'no-inferrable-types' and 'typedef' rules conflict #711

New issue

Open **zdychacek** opened this issue on Oct 3, 2015 · 31 comments

zdychacek commented on Oct 3, 2015

Hi again,

I am not sure if this is issue or design decision, but I can see conflict between rules `no-inferrebale-types` and `typedef`.

Ex.

```
function fn(): void {
    for (let i = 0; i < 100; i++) {
        console.log(i);
    }
}
```

Snippet from https://github.com/palantir/tslint/blob/master/docs/sample.tslint.json:

```
 "typedef": [true, ...],
 "no-inferrable-types": true,
```

When I have these two rules turned on, I get:

```
error (typedef) test.ts[2, 12]: expected variable-declaration: 'i' to have a typedef
```

Assignees

No one assigned

Labels

P1

Status: Accepting PRs

Type: Enhancement

Projects

None yet

Milestone

TSLint 5.x

20 participants

```
error (no-inferrable-types) test.ts[2, 14]: LHS type (number) inferred by RHS expression,
remove type annotation
```

Is there any way to have these two rules coexisted with each other?

For example, I want to have inferrable variables directly declared with primitive type (as rule doc says: `number`, `boolean` or `string`), but on the other hand, I want to force typedefs on non-primitive types.

Thanks,

O.

46

JKillian commented on Oct 3, 2015 Contributor

Good catch, thanks for the heads-up. I added the `no-inferrable-types` rule without thinking about how it might conflict with the `typedef` rule, whoops! For now I'd recommend turning one of the two off, or only using some options of the `typedef` rule.

Longer term, I think we'll either want to integrate `no-inferrable-types` into the `typedef` rule or we'll want to at least have TSLint detect conflicting configurations like having both rules on.

6

adidahiya added the Type: Question label on Oct 4, 2015

timbru31 commented on Oct 5, 2015

I am facing the same issue.
I've turned off no-inferrable-types for now.

Yes same here, I would like to have ability to have both the rules co-exist.
I would not want typedef rule to kick in if the variable's type can be inferred.
i.e. "no-inferrable-types" should have priority over "typedef"

3

This was referenced on Oct 17, 2015

Define no-null-keyword rule #722 Merged

Allow Conflicting Rules to Exist but not Be Concurrently Enabled #739 Closed

helios1138 commented on Dec 4, 2015

+1

adidahiya added Type: Enhancement Status: In Discussion and removed Type: Question labels on Dec 5, 2015

adidahiya self-assigned this on Dec 5, 2015

Connormiha commented on Dec 31, 2015

```
let id: number = 0;
for (let job: string of NAMES_PROFFESSIONS) {
     /** some code */
     id++;
}
```

adidahiya removed their assignment on Jan 6, 2016

cronon commented on Jan 21, 2016

+1

octogonz commented on Jan 26, 2016

Does your definition of "inferrable" types include constructor assignments?

```
// BAD (this hurts my eyes to read)
let labels: Map<string, string> = new Map<string, string>();
// GOOD (type is obvious)
let labels = new Map<string, string>();
```

but also...

```
// BAD (in a diff, it's not obvious what this type is)
let labels = this.buildLabels();
// GOOD
let labels: Map<string, string> = this.buildLabels();
```

9

Avol-V commented on Jan 28, 2016

Yes, it's dangerous. If I want to simplify my code and prevent to use type declaration for directly initialized variables, I can't do this strictly and this brings to such thing:

```
x = 1;
y = 1;
z = 1;
x = 's'; // Type 'string' is not assignable to type 'number'
y = 's'; // It's OK
z = 's'; // Type 'string' is not assignable to type 'number'
```

It's may be a very useful option to allow skip type declaration only for initialized variables.

Avol-V commented on Feb 24, 2016

... and really not only for primitive types, as **@pgonzal** says!
Look at this, it's terrible:

```
const onChange: () => void = () => this.update();
```

englercj commented on Mar 10, 2016

Ideally (imo) I would like a way to say "always require a typedef, unless the type is inferrable". Which I don't think is possible right now.

