`angular.directive` API Proposal

This document describes a proposal for an api change that would affect angular.directive and angular.widget api.

The goals of the api changes are:
- make simple directives simple to write
- allow the current api flexibility for complicated directives
- support externalization of directive templates
- build in DI support
- stop exposing scope/compiler as `this` in instance/template functions
- unify directives, widgets, markup and their "attribute" variants under a single api and use the priority property to apply them in the right order

The new method signature

```
angular.directive(selectorString, [configObject,]
templateOrInstanceFunction);
```

where configObject and it's defaults are:

```
{
  descend: true,
  directives: true,
  reuseScope: false,
  templateFn: false,
  template: 'urlToTemplate',
  priority: angular.directive.PRIORITY.NORMAL,
  ...,
  inject: ['dep1', 'dep2']
}
```

after this change, there is no angular.widget, everything is a directive.

a very simple directive would then look like:

```
angular.directive('uf:helloworld', function(instanceElement, scope) {

  //instance function content

});
```

notice that you don't need to deal with template vs instance function, scope is exposed via function argument rather than `this`.

a more complicated example would look like:

```
angular.directive('@uf:dashboard', {
  template: 'templates/widgets/dashboard.html',
  templateFn: true,
  inject: ['$defer', '$xhr']
}, function(templateElement) {
  //template fn content
  return function($defer, $xhr, instanceElement, scope) {
    //instance fn content
  }
})
```

notice that we now allow for directive template to be externalized, when the template is loaded, it is compiled and linked and attached to the DOM by angular rather than by the widget author.

since the author is doing something non-trivial, we expect her to understand the difference between the template and instance function and use them properly.

the DI dependencies are defined via inject property, so that people don't have to use angular.extend or other non-obvious apis to inject stuff into widgets.

once again the scope is exposed via `scope` function argument.

in both the template and instance functions `this` is not explicitly bound to anything (defaults to the window object). if someone doesn't like this, they are free to use ES5 strict mode to catch any global namespace leaks.

Things to discuss

- should config object properties be prefixed with "$". I think that since there is no chance of collision, the extra $ is just pain in the ass

- how does this setup help/harm testability. current widgets and directives are very difficult to test, are we making the situation better or worse with this change?

- are the proposed defaults ok? e.g. reuseScope=false means that by default a new scope will be created for each directive. for element directives this makes sense, for attribute directives it doesn't.

- how should we deal with the priority option? should we just create constants e.g. PRIORITY.NORMAL or should we prioritize directives relatively (e.g. have people declare that a directive must run before/after some other directive). Is there opportunity to create constants that would be tied to the new scope lifecycle?

TODO

- we need a way to tell angular that a new scope created from a directive should be child of the root scope, instead of a child of the enclosing scope. I actually think that this should be the default since we don't want directives to depend on the state of the outer scope by default.

- we need a way to provide access to values for attribute directives and attribute/value pairs for element directives