```
/*
 * Copyright (C) 1999 Lars Knoll (knoll@kde.org)
 *           (C) 1999 Antti Koivisto (koivisto@kde.org)
 *           (C) 2007 David Smith (catfish.man@gmail.com)
 * Copyright (C) 2003-2023 Apple Inc. All rights reserved.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public License
 * along with this library; see the file COPYING.LIB.  If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301, USA.
 */

#pragma once

#include "CaretRectComputation.h"
#include "FloatingObjects.h"
#include "LegacyLineLayout.h"
#include "LineWidth.h"
#include "RenderBlock.h"
#include "RenderLineBoxList.h"
#include "TrailingObjects.h"
#include <memory>

namespace WebCore {

class LineBreaker;
class RenderMultiColumnFlow;
class RenderRubyRun;

namespace LayoutIntegration {
class LineLayout;
}

namespace InlineIterator {
class LineBoxIterator;
}

#if ENABLE(TEXT_AUTOSIZING)
enum LineCount {
    NOT_SET = 0, NO_LINE = 1, ONE_LINE = 2, MULTI_LINE = 3
};
```

```
#endif

class RenderBlockFlow : public RenderBlock {
    WTF_MAKE_ISO_ALLOCATED(RenderBlockFlow);
public:
    RenderBlockFlow(Type, Element&, RenderStyle&&, OptionSet<BlockFlowFlag> =
     { });
    RenderBlockFlow(Type, Document&, RenderStyle&&, OptionSet<BlockFlowFlag> =
     { });
    virtual ~RenderBlockFlow();

    void layoutBlock(bool relayoutChildren, LayoutUnit pageLogicalHeight =
     0_lu) override;

protected:
    void willBeDestroyed() override;

    // This method is called at the start of layout to wipe away all of the
     floats in our floating objects list. It also
    // repopulates the list with any floats that intrude from previous
     siblings or parents. Floats that were added by
    // descendants are gone when this call completes and will get added back
     later on after the children have gotten
    // a relayout.
    void rebuildFloatingObjectSetFromIntrudingFloats();

    // RenderBlockFlow always contains either lines or paragraphs. When the
     children are all blocks (e.g. paragraphs), we call layoutBlockChildren.
    // When the children are all inline (e.g., lines), we call
     layoutInlineChildren.
    void layoutInFlowChildren(bool relayoutChildren, LayoutUnit&
     repaintLogicalTop, LayoutUnit& repaintLogicalBottom, LayoutUnit&
     maxFloatLogicalBottom);
    void layoutBlockChildren(bool relayoutChildren, LayoutUnit&
     maxFloatLogicalBottom);
    void layoutInlineChildren(bool relayoutChildren, LayoutUnit&
     repaintLogicalTop, LayoutUnit& repaintLogicalBottom);

    void simplifiedNormalFlowLayout() override;
    LayoutUnit shiftForAlignContent(LayoutUnit intrinsicLogicalHeight,
     LayoutUnit& repaintLogicalTop, LayoutUnit& repaintLogicalBottom);

    // RenderBlockFlows override these methods, since they are the only class
     that supports margin collapsing.
    LayoutUnit collapsedMarginBefore() const final { return
     maxPositiveMarginBefore() - maxNegativeMarginBefore(); }
    LayoutUnit collapsedMarginAfter() const final { return
     maxPositiveMarginAfter() - maxNegativeMarginAfter(); }

    void dirtyLinesFromChangedChild(RenderObject& child) final
    {
```

```
        if (legacyLineLayout())
            legacyLineLayout()->lineBoxes().dirtyLinesFromChangedChild(*this,
             child);
    }

    void paintColumnRules(PaintInfo&, const LayoutPoint&) override;

public:
    class MarginValues {
    public:
        MarginValues(LayoutUnit beforePos, LayoutUnit beforeNeg, LayoutUnit
         afterPos, LayoutUnit afterNeg)
            : m_positiveMarginBefore(beforePos)
            , m_negativeMarginBefore(beforeNeg)
            , m_positiveMarginAfter(afterPos)
            , m_negativeMarginAfter(afterNeg)
        {
        }

        LayoutUnit positiveMarginBefore() const { return
         m_positiveMarginBefore; }
