中がMC\_OPENとMC\_CLOSEで囲まれる。たとえば、本文が `a$y\text{x}$` のとき、CoordCache::Elementsの内容は、

```
DUMMY(NO_MARKER, MC_OPEN)
    INSET(char_=121('y'), NO_MARKER,
    MC_ORD)
    INSET(MARKER, MC_ORD, __data__=
    'text')
DUMMY(NO_MARKER, MC_OPEN)
    INSET(char_=120('x'), NO_
    MARKER, MC_ORD)
DUMMY(NO_MARKER, MC_CLOSE)
DUMMY(NO_MARKER, MC_CLOSE)
```

のようになる。

#### 4.4 インプットアイテム座標の補正

Qtライブラリにおいて、Input Methodのアイテム（変換候補ウィンドウ等）が持つインプットアイテム座標と、アプリケーションウィンドウが持つウィンドウ座標との相対的な位置関係は、インプットアイテムが画面上で移動すると、一般的にずれてしまう。これは、具体的

にはアプリケーション間のフォーカスの切替や、ポップアップウィンドウのポップアップやクローズ、ドロップダウンメニューの表示等によって起こる。これらの動作を行うと、両座標間の関係が切れてしまうので、元のウィンドウに戻ったときに、変換候補ウィンドウ等が正しくない位置に表示されるようになってしまう。これは、悪くすると変換候補ウィンドウが地の文を覆い隠して、正常な編集作業に支障を生じせしめるので補正しなくてはならない。そこで、両座標間の正常な相対的な位置関係を保存しておき、アプリケーションやウィンドウの切替が生じて復帰した際に、保存していた位置関係を復元するプロセスを設ける必要がある。

**ダイアログの表示** ダイアログを表示する仮想関数 `Dialog::showData(data)` に、ダイアログがフォーカスを得た理由（経緯）を渡す新たなバージョン `Dialog::showData(data, Qt::FocusReason reason)` を新設し、`Dialog::showView()` も理由を渡せるように、`Dialog::showView(reason)` に変更した。これは、ダイアログ `findreplaceadv`<sup>14)</sup> は、他のダイアログの場合と違い、相対座標に特別な扱いが必要となるためである。`reason == Qt::OtherFocusReason` をこのダイアログのために予約しておくこととした。

## 5 おわりに

LyX のウェブページを閲覧すると、新規参入の開発者向けに参考文献の紹介が為されるなどがえる。とくに学生にとっては、大規模なオープンソースプロジェクトに参加する経験は有益

14) LyX メニューにおける編集→検索・置換（詳細）で現れるダイアログに該当する。



であろうとおもわれる。プロジェクト継続の面からも新しい方々の参加を期待したい。

思えば、最初にワードプロセッサに触れたのは、学部1回生のときに属した政治学ゼミのゼミ論作成のときであった。指導の足立幸男先生からお借りしたワープロは、2行10文字列程度がウィンドウ表示される単漢字変換（そのうち1行は変換候補の表示に充てられる）のもので、おそらく当時でも非力になりつつあったモデルだと思われるが、とても画面を見ながら文面を考えるなど云う芸当のできるものではなく、清書専用機といった風情であった。修士論文作成のときには、ブラウン管モニタのフリッカーに30分ごとにノックアウトされ、息も絶え絶えに論文を書かなくてはならなかった。一日中画面を見ながら直接文章を書けるといういまの環境に感謝したい。

