3.5. Manipulating the page

You can manipulate the browser page in one of two ways. The preferred way is to treat it as a panel—where GWT looks after you a lot—or via the DOM directly—the Wild West approach. We'll look at both of these in this section, starting with the preferred approach.

3.5.1. Using the RootPanel/RootLayoutPanel

Manipulating the browser page to show your widgets is surprisingly easy, and you already know how to do that. The HTML page is generally treated as a panel—the `RootPanel` —and you use that panel's `add` method.

Definition

The `RootPanel` provides access to the underlying HTML page.

You can't create a `RootPanel` in your code; rather you get access to it by calling `Root-Panel` 's static `get` method. Accessing the whole HTML page is achieved by calling

```
RootPanel.get()
```

You use that approach to place the feedback link in the `BasicProject` onto the page:

```
RootPanel.get().add(feedback)
```

`RootPanel` is a subclass of `AbsolutePanel` , which allows you to give absolute coordinates for the widget, so you could write `add(feedback, 400,200)` to place the feedback panel in an absolute position if you wanted to. But in `BasicProject` we decide to position the panel using CSS styling, as you'll see later, so you use the `add` method without parameters.

The first time you call the `get` method in `RootPanel` it links your application into the window-closing event of the browser. This allows GWT to clean up and remove widgets and events from the DOM before the browser window closes—one way GWT works hard in the background to prevent memory leaks and problems for you.

You can also use `RootPanel` to access defined segments in the HTML by supplying a `String` that matches the `id` attribute —remember you want to place a logo image into a specifically defined slot in the HTML and you defined that location by giving a `div` element an `id` ; back in listing 3.1 we had:

```
<div id="logo"></div>
```

To access that part of the HTML and insert the logo, you pass in the element's `id` to the `get` method and check to see if the result isn't equal to `null` (if GWT can't find the element, then the `RootPanel.get(element)` method will return a `null`):

```
Image logo = new Image(GWT.getModuleBaseUrl+"../logo.png");

RootPanel logoSlot = RootPanel.get("logo");

if (logoSlot!=null)
```

```
logoSlot.add(logo);
```

What is the RootLayoutPanel?

If you're using a layout panel (which needs to know about its container resizing so it can resize itself), then you should add it to the browser page using `RootLayoutPanel` rather than `RootPanel` (if you don't, then it won't get resize events).

It's not possible to wrap parts of the page with a `RootLayoutPanel`, so a `Root-LayoutPanel(" someElement ")` call isn't available.

If you look at the `setUpGui` method in the `BasicProject` application, then you'll see you use both techniques of `RootPanel` to get the widgets and panels onto the page.

The second but less-preferred way of manipulating the page is directly via the DOM.

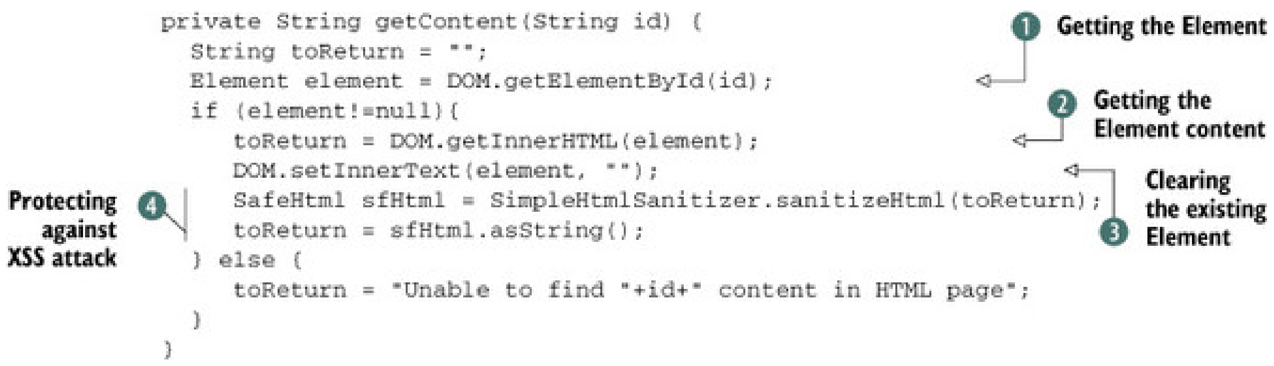
3.5.2. Manipulating the DOM directly

Most if not 99% of the time you'll have no need to directly manipulate the DOM; you can do what you need using widgets and panels. This means you can generally live in a fairly protected world and let GWT worry about most issues you might encounter.

Sometimes, though, you might need more control over the DOM, and GWT doesn't restrict you from doing this or force you to do it in only one way. In `BasicProject` you wish to take content from the HTML page and use it in the tabs of the `TabPanel` (you'll also have to delete that content from the HTML page so it's not shown twice). You can do all that only through direct DOM manipulation.

Listing 3.4 shows the `getContent` method you define in BasicProject.java to do these tasks, and it makes use of the both the `getElementId` and `getInnerHTML` methods from GWT's `DOM` class in the `com.google.gwt.user.client` package.

Listing 3.4. Accessing the DOM directly



```
private String getContent(String id) {
  String toReturn = "";
  Element element = DOM.getElementById(id);
  if (element!=null){
     toReturn = DOM.getInnerHTML(element);
     DOM.setInnerText(element, "");
     SafeHtml sfHtml = SimpleHtmlSanitizer.sanitizeHtml(toReturn);
     toReturn = sfHtml.asString();
  } else {
     toReturn = "Unable to find "+id+" content in HTML page";
  }
}
```

1 Getting the Element
2 Getting the Element content
3 Clearing the existing Element
4 Protecting against XSS attack

This `DOM` class contains many methods, and you use only three in listing 3.4. To get the text from the HTML page, you first find the `Element` that contains it (1)—remember way back in listing 3.1 you gave the `div` s in the HTML `id` s specifically for this situation; it's those `id` s that you now use.

Assuming that you've found the right `Element`, for example, the `Element` for `id contact`,

```
<div id="contact">
   You can contact us at
```

```
      <a href="mailto:a@a.com">1 Rudolf Street, The North Pole</a>

</div>
```

you get its inner HTML ❷, which happens to be

```
You can contact us at

   <a href="mailto:a@a.com">1 Rudolf Street, The North Pole</a>
```

Now that you have the inner HTML of the `Element`, you should clear the `Element` to prevent it from still being visible. That's done in ❸ by setting the DOM element's text to the blank string, giving you the following on the HTML page:

```
<div id="contact"></div>
```

You may have noticed that you return an object of type `SafeHtml` from the `get-Content` method rather than a `String` ❹. That's to try to protect your application against attacks such as cross-site scripting and is one part of GWT's approach to help secure applications—we'll come back to this at the end of the chapter.

Now you should be able to read the `setUpGui` method and the methods it calls of the application and understand what's going on with creating all the widgets and panels and getting them onto the browser page. Take some time to become comfortable with panels, widgets, and adding them to the page, and when you come back we'll look at how to react to the user doing something in the application.