        LayoutUnit negativeMarginBefore() const { return
         m_negativeMarginBefore; }
        LayoutUnit positiveMarginAfter() const { return m_positiveMarginAfter;
         }
        LayoutUnit negativeMarginAfter() const { return m_negativeMarginAfter;
         }

        void setPositiveMarginBefore(LayoutUnit pos) { m_positiveMarginBefore
         = pos; }
        void setNegativeMarginBefore(LayoutUnit neg) { m_negativeMarginBefore
         = neg; }
        void setPositiveMarginAfter(LayoutUnit pos) { m_positiveMarginAfter =
         pos; }
        void setNegativeMarginAfter(LayoutUnit neg) { m_negativeMarginAfter =
         neg; }

    private:
        LayoutUnit m_positiveMarginBefore;
        LayoutUnit m_negativeMarginBefore;
        LayoutUnit m_positiveMarginAfter;
        LayoutUnit m_negativeMarginAfter;
    };
    MarginValues marginValuesForChild(RenderBox& child) const;

    // Allocated only when some of these fields have non-default values
    struct RenderBlockFlowRareData {
        WTF_MAKE_NONCOPYABLE(RenderBlockFlowRareData); WTF_MAKE_FAST_ALLOCATED;
    public:
        RenderBlockFlowRareData(const RenderBlockFlow& block)
```

```
        : m_margins(positiveMarginBeforeDefault(block),
         negativeMarginBeforeDefault(block),
         positiveMarginAfterDefault(block),
         negativeMarginAfterDefault(block))
        , m_lineBreakToAvoidWidow(-1)
        , m_didBreakAtLineToAvoidWidow(false)
    {
    }

    static LayoutUnit positiveMarginBeforeDefault(const RenderBlock& block)
    {
        return std::max<LayoutUnit>(block.marginBefore(), 0);
    }
    static LayoutUnit negativeMarginBeforeDefault(const RenderBlock& block)
    {
        return std::max<LayoutUnit>(-block.marginBefore(), 0);
    }
    static LayoutUnit positiveMarginAfterDefault(const RenderBlock& block)
    {
        return std::max<LayoutUnit>(block.marginAfter(), 0);
    }
    static LayoutUnit negativeMarginAfterDefault(const RenderBlock& block)
    {
        return std::max<LayoutUnit>(-block.marginAfter(), 0);
    }

    MarginValues m_margins;
    int m_lineBreakToAvoidWidow;

    SingleThreadWeakPtr<RenderMultiColumnFlow> m_multiColumnFlow;

    bool m_didBreakAtLineToAvoidWidow : 1;
};

class MarginInfo {
    // Collapsing flags for whether we can collapse our margins with our
     children's margins.
    bool m_canCollapseWithChildren : 1;
    bool m_canCollapseMarginBeforeWithChildren : 1;
    bool m_canCollapseMarginAfterWithChildren : 1;

    // Whether or not we are a quirky container, i.e., do we collapse away
     top and bottom
    // margins in our container. Table cells and the body are the common
     examples. We
    // also have a custom style property for Safari RSS to deal with
     TypePad blog articles.
    bool m_quirkContainer : 1;

    // This flag tracks whether we are still looking at child margins that
     can all collapse together at the beginning of a block.
```

```
    // They may or may not collapse with the top margin of the block
     (|m_canCollapseTopWithChildren| tells us that), but they will
    // always be collapsing with one another. This variable can remain set
     to true through multiple iterations
    // as long as we keep encountering self-collapsing blocks.
    bool m_atBeforeSideOfBlock : 1;

    // This flag is set when we know we're examining bottom margins and we
     know we're at the bottom of the block.
    bool m_atAfterSideOfBlock : 1;

    // These variables are used to detect quirky margins that we need to
     collapse away (in table cells
    // and in the body element).
    bool m_hasMarginBeforeQuirk : 1;
    bool m_hasMarginAfterQuirk : 1;
    bool m_determinedMarginBeforeQuirk : 1;

    // These flags track the previous maximal positive and negative
     margins.
