Attention: This is a public document!

Github issue: #3605

Angular 2 offline template compilation proposal

Tobias Bosch (tbosch@google.com)
Date: 9/8/2015

Goal

- Reduce startup time of Angular apps by moving parts of the compilation process into a build step
- Do this in a way that does not significantly increase the size of the application

Results from a prototype

About 3x faster in bootstrap, see this sheet.

Proposal

Precompiled template

Original file:

```
import {NgIf} from 'angular2/angular2';
import {MyButton} from './my-button';

@Component()
@View(
  template: `
    <div attr1="value1" class="class1">some text</div>
    <div [title]="'test'"></div>
    <my-button (click)="shown = true">greet</my-button>
    <template ng-if="shown">hello {{user.name}}!</template>
  `,
  styles: ['.div {background-color: black}'],
  directives: [MyButton, NgIf])
class Main() {
  shown: boolean = false;
}
```

Precompiled template:

```
import {NgIf} from 'angular2/angular2';
```

```
import {MyButton} from './my-button';

var CA = '_ng-content-23';
var HA = '_ng-host-23';

export var styles = ['.div['+CA+']{background-color: black}'];

export var hostAttrs = [HA,''];

export function elements(be, bt, bc, ee, tt) {
  be('div',[CA,'','attr1','value1'],['class1']);
    tt('some text');
  ee();
  be('div',[CA,'',],[],true);
  ee();
  bc('my-button',[CA,'',],[],['click'],[MyButton]);
    tt('greet');
  ee();
  bt(NgIf);
    tt('',true);
  ee();
}
```

The functions be, ee, … stand for beginElement, text, beginTemplate, endTemplate, beginComponent, endElement. They are used as builder functions and might be directly forwarded to visitors if needed. Note that we allow an optimizer to rename these functions into single letter function names.

Size comparison:

- @View annotation with module imports for the used directives: 445 Bytes with gzip
- precompiled template: 418 bytes with gzip
- ratio: 1.07x bigger after gzip

Size notes:

- The precompiled template does not contain expressions or bindings, as they are contained in the generated change detectors.
- The precompiled template does contain the additional attributes for style encapsulation.

Changes to the APIs

New APIs:

```
interface Renderer {
  createProtoView():ProtoViewBuilder;
```

```
}

interface ProtoViewBuilder() {
  beginFragment();
  beginElement(name:string, attrs:string[], classes:string[],
bound: boolean);
  endElement();
  text(value:string, bound: boolean);
  ...
  build():RenderProtoViewRef;
}
```

Which parts of the compiler should not be changed (at least for now)?

- keep AppProtoView / DomProtoView / ProtoViewFactory as is, only change the way we create them
- includes keeping ProtoChangeDetector, ProtoElementInjector

Properties

Code changes to Angular are local to the compiler.

- other team members can work in parallel without creating merge conflicts.
- the complexity of the change is mostly understood

Visitor / Transformer design

- good way to store hierarchy information with little size
    - transformers / visitors can decide on their own how they want to store parent/child relationship (e.g. parent -> children for DOM or children -> parent for injectors)
    - e.g. distance to parent (needed for injectors) can be calculated from this easily
- good way to store meta information on elements and text nodes
    - no need for separate ElementBinders array / ng-binding class
    - no need to treat text nodes specially, can just associate metadata to them
    - metadata is stored directly on the element / text node. This makes projection and ProtoView merging a lot easier, as we don't have to recalculate the ElementBinders order after moving DOM around…
    - can have metadata on elements that won't add entries to arrays for app side (e.g. whether there is a ng-content element)
- hopefully less intermediate arrays / less memory pressure as we can chain visitors / transformers

The precompiled template contains all of the@View information

- but only the directives/directive bindings that are really used

- helps in tree shaking!

Templates can be loaded on app side
- data flow is now also app -> render, also for creating proto views.
- are able to merge proto views synchronously -> can do it lazily!
    - i.e. merge embedded ng-if only when their first view is created
    - i.e. create the AppProtoView eagerly, only the ProtoViewMergeMapping and the DomProtoView lazily.
- no need for ProtoViewDto any more!

ProtoViews can be merged (or even created) lazily
- as we have all information in a central place on the app side from which we can create everything that is needed synchronously

Easy to include into an application as it is just a source code module

DomViews can be directly created from this
- we can still create DOM nodes that we just clone afterwards for "hot" templates that are cloned many times -> can reduce the number of nodes in the system
- also any other renderer can use this to create native elements, e.g. for react-native, nativescript

Nested components don't get merged until we read out the template
- provides locality: when a template is changed, only this template needs to be precompiled again, but no other templates
- important for small size of templates, as otherwise the templates of components would be serialized multiple times

Probably also speeds up on-the-fly compilation
- e.g. for ProtoView merging: no need to create ElementBinders after changing the DOM

Alternate proposals

Serialize current `ProtoViewDto` and `DomProtoView`

Template would grow big easily
- we have a separate ProtoView per <template>
- `DomProtoView` is already merged, i.e. it contain nested components multiple times
- does not speedup proto view merging -> could get only 1.4x faster at most (see numbers above)

More radical changes (e.g. pregenerate DI, …)

This could lead to even better compile / view instantiation times. However, we need more data before we do these kind of changes, as it is not clear right now what is the main contributor to the compile / view instantiation time in Angular after the changes outlined above.

The idea is to implement the proposal of this document first, and then take the learnings to work on more improvements.

Related projects

- Incremental dom

Compile Process

On the fly:

1. load and compile single template
2. load and compile templates of nested components
3. call visitors with merged elements

Transformers:

1. load and compile single template
2. serialize into file with references to other components
3. load templates via an import
4. call visitors with merged elements

I.e. 2 stages:

1. compile single template
    a. needs to normalize some things as well (e.g. <div *ng-if> into <template ng-if>)
    b. output is used to generate change detectors and precompiled templates like above
2. continue compilation with merged templates

Content projection:

- while we execute the template on the render side
- can calculate index mapping while we do this as well
- probably only needed on render side, although we need to walk nested components on the app side as well for things like `hostElementIndices` without projecting though.
- buffer light dom nodes until `endComponent`, then do the projection.
- still need a ngContentCount on beginTemplate for app side

TemplateVisitor / TemplateTransformers

- ComponentMerger
- AppProtoViewFactory
- DomProtoViewFactory / DomViewFactory

Milestones

1. POC to find out whether estimates hold
    - get more data where we spend our time in compilation?
        - to get an estimate of how much this approach will change, as we keep a lot of things the same...
        - just add a Date.now() and global counter in the largeapp app of Ferhat
    - hand write templates by hand for the largeapp benchmark and proof the goals: measure creation of the largeapp on first click and measure download size change in %.
    - keep the expressions in the template for now to not be blocked

2. Implementation of this proposal

3. Revisit numbers and think about further changes