Ivy compatibility guide

The Angular team has worked hard to ensure Ivy is as backwards-compatible with the previous rendering engine ("View Engine") as possible. However, in rare cases, minor changes were necessary to ensure that the Angular's behavior was predictable and consistent, correcting issues in the View Engine implementation. In order to smooth the transition, we have provided automated migrations wherever possible so your application and library code is migrated automatically by the CLI. That said, some applications will likely need to apply some manual updates.

How to debug errors with Ivy

In version 10, a few deprecated APIs have been removed and there are a few breaking changes unrelated to Ivy. If you're seeing errors after updating to version 9, you'll first want to rule those changes out.

To do so, temporarily turn off Ivy in your `tsconfig.base.json` and re-start your app.

If you're still seeing the errors, they are not specific to Ivy. In this case, you may want to consult the general version 10 guide. If you've opted into any of the new, stricter type-checking settings, you may also want to check out the template type-checking guide.

If the errors are gone, switch back to Ivy by removing the changes to the `tsconfig.base.json` and review the list of expected changes below.

Payload size debugging

If you notice that the size of your application's main bundle has increased with Ivy, you may want to check the following:

1. Verify that the components and `NgModules` that you want to be lazy loaded are only imported in lazy modules. Anything that you import outside lazy modules can end up in the main bundle. See more details in the original issue here.
2. Check that imported libraries have been marked side-effect-free. If your app imports from shared libraries that are meant to be free from side effects, add "sideEffects": false to their `package.json`. This will ensure that the libraries will be properly tree-shaken if they are imported but not directly referenced. See more details in the original issue here.
3. Projects not using Angular CLI will see a significant size regression unless they update their minifier settings and set compile-time constants `ngDevMode`, `ngI18nClosureMode` and `ngJitMode` to `false` (for Terser, please set these to `false` via `global_defs` config option). Please note that these constants are not meant to be used by 3rd party library or application code as they are not part of our public api surface and might change in the future.

Changes you may see

- By default, `@ContentChildren` queries will only search direct child nodes in the DOM hierarchy (previously, they would search any nesting level in the DOM as long as another directive wasn't matched above it). See further details.
- All classes that use Angular DI must have an Angular decorator like `@Directive()` or `@Injectable` (previously, undecorated classes were allowed in AOT mode only or if injection flags were used). See further details.
- Unbound inputs for directives (e.g. name in `<my-comp name="">`) are now set upon creation of the view, before change detection runs (previously, all inputs were set during change detection).
- Static attributes set directly in the HTML of a template will override any conflicting host attributes set by directives or components (previously, static host attributes set by directives / components would override static template attributes if conflicting).

Less common changes

- Properties like `host` inside `@Component` and `@Directive` decorators can be inherited (previously, only properties with explicit field decorators like `@HostBinding` would be inherited).
- HammerJS support is opt-in through importing the `HammerModule` (previously, it was always included in production bundles regardless of whether the app used HammerJS).
- `@ContentChild` and `@ContentChildren` queries will no longer be able to match their directive's own host node (previously, these queries would match the host node in addition to its content children).
- If a token is injected with the `@Host` or `@Self` flag, the module injector is not searched for that token (previously, tokens marked with these flags would still search at the module level).
- When accessing multiple local refs with the same name in template bindings, the first is matched (previously, the last instance was matched).
- Directives that are used in an exported module (but not exported themselves) are exported publicly (previously, the compiler would automatically write a private, aliased export that it could use its global knowledge to resolve downstream).
- Foreign functions or foreign constants in decorator metadata aren't statically resolvable (previously, you could import a constant or function from another compilation unit, like a library, and use that constant/function in your `@NgModule` definition).
- Forward references to directive inputs accessed through local refs are no longer supported by default. details
- If there is both an unbound class attribute and a `[class]` binding, the classes in the unbound attribute will also be added (previously, the class binding would overwrite classes in the unbound attribute).
- It is now an error to assign values to template-only variables like `item` in `ngFor="let item of items"` (previously, the compiler would ignore these assignments).
- It's no longer possible to overwrite lifecycle hooks with mocks on directive instances for testing (instead, modify the lifecycle hook on the directive type itself).
- Special injection tokens (such as `TemplateRef` or `ViewContainerRef`) return a new instance whenever they are requested (previously, instances of special tokens were shared if requested on the same node). This primarily affects tests that do identity comparison of these objects.
- ICU parsing happens at runtime, so only text, HTML tags and text bindings are allowed inside ICU cases (previously, directives were also permitted inside ICUs).
- Adding text bindings into i18n translations that are not present in the source template itself will throw a runtime error (previously, including extra bindings in translations was permitted).
- Extra HTML tags in i18n translations that are not present in the source template itself will be rendered as plain text (previously, these tags would render as HTML).
- Providers formatted as `{provide: X}` without a `useValue`, `useFactory`, `useExisting`, or `useClass` property are treated like `{provide: X, useClass: X}` (previously, it defaulted to `{provide: X, useValue: undefined}`).
- `DebugElement.attributes` returns `undefined` for attributes that were added and then subsequently removed (previously, attributes added and later removed would have a value of `null`).
- `DebugElement.classes` returns `undefined` for classes that were added and then subsequently removed (previously, classes added and later removed would have a value of `false`).
- If selecting the native `<option>` element in a `<select>` where the `<option>`s are created via `*ngFor`, use the `[selected]` property of an `<option>` instead of binding to the `[value]` property of the `<select>` element (previously, you could bind to either.) details
- Embedded views (such as ones created by `*ngFor`) are now inserted in front of anchor DOM comment node (e.g. `<!--ng-for-of-->`) rather than behind it as was the case previously. In most cases this does not have any impact on rendered DOM. In some cases (such as animations delaying the removal of an embedded view) any new embedded views will be inserted after the embedded view being animated away. This difference only last while the animation is active, and might alter the visual appearance of the animation. Once the animation is finished the resulting rendered DOM is identical to that rendered with View Engine.