    LayoutUnit m_positiveMargin;
    LayoutUnit m_negativeMargin;

public:
    MarginInfo(const RenderBlockFlow&, LayoutUnit beforeBorderPadding,
     LayoutUnit afterBorderPadding);

    void setAtBeforeSideOfBlock(bool b) { m_atBeforeSideOfBlock = b; }
    void setAtAfterSideOfBlock(bool b) { m_atAfterSideOfBlock = b; }
    void clearMargin()
    {
        m_positiveMargin = 0;
        m_negativeMargin = 0;
    }
    void setHasMarginBeforeQuirk(bool b) { m_hasMarginBeforeQuirk = b; }
    void setHasMarginAfterQuirk(bool b) { m_hasMarginAfterQuirk = b; }
    void setDeterminedMarginBeforeQuirk(bool b) {
     m_determinedMarginBeforeQuirk = b; }
    void setPositiveMargin(LayoutUnit p) { m_positiveMargin = p; }
    void setNegativeMargin(LayoutUnit n) { m_negativeMargin = n; }
    void setPositiveMarginIfLarger(LayoutUnit p)
    {
        if (p > m_positiveMargin)
            m_positiveMargin = p;
    }
    void setNegativeMarginIfLarger(LayoutUnit n)
    {
        if (n > m_negativeMargin)
            m_negativeMargin = n;
    }
```

```
    void setMargin(LayoutUnit p, LayoutUnit n) { m_positiveMargin = p;
     m_negativeMargin = n; }
    void setCanCollapseMarginAfterWithChildren(bool collapse) {
     m_canCollapseMarginAfterWithChildren = collapse; }

    bool atBeforeSideOfBlock() const { return m_atBeforeSideOfBlock; }
    bool canCollapseWithMarginBefore() const { return
     m_atBeforeSideOfBlock && m_canCollapseMarginBeforeWithChildren; }
    bool canCollapseWithMarginAfter() const { return m_atAfterSideOfBlock
     && m_canCollapseMarginAfterWithChildren; }
    bool canCollapseMarginBeforeWithChildren() const { return
     m_canCollapseMarginBeforeWithChildren; }
    bool canCollapseMarginAfterWithChildren() const { return
     m_canCollapseMarginAfterWithChildren; }
    bool quirkContainer() const { return m_quirkContainer; }
    bool determinedMarginBeforeQuirk() const { return
     m_determinedMarginBeforeQuirk; }
    bool hasMarginBeforeQuirk() const { return m_hasMarginBeforeQuirk; }
    bool hasMarginAfterQuirk() const { return m_hasMarginAfterQuirk; }
    LayoutUnit positiveMargin() const { return m_positiveMargin; }
    LayoutUnit negativeMargin() const { return m_negativeMargin; }
    LayoutUnit margin() const { return m_positiveMargin -
     m_negativeMargin; }
};

bool shouldTrimChildMargin(MarginTrimType, const RenderBox&) const;

void layoutBlockChild(RenderBox& child, MarginInfo&, LayoutUnit&
 previousFloatLogicalBottom, LayoutUnit& maxFloatLogicalBottom);
void adjustPositionedBlock(RenderBox& child, const MarginInfo&);
void adjustFloatingBlock(const MarginInfo&);

void trimBlockEndChildrenMargins();

void setStaticInlinePositionForChild(RenderBox& child, LayoutUnit
 blockOffset, LayoutUnit inlinePosition);
void updateStaticInlinePositionForChild(RenderBox& child, LayoutUnit
 logicalTop, IndentTextOrNot shouldIndentText);

LayoutUnit startAlignedOffsetForLine(LayoutUnit position, IndentTextOrNot);

LayoutUnit collapseMargins(RenderBox& child, MarginInfo&);
LayoutUnit collapseMarginsWithChildInfo(RenderBox* child, RenderObject*
 prevSibling, MarginInfo&);

LayoutUnit clearFloatsIfNeeded(RenderBox& child, MarginInfo&, LayoutUnit
 oldTopPosMargin, LayoutUnit oldTopNegMargin, LayoutUnit yPos);
LayoutUnit estimateLogicalTopPosition(RenderBox& child, const MarginInfo&,
 LayoutUnit& estimateWithoutPagination);
void marginBeforeEstimateForChild(RenderBox&, LayoutUnit&, LayoutUnit&)
 const;
```

```
void handleAfterSideOfBlock(LayoutUnit top, LayoutUnit bottom,
 MarginInfo&);
void setCollapsedBottomMargin(const MarginInfo&);

bool childrenPreventSelfCollapsing() const final;

bool shouldBreakAtLineToAvoidWidow() const { return hasRareBlockFlowData()
 && rareBlockFlowData()->m_lineBreakToAvoidWidow >= 0; }
void clearShouldBreakAtLineToAvoidWidow() const;
int lineBreakToAvoidWidow() const { return hasRareBlockFlowData() ?