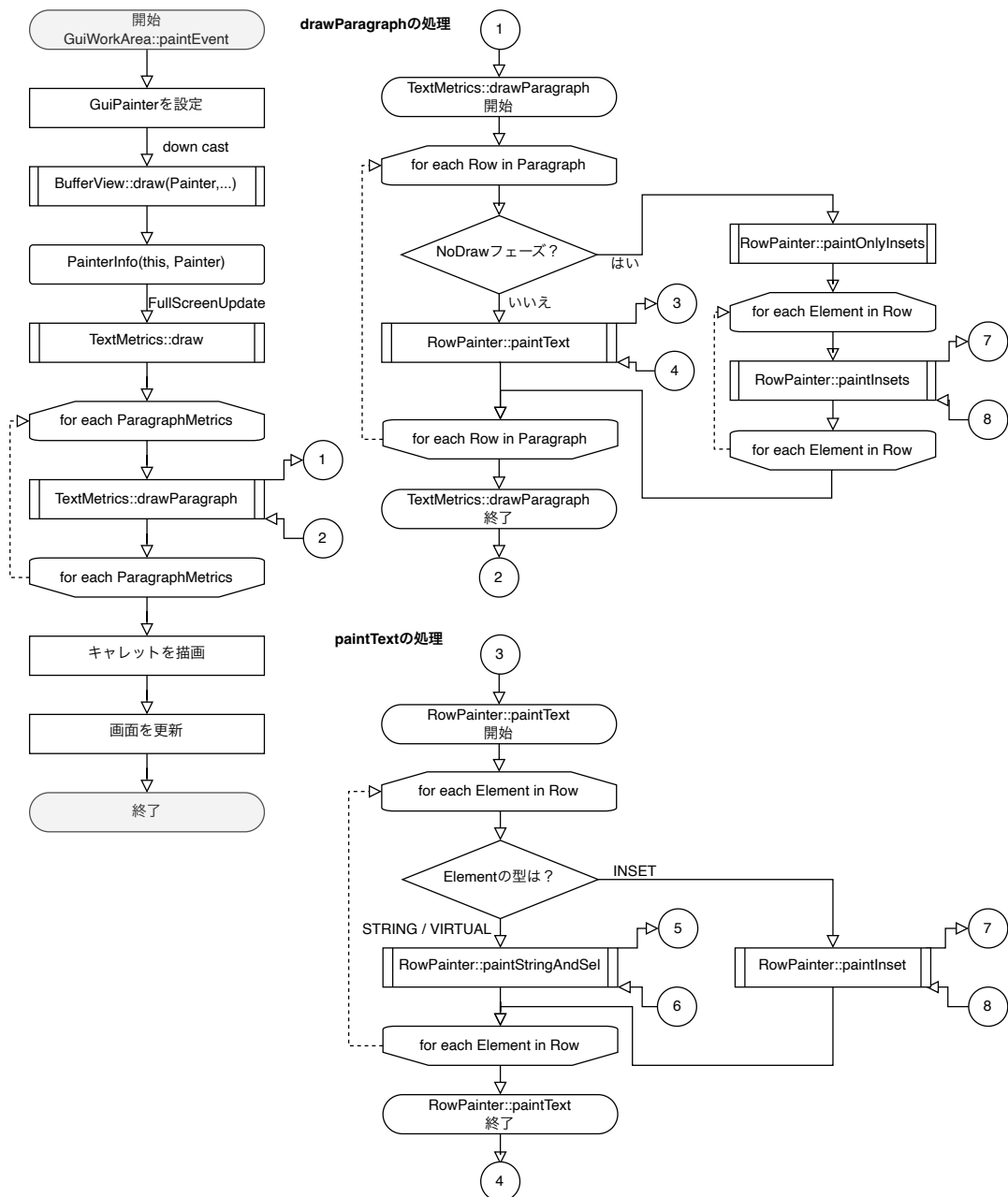
なお、本年度を以て、学部長を務めていた佐藤正市先生が退職される。職務を引き継ぐに当たり、さまざまなご指導ご鞭撻を賜った。ここに厚く御礼を申し上げたい。

## 参考文献

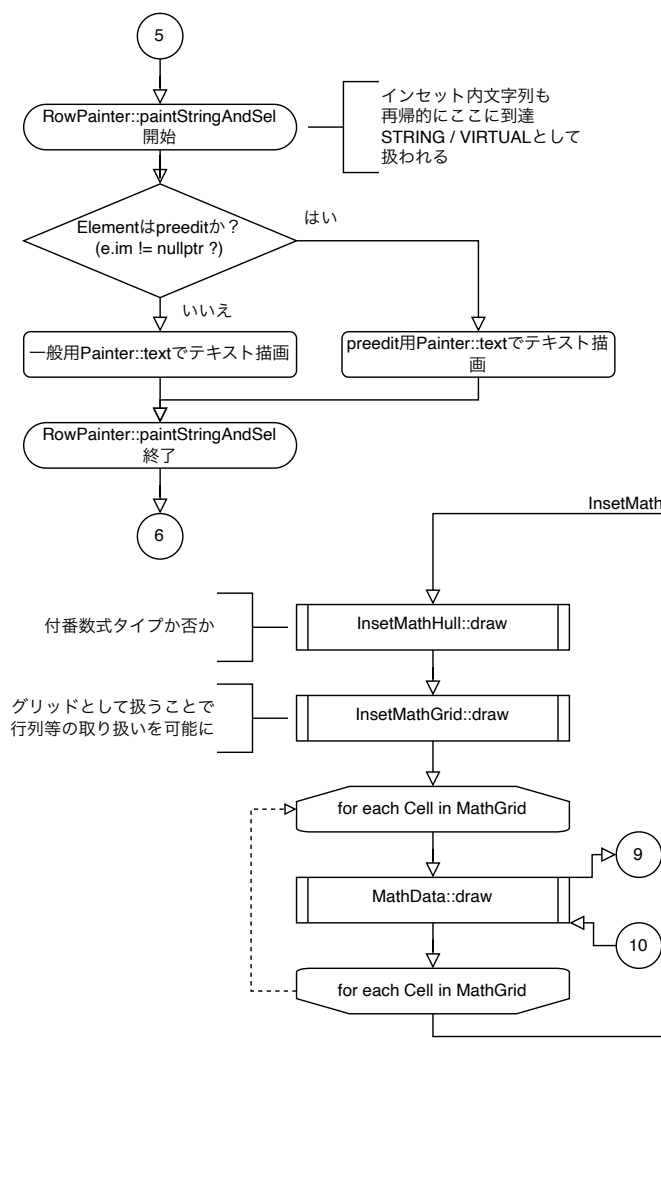
- [1] LyX master branchソース ([git.lyx.org](https://git.lyx.org), 2024年9月、LyX 2.5開発時点)
- [2] Qt C++ Class References for Qt 5 (<https://doc.qt.io/qt-5/classes.html>) and Qt 6 (<https://doc.qt.io/qt-6/classes.html>)
- [3] Glenn E. Krasner and Stephan T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3)26-49, August/September 1988.

