NAME

TAP::Parser::Scheduler - Schedule tests during parallel testing

VERSION

Version 3.36

SYNOPSIS

```
use TAP::Parser::Scheduler;
```

DESCRIPTION

METHODS

Class Methods

new

```
my $sched = TAP::Parser::Scheduler->new(tests => \@tests);
my $sched = TAP::Parser::Scheduler->new(
tests => [ ['t/test_name.t','Test Description'], ... ],
rules => \%rules,
);
```

Given 'tests' and optional 'rules' as input, returns a new `TAP::Parser::Scheduler` object. Each member of `@tests` should be either a a test file name, or a two element arrayref, where the first element is a test file name, and the second element is a test description. By default, we'll use the test name as the description.

The optional `rules` attribute provides direction on which tests should be run in parallel and which should be run sequentially. If no rule data structure is provided, a default data structure is used which makes every test eligible to be run in parallel:

```
{ par => '**' },
```

The rules data structure is documented more in the next section.

Rules data structure

The "`rules`" data structure is the the heart of the scheduler. It allows you to express simple rules like ''run all tests in sequence'' or ''run all tests in parallel except these five tests.". However, the rules structure also supports glob-style pattern matching and recursive definitions, so you can also express arbitarily complicated patterns.

The rule must only have one top level key: either 'par' for ''parallel'' or 'seq' for ''sequence''.

Values must be either strings with possible glob-style matching, or arrayrefs of strings or hashrefs which follow this pattern recursively.

Every element in an arrayref directly below a 'par' key is eligible to be run in parallel, while vavalues directly below a 'seq' key must be run in sequence.

Rules examples

Here are some examples:

```
# All tests be run in parallel (the default rule)
{ par => '**' },

# Run all tests in sequence, except those starting with "p"
{ par => 't/p*.t' },

# Run all tests in parallel, except those starting with "p"
{
seq => [
{ seq => 't/p*.t' },
{ par => '**' },
],
```

```
}

# Run some startup tests in sequence, then some parallel tests than some
# teardown tests in sequence.
{
seq => [
{ seq => 't/startup/*.t' },
{ par => ['t/a/*.t','t/b/*.t','t/c/*.t'], }
{ seq => 't/shutdown/*.t' },
],
},
```

Rules resolution

- By default, all tests are eligible to be run in parallel. Specifying any of your own rules removes this one.
- "First match wins". The first rule that matches a test will be the one that applies.
- Any test which does not match a rule will be run in sequence at the end of the run.
- The existence of a rule does not imply selecting a test. You must still specify the tests to run.
- Specifying a rule to allow tests to run in parallel does not make the run in parallel. You still need specify the number of parallel `jobs` in your Harness object.

Glob-style pattern matching for rules

We implement our own glob-style pattern matching. Here are the patterns it supports:

```
** is any number of characters, including /, within a pathname
* is zero or more characters within a filename/directory name
? is exactly one character within a filename/directory name
{foo,bar,baz} is any of foo, bar or baz.
\ is an escape character
```

Instance Methods

get_all

Get a list of all remaining tests.

get_job

Return the next available job as TAP::Parser::Scheduler::Job object or `undef` if none are available. Returns a TAP::Parser::Scheduler::Spinner if the scheduler still has pending jobs but none are available to run right now.

as_string

Return a human readable representation of the scheduling tree. For example:

```
my @tests = (qw{
t/startup/foo.t
t/shutdown/foo.t

t/a/foo.t t/b/foo.t t/c/foo.t t/d/foo.t
});
my $sched = TAP::Parser::Scheduler->new(
tests => \@tests,
rules => {
seq => [
{ seq => 't/startup/*.t' },
{ par => ['t/a/*.t','t/b/*.t','t/c/*.t'] },
{ seq => 't/shutdown/*.t' },
```

```
    ],
    },
    );
```

Produces:

```
    par:
    seq:
    par:
    seq:
    par:
    seq:
    't/startup/foo.t'
    par:
    seq:
    't/a/foo.t'
    seq:
    't/b/foo.t'
    seq:
    't/c/foo.t'
    par:
    seq:
    't/shutdown/foo.t'
    't/d/foo.t'
```