 rareBlockFlowData()->m_lineBreakToAvoidWidow : -1; }
void setBreakAtLineToAvoidWidow(int);
void clearDidBreakAtLineToAvoidWidow();
void setDidBreakAtLineToAvoidWidow();
bool didBreakAtLineToAvoidWidow() const { return hasRareBlockFlowData() &&
 rareBlockFlowData()->m_didBreakAtLineToAvoidWidow; }

RenderMultiColumnFlow* multiColumnFlow() const { return
 hasRareBlockFlowData() ? multiColumnFlowSlowCase() : nullptr; }
RenderMultiColumnFlow* multiColumnFlowSlowCase() const;
void setMultiColumnFlow(RenderMultiColumnFlow&);
void clearMultiColumnFlow();
bool willCreateColumns(std::optional<unsigned> desiredColumnCount =
 std::nullopt) const;
virtual bool requiresColumns(int) const;

bool containsFloats() const override { return m_floatingObjects &&
 !m_floatingObjects->set().isEmpty(); }
bool containsFloat(RenderBox&) const;
bool subtreeContainsFloats() const;
bool subtreeContainsFloat(RenderBox&) const;

void deleteLines() override;
void computeOverflow(LayoutUnit oldClientAfterEdge, bool recomputeFloats =
 false) override;
Position positionForPoint(const LayoutPoint&) override;
VisiblePosition positionForPoint(const LayoutPoint&, const
 RenderFragmentContainer*) override;

LayoutUnit lowestFloatLogicalBottom(FloatingObject::Type =
 FloatingObject::FloatLeftRight) const;

void removeFloatingObjects();
void markAllDescendantsWithFloatsForLayout(RenderBox* floatToRemove =
 nullptr, bool inLayout = true);
void markSiblingsWithFloatsForLayout(RenderBox* floatToRemove = nullptr);

const FloatingObjectSet* floatingObjectSet() const { return
 m_floatingObjects ? &m_floatingObjects->set() : nullptr; }

FloatingObject& insertFloatingObjectForIFC(RenderBox&);
```

```
LayoutUnit logicalTopForFloat(const FloatingObject& floatingObject) const
 { return isHorizontalWritingMode() ? floatingObject.y() :
 floatingObject.x(); }
LayoutUnit logicalBottomForFloat(const FloatingObject& floatingObject)
 const { return isHorizontalWritingMode() ? floatingObject.maxY() :
 floatingObject.maxX(); }
LayoutUnit logicalLeftForFloat(const FloatingObject& floatingObject) const
 { return isHorizontalWritingMode() ? floatingObject.x() :
 floatingObject.y(); }
LayoutUnit logicalRightForFloat(const FloatingObject& floatingObject)
 const { return isHorizontalWritingMode() ? floatingObject.maxX() :
 floatingObject.maxY(); }
LayoutUnit logicalWidthForFloat(const FloatingObject& floatingObject)
 const { return isHorizontalWritingMode() ? floatingObject.width() :
 floatingObject.height(); }
LayoutUnit logicalHeightForFloat(const FloatingObject& floatingObject)
 const { return isHorizontalWritingMode() ? floatingObject.height() :
 floatingObject.width(); }

void setLogicalTopForFloat(FloatingObject& floatingObject, LayoutUnit
 logicalTop)
{
    if (isHorizontalWritingMode())
        floatingObject.setY(logicalTop);
    else
        floatingObject.setX(logicalTop);
}
void setLogicalLeftForFloat(FloatingObject& floatingObject, LayoutUnit
 logicalLeft)
{
    if (isHorizontalWritingMode())
        floatingObject.setX(logicalLeft);
    else
        floatingObject.setY(logicalLeft);
}
void setLogicalHeightForFloat(FloatingObject& floatingObject, LayoutUnit
 logicalHeight)
{
    if (isHorizontalWritingMode())
        floatingObject.setHeight(logicalHeight);
    else
        floatingObject.setWidth(logicalHeight);
}
void setLogicalWidthForFloat(FloatingObject& floatingObject, LayoutUnit
 logicalWidth)
{
    if (isHorizontalWritingMode())
        floatingObject.setWidth(logicalWidth);
    else
        floatingObject.setHeight(logicalWidth);
```

```
    }
    void setLogicalMarginsForFloat(FloatingObject& floatingObject, LayoutUnit
     logicalLeftMargin, LayoutUnit logicalBeforeMargin)
    {
        if (isHorizontalWritingMode())
            floatingObject.setMarginOffset(LayoutSize(logicalLeftMargin,
             logicalBeforeMargin));
        else
            floatingObject.setMarginOffset(LayoutSize(logicalBeforeMargin,
             logicalLeftMargin));
    }

    LayoutPoint flipFloatForWritingModeForChild(const FloatingObject&, const
     LayoutPoint&) const;

    LegacyRootInlineBox* firstRootBox() const { return legacyLineLayout() ?
