NAME

Test::Builder::Module - Base class for test modules

SYNOPSIS

```
# Emulates Test::Simple
package Your::Module;

my $CLASS = __PACKAGE__;

use base 'Test::Builder::Module';
@EXPORT = qw(ok);

sub ok ($;$) {
    my $tb = $CLASS->builder;
    return $tb->ok(@_);
}

1;
```

DESCRIPTION

This is a superclass for Test::Builder-based modules. It provides a handful of common functionality and a method of getting at the underlying Test::Builder object.

Importing

Test::Builder::Module is a subclass of Exporter which means your module is also a subclass of Exporter. @EXPORT, @EXPORT_OK, etc... all act normally.

A few methods are provided to do the `use Your::Module tests =` 23> part for you.

import

Test::Builder::Module provides an import() method which acts in the same basic way as Test::More's, setting the plan and controling exporting of functions and variables. This allows your module to set the plan independent of Test::More.

All arguments passed to import() are passed onto `Your::Module->builder->plan()` with the exception of `import =`[qw(things to import)]>.

```
use Your::Module import => [qw(this that)], tests => 23;
```

says to import the functions this() and that() as well as set the plan to be 23 tests.

import() also sets the exported_to() attribute of your builder to be the caller of the import() function.

Additional behaviors can be added to your import() method by overriding import_extra().

import_extra

```
Your::Module->import_extra(\@import_args);
```

import_extra() is called by import(). It provides an opportunity for you to add behaviors to your module based on its import list.

Any extra arguments which shouldn't be passed on to plan() should be stripped off by this method.

See Test::More for an example of its use.

NOTE This mechanism is *VERY ALPHA AND LIKELY TO CHANGE* as it feels like a bit of an ugly

hack in its current form.

Builder

Test::Builder::Module provides some methods of getting at the underlying Test::Builder object.

builder

```
my $builder = Your::Class->builder;
```

This method returns the Test::Builder object associated with Your::Class. It is not a constructor so you can call it as often as you like.

This is the preferred way to get the Test::Builder object. You should *not* get it via `Test::Builder->new` as was previously recommended.

The object returned by builder() may change at runtime so you should call builder() inside each function rather than store it in a global.

```
sub ok {
    my $builder = Your::Class->builder;

    return $builder->ok(@_);
}
```