How repaint works

Whenever something happens that should cause the screen contents to change, a repaint must be scheduled.

The way this is done is by calling one of several repaint functions, which tell the OS that a certain rectangle has been invalidated (=needs to be repainted), so that it will trigger a paint event, which will do the actual painting (unless the window is not visible for some reason, e.g. minimized or covered by another window).

Note that calling repaint on a certain renderer will potentially repaint other renderers too, if they cover the same rectangle on the screen.

Repaint functions

- `checkForRepaintDuringLayout()`

  In general, the repaint functions are called during layout, because a layout is the most common reason why something has to be repainted. However, you should only schedule a repaint if `checkForRepaintDuringLayout()` returns true. *TODO: Why is this the case? What does this function do? Is it for FOUC?*

- `repaint()`
  The most generic repaint function is `RenderObject::repaint()`. This simply invalidates the current rectangle of the frame (to be exact, the visual overflow rect). This can be useful, but the more common way is:

- `LayoutRepainter`

  LayoutRepainter is a helper class for use in your layout()/layoutBlock() function. You'd create it on the stack at the start of your function:

  ```
  LayoutRepainter repainter(*this, checkForRepaintDuringLayout());
  ```

  And at the end, before calling `setNeedsLayout(false)`, you call:

  ```
  repainter.repaintAfterLayout();
  ```

  This class ensures that both the old and the new rectangle of the renderer get invalidated -- when the `RenderObject` moves, both its old and new position need repainting.

  Internally, this class calls `repaintAfterLayoutIfNeeded`.

- `repaintDuringLayoutIfMoved()`

  Sometimes, a renderer moves a child during its layout after it has already called `child->layout()`. In this case, unless the parent will get repainted anyway, it may be

necessary to explicitly invalidate the child's rectangle, which is done using this function. You have to make sure to store the old child's rectangle before moving it:

```
LayoutRect oldRect = child->frameRect();

// Move child, child->setLocation(...), etc

child->repaintDuringLayoutIfMoved(oldRect);
```

For an example, see `RenderBlock::layoutBlockChild` - this function tries to not call `repaintDuringLayoutIfMoved` in the common case (by setting the position before calling `child->layout()`, so that the child's `layout()` would call repaint), but it does call `repaintDuringLayoutIfMoved` when it has to.

It should be noted that during layout, repaint notifications to the OS are actually deferred until layout is complete. This is done in `FrameView::layout()`, which calls `beginDeferredRepaints()` and `endDeferredRepaints()` around the call to `root->layout()`. The OS will not be notified about the invalidation rects until `endDeferredRepaints()` is called. At that point, the union of all rects that were invalidated during layout will be sent to the OS.

Non-layout repaints

Sometimes, a repaint is necessary even though no layout needs to be done. For example, this happens when an element's color (or background-color, etc) changes. In that case, the repaint will be scheduled by `styleWillChange`, e.g. `RenderLayerModelObject::styleWillChange`:

```
} else if (diff == StyleDifferenceRepaint || /* ... */)
    repaint();
```