     legacyLineLayout()->firstRootBox() : nullptr; }
    LegacyRootInlineBox* lastRootBox() const { return legacyLineLayout() ?
     legacyLineLayout()->lastRootBox() : nullptr; }

    void setChildrenInline(bool) final;

    bool hasLines() const;
    void invalidateLineLayoutPath() final;
    void computeAndSetLineLayoutPath();

    enum LineLayoutPath { UndeterminedPath = 0, ModernPath, LegacyPath,
     ForcedLegacyPath };
    LineLayoutPath lineLayoutPath() const { return
     static_cast<LineLayoutPath>(renderBlockFlowLineLayoutPath()); }
    void setLineLayoutPath(LineLayoutPath path) {
     setRenderBlockFlowLineLayoutPath(path); }

    int lineCount() const;

    void setHasMarkupTruncation(bool b) {
     setRenderBlockFlowHasMarkupTruncation(b); }
    bool hasMarkupTruncation() const { return
     renderBlockFlowHasMarkupTruncation(); }

    bool containsNonZeroBidiLevel() const;

    const LegacyLineLayout* legacyLineLayout() const;
    LegacyLineLayout* legacyLineLayout();
    const LayoutIntegration::LineLayout* modernLineLayout() const;
    LayoutIntegration::LineLayout* modernLineLayout();

#if ENABLE(TREE_DEBUGGING)
    void outputFloatingObjects(WTF::TextStream&, int depth) const;
    void outputLineTreeAndMark(WTF::TextStream&, const LegacyInlineBox*
     markedBox, int depth) const;
```

```
#endif

    // Returns the logicalOffset at the top of the next page. If the offset
     passed in is already at the top of the current page,
    // then nextPageLogicalTop with ExcludePageBoundary will still move to the
     top of the next page. nextPageLogicalTop with
    // IncludePageBoundary set will not.
    //
    // For a page height of 800px, the first rule will return 800 if the value
     passed in is 0. The second rule will simply return 0.
    enum PageBoundaryRule { ExcludePageBoundary, IncludePageBoundary };
    LayoutUnit nextPageLogicalTop(LayoutUnit logicalOffset, PageBoundaryRule =
     ExcludePageBoundary) const;
    LayoutUnit pageLogicalTopForOffset(LayoutUnit offset) const;
    LayoutUnit pageLogicalHeightForOffset(LayoutUnit offset) const;
    LayoutUnit pageRemainingLogicalHeightForOffset(LayoutUnit offset,
     PageBoundaryRule = IncludePageBoundary) const;
    LayoutUnit logicalHeightForChildForFragmentation(const RenderBox& child)
     const;
    bool hasNextPage(LayoutUnit logicalOffset, PageBoundaryRule =
     ExcludePageBoundary) const;

    void updateColumnProgressionFromStyle(const RenderStyle&);
    void updateStylesForColumnChildren(const RenderStyle* oldStyle);

    bool needsLayoutAfterFragmentRangeChange() const override;
    WEBCORE_EXPORT RenderText* findClosestTextAtAbsolutePoint(const
     FloatPoint&);

    // A page break is required at some offset due to space shortage in the
     current fragmentainer.
    void setPageBreak(LayoutUnit offset, LayoutUnit spaceShortage);
    // Update minimum page height required to avoid fragmentation where it
     shouldn't occur (inside
    // unbreakable content, between orphans and widows, etc.). This will be
     used as a hint to the
    // column balancer to help set a good minimum column height.
    void updateMinimumPageHeight(LayoutUnit offset, LayoutUnit minHeight);

    void adjustSizeContainmentChildForPagination(RenderBox& child, LayoutUnit
     offset);

    void addFloatsToNewParent(RenderBlockFlow& toBlockFlow) const;

    inline LayoutUnit endPaddingWidthForCaret() const;

    LayoutUnit adjustEnclosingTopForPrecedingBlock(LayoutUnit top) const;

    std::optional<LayoutUnit> lowestInitialLetterLogicalBottom() const;

protected:
```

```
bool isChildEligibleForMarginTrim(MarginTrimType, const RenderBox&) const
 final;

bool shouldResetLogicalHeightBeforeLayout() const override { return true; }

void computeIntrinsicLogicalWidths(LayoutUnit& minLogicalWidth,
 LayoutUnit& maxLogicalWidth) const override;

bool pushToNextPageWithMinimumLogicalHeight(LayoutUnit& adjustment,
 LayoutUnit logicalOffset, LayoutUnit minimumLogicalHeight) const;

// If the child is unsplittable and can't fit on the current page, return
 the top of the next page/column.