64

JKillian referenced this issue on Mar 17, 2016

["typedef" > "variable-declaration"] conflicting in usage with ["no-inferrable-types": false] #1038 Closed

dsebastien referenced this issue on Apr 4, 2016

Add tslint rules #80 Open

2 of 4 tasks complete

abierbaum referenced this issue on May 1, 2016

no-inferrable-types: Add ignore-params flag #1190 Merged

abierbaum commented on May 1, 2016 Contributor

I ran into this and made an `ignore-params` flag that helps out in my case. I want to force typedefs for all method/function parameters even when they can be easily inferred.

See PR: #1190 if you want to try it out

JKillian referenced this issue on May 20, 2016

Inferred default parameter is erroneous using "typedef" "parameter" #1263 Closed

corydeppen commented on Jun 13, 2016 Contributor

It's been a while since the original issue was submitted. Is the recommend option at this point to disable `no-inferrable-types` and include the type on everything? Anything else to try and enable and combine both `no-inferrable-types` and `typedef` seems like a hack and results in a bunch of pointless warnings. Hoping for a better solution in the near future.

1

azdavis added a commit to azdavis/azdavis.xyz that referenced this issue on Aug 18, 2016

`allow inferrable-types` ··· `c6bfd64`

adidahiya modified the milestones: **TSLint v3.x**, **TSLint v4.x** on Sep 1, 2016

10 hidden items
Load more...

michaeljota commented on Jul 9, 2017

I think the best way to handle this is to deprecate the `no-inferrable-types` and just pass an option object to `typedef` to ignore the lack of type definitions if the type is inferrable according to certain patters, as `initialized`, `initialized primitives`, `call signatures` and other patters that would fulfill our needs as developers.

For me this makes more sense, cause there should be always a typedef, unless there is something telling you what it's the type of the function. And it would be configurable as well, because, maybe we want the initialized `properties` in a class to have inferrable type, but not for the `call signatures` for example.

1

octogonz commented on Jul 11, 2017

It would be great if someone could pitch in to get this fixed. Until then, we are forced to choose between "not enough type declarations to be readable" versus "cluttered with too many type declarations".

This issue has been open since Oct 3, 2015 -- since then, my team has authored around 2000 TypeScript source files that are all cluttered with too many type declarations.

michaeljota commented on Jul 12, 2017

The `typedef` accepts a configuration. So maybe just another configuration just to ignore the type definition if the type is inferrable, meaning

> Just type wherever is not being initialized.

15 4

marcdumais-work referenced this issue on Aug 1, 2017

[quality] consider adding more rules to dev.tslint.json #356 Closed

whyboris commented on Sep 26, 2017 • edited

It looks like because there is no clear consensus on how to deal with the issue, no progress is being made. Example comment: theia-ide/theia#356 (comment)

I suspect we all agree that any solution is better than leaving things as is. If you think any change to the status-quo with respect to this issue is good, please this comment. If you're knowledgeable enough to create a PR to fix this issue, please help all of us out .

49

ajafff referenced this issue on Dec 20, 2017

Conflict between "typedef" and "no-inferrable-types" #3595 Closed

1 of 2 tasks complete

JKillian referenced this issue on May 16, 2018

'expected variable-declaration to have a typedef' lint error on for loops #3912

Closed

JKillian commented on May 16, 2018 • edited

Contributor

Would accept a PR to:

- add an option to `typedef` that lets it ignore cases where `no-inferrable-types` says not to provide a type

> The typedef accepts a configuration. So maybe just another configuration just to ignore the type definition if the type is inferrable, meaning 'Just type wherever is not being initialized.'

The `typedef` rule is for people who like having explicit type definitions in their codebase. If you desire the above behavior to "just type wherever is not being initialized", you're better off disabling `typedef` and making sure you have `noImplictAny` enabled as a TypeScript compiler option.