## A フローチャート

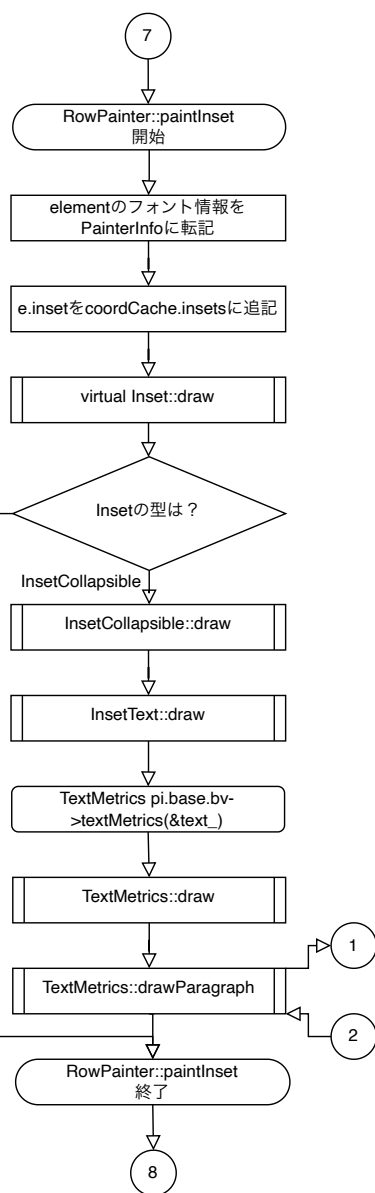
下図は、描画時の流れをpreedit文字列の描画に関連する部分を主に抽出したものである。



## paintStringAndSelの処理



## paintInsetの処理



```

graph TD
    Start((9)) --> MathDataDraw[MathData::draw]
    MathDataDraw --> MathRowDraw[MathRow::draw]
    MathRowDraw --> Loop1[for each Element in MathRow]
    Loop1 --> ElementType{Elementの型?}
    ElementType -- INSET --> VirtualInsetDraw[virtual Inset::draw]
    ElementType -- BOX --> DrawBox[ボックスを描く]
    ElementType -- DUMMY --> FormulaCompletion[数式補完処理]
    VirtualInsetDraw --> InsetType{Insetの型?}
    InsetType -- InsetMath --> InsetMathCharDraw[InsetMathChar::draw]
    InsetMathCharDraw --> PainterInfoDraw[PainterInfo::draw]
    PainterInfoDraw --> NoDrawPhase{NoDrawフェーズ?}
    NoDrawPhase -- はい --> NullPainterText[NullPainter::text]
    NoDrawPhase -- いいえ --> GuiPainterText[GuiPainter::text]
    InsetType -- InsetMathNest --> InsetMathFontDraw[InsetMathFont::draw]
    InsetMathFontDraw --> Cell[Cell]
    Cell --> MathDataDraw2[MathData::draw]
    MathDataDraw2 --> FormulaCompletion
    NullPainterText --> FormulaCompletion
    GuiPainterText --> FormulaCompletion
    FormulaCompletion --> Loop2[for each Element in MathRow]
    Loop2 --> ElementType

```

【変更点】  
e.im != nullptrのときに  
呼ぶInset::drawをInputMethod  
対応のものにする

数式内テキストの場合  
\text{()}, \mbox{()}等

【変更点】  
InputMethod  
対応にする

【変更点】  
InputMethod  
対応にする

【変更点】  
InputMethod  
対応にする

数式補完処理

for each Element in MathRow

10