LayoutUnit adjustForUnsplittableChild(RenderBox& child, LayoutUnit
 logicalOffset, LayoutUnit beforeMargin = 0_lu, LayoutUnit afterMargin =
 0_lu);
LayoutUnit adjustBlockChildForPagination(LayoutUnit logicalTopAfterClear,
 LayoutUnit estimateWithoutPagination, RenderBox& child, bool
 atBeforeSideOfBlock);
LayoutUnit applyBeforeBreak(RenderBox& child, LayoutUnit logicalOffset);
 // If the child has a before break, then return a new yPos that shifts to
 the top of the next page/column.
LayoutUnit applyAfterBreak(RenderBox& child, LayoutUnit logicalOffset,
 MarginInfo&); // If the child has an after break, then return a new
 offset that shifts to the top of the next page/column.

LayoutUnit maxPositiveMarginBefore() const { return hasRareBlockFlowData()
 ? rareBlockFlowData()->m_margins.positiveMarginBefore() :
 RenderBlockFlowRareData::positiveMarginBeforeDefault(*this); }
LayoutUnit maxNegativeMarginBefore() const { return hasRareBlockFlowData()
 ? rareBlockFlowData()->m_margins.negativeMarginBefore() :
 RenderBlockFlowRareData::negativeMarginBeforeDefault(*this); }
LayoutUnit maxPositiveMarginAfter() const { return hasRareBlockFlowData()
 ? rareBlockFlowData()->m_margins.positiveMarginAfter() :
 RenderBlockFlowRareData::positiveMarginAfterDefault(*this); }
LayoutUnit maxNegativeMarginAfter() const { return hasRareBlockFlowData()
 ? rareBlockFlowData()->m_margins.negativeMarginAfter() :
 RenderBlockFlowRareData::negativeMarginAfterDefault(*this); }

void initMaxMarginValues()
{
    if (!hasRareBlockFlowData())
        return;

    rareBlockFlowData()->m_margins =
     MarginValues(RenderBlockFlowRareData::positiveMarginBeforeDefault
     (*this) , RenderBlockFlowRareData::negativeMarginBeforeDefault(*this),
        RenderBlockFlowRareData::positiveMarginAfterDefault(*this),
         RenderBlockFlowRareData::negativeMarginAfterDefault(*this));
}
```

```
    void setMaxMarginBeforeValues(LayoutUnit pos, LayoutUnit neg);
    void setMaxMarginAfterValues(LayoutUnit pos, LayoutUnit neg);

    void styleWillChange(StyleDifference, const RenderStyle& newStyle)
     override;
    void styleDidChange(StyleDifference, const RenderStyle* oldStyle) override;

    void createFloatingObjects();

    std::optional<LayoutUnit> firstLineBaseline() const override;
    std::optional<LayoutUnit> lastLineBaseline() const override;
    std::optional<LayoutUnit> inlineBlockBaseline(LineDirectionMode) const
     override;

    void setComputedColumnCountAndWidth(int, LayoutUnit);

    LayoutUnit computedColumnWidth() const;
    unsigned computedColumnCount() const;

    bool isTopLayoutOverflowAllowed() const override;
    bool isLeftLayoutOverflowAllowed() const override;

    virtual void computeColumnCountAndWidth();

    virtual void cachePriorCharactersIfNeeded(const
     CachedLineBreakIteratorFactory&) { }

protected:
    // Called to lay out the legend for a fieldset or the ruby text of a ruby
     run. Also used by multi-column layout to handle
    // the flow thread child.
    void layoutExcludedChildren(bool relayoutChildren) override;
    void addOverflowFromFloats();

private:
    bool recomputeLogicalWidthAndColumnWidth();
    LayoutUnit columnGap() const;

    RenderBlockFlow* previousSiblingWithOverhangingFloats(bool&
     parentHasFloats) const;

    void checkForPaginationLogicalHeightChange(bool& relayoutChildren,
     LayoutUnit& pageLogicalHeight, bool& pageLogicalHeightChanged);

    void paintInlineChildren(PaintInfo&, const LayoutPoint&) override;
    void paintFloats(PaintInfo&, const LayoutPoint&, bool preservePhase =
     false) override;

    void repaintOverhangingFloats(bool paintAllDescendants) final;
    void clipOutFloatingObjects(RenderBlock&, const PaintInfo*, const
     LayoutPoint&, const LayoutSize&) override;
```

```
FloatingObject* insertFloatingObject(RenderBox&);
void removeFloatingObject(RenderBox&);
void removeFloatingObjectsBelow(FloatingObject*, int logicalOffset);
void computeLogicalLocationForFloat(FloatingObject&, LayoutUnit&
 logicalTopOffset);

// Called from lineWidth, to position the floats added in the last line.