There's also the tricky case where some things that are initialized need a typedef anyways, as in the following snippet:

```
interface Literal {
    field: "value"
}

const literal0 = {
    field: "value",
};
const literal1: Literal = {
    field: "value",
};
```

```
func(literal);
```

JKillian added Status: Accepting PRs Type: Enhancement and removed Status: In Discussion Type: Enhancement labels on May 16, 2018

JKillian commented on May 16, 2018 Contributor

Also, while we get this fixed, wanted to mention that there are likely some great 3rd-party rules out there, like `no-unnecessary-type-annotation` (https://github.com/ajafff/tslint-consistent-codestyle/blob/master/docs/no-unnecessary-type-annotation.md) for example. If anyone knows of any other 3rd-party rules that give the desired behavior, please post them here and we can officially recommend them or adopt them into core if it makes sense.

michaeljota commented on May 16, 2018

@JKillian thanks for the recommendation, I think that's actually what I wanted. There is a very good post about avoiding `any` type: Don't use "naked any", create an "any interface" instead.

About:

```
interface Literal {
    field: "value"
}

const literal0 = {
    field: "value",
};
const literal1: Literal = {
    field: "value",
};

const func = (obj: Literal) => { };
```

I don't see how this could be an undesired behavior nor a tricky case. You want to make sure that `obj` have a property `field` with value `value`, and even when you are initializing `literal0` with a property that seems like the constrains, you could modify that to another string.

I know is not a good use case, but most of the cases when you are using a literal, you probably want that literal, not a primitive.

sandrocsimas commented on Jun 9, 2018

I have the following configuration:

```
"no-inferrable-types": true,
"typedef": [true, "call-signature", "parameter"],
```

And this code:

```
private static readonly DEVICE_UID: string = 'device_uid';
private static readonly DEVICE_PLATFORM: string = 'browser';
private static readonly AGENT_DEFAULT_ICON = 'http://localhost:3000/icon.png';
```

Why I'm not getting error in the two first declarations?

estaub commented on Jun 10, 2018

@sandrocsimas Interesting, but off-topic I think; AFAICT that problem's unrelated to this issue. I'd suggest you start another issue (fwiw!).

sandrocsimas commented on Jun 10, 2018

michaeljota commented on Jun 10, 2018

@**sandrocsimas** that's because it's a readonly property, and such Typescript infer its type as a literal. Typing it as a string you are telling that it should have a string, it does not necessarily will have that literal value and the value should not change statically.

2

kopelli added a commit to kopelli-forks/bitburner that referenced this issue on Jun 25, 2018

`[chore] Disable "no-inferrable-types" lint rule` ··· `1857103`

kopelli referenced this issue on Jun 25, 2018

[chore] Disable "no-inferrable-types" lint rule #332 Merged

FiretronP75 commented on Aug 28, 2018

It would be nice to have a 'require-typedef-except-inferrable' rule.

10

michaeljota commented on Aug 28, 2018

@**FiretronP75** as @**JKillian** said, that's just `noImplicitAny` option of the TSC.

1

@michaeljota thanks, I didn't realize the `noImplicitAny` option of the compiler gives exceptions for inferrable. It still would be nice to have in tslint though, for the option of making it a warning instead of breaking compile, and for having the tslint comment flags.

1

michaeljota commented on Aug 29, 2018 • edited

I see why this would be something wanted, but having `no-unused-variables` as an example, I don't think use cases covered by the TSC are going to be supported by the TSLint team. I know that is not the same an *linter error* than a *compiler error* but at the end, they both are about written better code. Now days with solutions as Webpack or Parcel that allows you to compile and run the code even with TSC errors, I don't see this as a real issue.

aaronchenwei referenced this issue on Nov 16, 2018

review rule - no-inferrable-types #92 Closed

Bernd-L commented on Feb 17

Has this been fixed in the latest version?

michaeljota commented on Feb 19

I still don't think this is on the roadmap. You should consider using `noImplicitAny` from the TSC

1

Unnecessary member-variable-declaration expected for 'state' and 'defaultProps' #174 Closed

glen-84 referenced this issue 4 days ago

[no-inferrable-types][typedef] Rules clash in 'all' config #902 Open