// Returns true if and only if it has positioned any floats.
bool positionNewFloats();
void clearFloats(UsedClear);
FloatingObjects* floatingObjects() { return m_floatingObjects.get(); }

LayoutUnit logicalRightFloatOffsetForLine(LayoutUnit logicalTop,
 LayoutUnit fixedOffset, LayoutUnit logicalHeight) const override;
LayoutUnit logicalLeftFloatOffsetForLine(LayoutUnit logicalTop, LayoutUnit
 fixedOffset, LayoutUnit logicalHeight) const override;

LayoutUnit logicalRightOffsetForPositioningFloat(LayoutUnit logicalTop,
 LayoutUnit fixedOffset, bool applyTextIndent, LayoutUnit*
 heightRemaining) const;
LayoutUnit logicalLeftOffsetForPositioningFloat(LayoutUnit logicalTop,
 LayoutUnit fixedOffset, bool applyTextIndent, LayoutUnit*
 heightRemaining) const;

LayoutUnit nextFloatLogicalBottomBelow(LayoutUnit) const;
LayoutUnit nextFloatLogicalBottomBelowForBlock(LayoutUnit) const;

LayoutUnit addOverhangingFloats(RenderBlockFlow& child, bool
 makeChildPaintOtherFloats);
bool hasOverhangingFloat(RenderBox&);
void addIntrudingFloats(RenderBlockFlow* prev, RenderBlockFlow* container,
 LayoutUnit xoffset, LayoutUnit yoffset);
inline bool hasOverhangingFloats() const;
LayoutUnit getClearDelta(RenderBox& child, LayoutUnit yPos);

void determineLogicalLeftPositionForChild(RenderBox& child,
 ApplyLayoutDeltaMode = DoNotApplyLayoutDelta);

bool hitTestFloats(const HitTestRequest&, HitTestResult&, const
 HitTestLocation& locationInContainer, const LayoutPoint&
 accumulatedOffset) override;
bool hitTestInlineChildren(const HitTestRequest&, HitTestResult&, const
 HitTestLocation& locationInContainer, const LayoutPoint&
 accumulatedOffset, HitTestAction) override;

void addOverflowFromInlineChildren() override;

void markLinesDirtyInBlockRange(LayoutUnit logicalTop, LayoutUnit
 logicalBottom, LegacyRootInlineBox* highest = 0);
```

```
    GapRects inlineSelectionGaps(RenderBlock& rootBlock, const LayoutPoint&
     rootBlockPhysicalPosition, const LayoutSize& offsetFromRootBlock,
        LayoutUnit& lastLogicalTop, LayoutUnit& lastLogicalLeft, LayoutUnit&
         lastLogicalRight, const LogicalSelectionOffsetCaches&, const
         PaintInfo*) override;

    VisiblePosition positionForPointWithInlineChildren(const LayoutPoint&
     pointInLogicalContents, const RenderFragmentContainer*) override;
    void addFocusRingRectsForInlineChildren(Vector<LayoutRect>& rects, const
     LayoutPoint& additionalOffset, const RenderLayerModelObject*) const
     override;

public:
    virtual std::optional<TextAlignMode> overrideTextAlignmentForLine(bool /*
     endsWithSoftBreak */) const { return { }; }
    virtual void adjustInlineDirectionLineBounds(int /*
     expansionOpportunityCount */, float& /* logicalLeft */, float& /*
     logicalWidth */) const { }

private:
    bool hasLineLayout() const;
    bool hasLegacyLineLayout() const;

    bool hasModernLineLayout() const;
    void layoutModernLines(bool relayoutChildren, LayoutUnit&
     repaintLogicalTop, LayoutUnit& repaintLogicalBottom);
    bool tryComputePreferredWidthsUsingModernPath(LayoutUnit& minLogicalWidth,
     LayoutUnit& maxLogicalWidth);
    void setStaticPositionsForSimpleOutOfFlowContent();

    void adjustIntrinsicLogicalWidthsForColumns(LayoutUnit& minLogicalWidth,
     LayoutUnit& maxLogicalWidth) const;
    void computeInlinePreferredLogicalWidths(LayoutUnit& minLogicalWidth,
     LayoutUnit& maxLogicalWidth) const;
    void adjustInitialLetterPosition(RenderBox& childBox, LayoutUnit&
     logicalTopOffset, LayoutUnit& marginBeforeOffset);

    void setTextBoxTrimForSubtree(const RenderBlockFlow*
     inlineFormattingContextRootForTextBoxTrimEnd = nullptr);
    void adjustTextBoxTrimAfterLayout();

#if ENABLE(TEXT_AUTOSIZING)
    int m_widthForTextAutosizing;
    unsigned m_lineCountForTextAutosizing : 2;
#endif
    // FIXME: This is temporary until after we remove the forced "line layout
     codepath" invalidation.
    std::optional<LayoutUnit>
     m_previousModernLineLayoutContentBoxLogicalHeight;
```

```
    std::optional<LayoutUnit>
     selfCollapsingMarginBeforeWithClear(RenderObject* candidate);

public:
    // Computes a deltaOffset value that put a line at the top of the next
     page if it doesn't fit on the current page.
    void adjustLinePositionForPagination(LegacyRootInlineBox*, LayoutUnit&
     deltaOffset);

    struct LinePaginationAdjustment {
        LayoutUnit strut { 0_lu };
        bool isFirstAfterPageBreak { false };
    };
    LinePaginationAdjustment computeLineAdjustmentForPagination(const
     InlineIterator::LineBoxIterator&, LayoutUnit deltaOffset, LayoutUnit
     floatMinimumBottom = { });
    bool relayoutForPagination();

    bool hasRareBlockFlowData() const { return m_rareBlockFlowData.get(); }
    RenderBlockFlowRareData* rareBlockFlowData() const {
     ASSERT_WITH_SECURITY_IMPLICATION(hasRareBlockFlowData()); return
     m_rareBlockFlowData.get(); }
    RenderBlockFlowRareData& ensureRareBlockFlowData();
    void materializeRareBlockFlowData();

#if ENABLE(TEXT_AUTOSIZING)
    void adjustComputedFontSizes(float size, float visibleWidth);
    void resetComputedFontSize()
    {
        m_widthForTextAutosizing = -1;
        m_lineCountForTextAutosizing = NOT_SET;
    }
#endif

protected:
    std::unique_ptr<FloatingObjects> m_floatingObjects;
    std::unique_ptr<RenderBlockFlowRareData> m_rareBlockFlowData;

private:
    std::variant<
        std::monostate,
        std::unique_ptr<LayoutIntegration::LineLayout>,
        std::unique_ptr<LegacyLineLayout>
    > m_lineLayout;

    friend class LineBreaker;
    friend class LineWidth; // Needs to know FloatingObject
    friend class LegacyLineLayout;
};

inline bool RenderBlockFlow::hasLineLayout() const
```

```
{
    return !std::holds_alternative<std::monostate>(m_lineLayout);
}

inline bool RenderBlockFlow::hasLegacyLineLayout() const
{
    return
     std::holds_alternative<std::unique_ptr<LegacyLineLayout>>(m_lineLayout);
}

inline const LegacyLineLayout* RenderBlockFlow::legacyLineLayout() const
{
    return hasLegacyLineLayout() ?
     std::get<std::unique_ptr<LegacyLineLayout>>(m_lineLayout).get() : nullptr;
}

inline LegacyLineLayout* RenderBlockFlow::legacyLineLayout()
{
    return hasLegacyLineLayout() ?
     std::get<std::unique_ptr<LegacyLineLayout>>(m_lineLayout).get() : nullptr;
}

inline bool RenderBlockFlow::hasModernLineLayout() const
{
    return
     std::holds_alternative<std::unique_ptr<LayoutIntegration::LineLayout>>
     (m_lineLayout);
}

inline const LayoutIntegration::LineLayout*
 RenderBlockFlow::modernLineLayout() const
{
    return hasModernLineLayout() ?
     std::get<std::unique_ptr<LayoutIntegration::LineLayout>>(m_lineLayout)
     .get() : nullptr;
}

inline LayoutIntegration::LineLayout* RenderBlockFlow::modernLineLayout()
{
    return hasModernLineLayout() ?
     std::get<std::unique_ptr<LayoutIntegration::LineLayout>>(m_lineLayout)
     .get() : nullptr;
}

} // namespace WebCore

SPECIALIZE_TYPE_TRAITS_RENDER_OBJECT(RenderBlockFlow, isRenderBlockFlow())
```