@alpha

建议的含义
指定 API 项的发布阶段为 `alpha`。它旨在供 第三方开发人员，但尚未发布。该工具可以从 公开发布。

例

```
/**
 * 表示目录中的一本书。
 * @public
 * */
export class Book {
    /**
     * 书的标题。
     * @alpha
     * */
    public get title(): string;

    /**
     * 书的作者。
     * */
    public get author(): string;
}
```

在此示例中，从包含类 while 被标记为 “alpha”。 `Book.author` `@public` `Book.title`

@beta

建议的含义
指定 API 项目的发布阶段为 `beta`。它已以实验方式向第三方开发人员发布 用于收集反馈。API 不应该在生产中使用，因为它的合约可能会
如有更改，恕不另行通知。该工具可以从公开版本中修剪声明，但可以将其包含在 开发人员预览版。

例

```
/**
 * 表示目录中的一本书。
 * @public
 * */
export class Book {
    /**
     * 书的标题。
     * @beta
     * */
    public get title(): string;

    /**
     * 书的作者。
     * */
    public get author(): string;
}
```

在此示例中，从包含类 while 被标记为 “beta”。Book.author@publicBook.title

@category

当 列在页面的索引中。可以多次指定它以在多个 标题。`@category`

例

```
/**
 * @category General Use
 */
export function runProcess(): void;

/**
 * @category Advanced Use
 */
export function unref(): void;

/**
 * @category Advanced Use
 */
export function ref(): void;
```

导航自定义
可以使用 选项将类别添加到导航树中。这可以通过指定 和 modifier 标记中的
对父反射的注释。`navigation.includeCategories` `@showCategories` `@hideCategories`

@categoryDescription

该标签可用于提供有关反射类别的其他上下文 它是使用 @category 标签创建的。
@categoryDescription

该标签应放置在包含反射的注释中 标有 的子反射。@categoryDescription@category

的第一行将作为类别名称，接下来的几行将 用于描述。@categoryDescription

例

```
/**
 * @categoryDescription Advanced Use
 * These functions are available for...
 * @module
 */

/**
 * @category General Use
 */
export function runProcess(): void;

/**
 * @category Advanced Use
 */
export function unref(): void;

/**
 * @category Advanced Use
 */
```

```
export function ref(): void;
```

@decorator

用法
ECMAScript 装饰器有时是一个重要的部分 API 协定中。但是，目前 TypeScript 编译器在 .d.ts 输出文件中不表示装饰器 由 API
使用者使用。该标签提供了一种解决方法，允许引用装饰器表达式 在文档评论中。@decorator

例

```
class Book {
    /**
     * 书的标题。
     * @decorator `@jsonSerialized`
     * @decorator `@jsonFormat(JsonFormats.Url)`
     * */
    @jsonSerialized
    @jsonFormat(JsonFormats.Url)
    public website: string;
}
```

@deprecated

用法
此块标签表示 API 项目不再受支持，并且可能会在未来版本中删除。 该标签后跟一个描述推荐替代方案的句子。它以递归方式应用
添加到容器的成员。例如，如果某个类已弃用，则其所有成员也已弃用。@deprecated

例

```
/**
 * 可以呈现的控件的基类。
 *
 * @deprecated 使用新的基类 {@link Control} 代替.
 * */
export class VisualControl {
}
```

@defaultvalue

用法
如果未明确分配值，则此块标记用于记录字段或属性的默认值。

此标记只能与属于 TypeScript 或 .`class` `interface`

例

```
enum WarningStyle {
    DialogBox,
    StatusMessage,
    LogOnly
}
```

```
interface IWarningOptions {
    /**
     * 确定如何显示警告。
     *
     * @remarks
     * 详情请参见 {@link WarningStyle| the WarningStyle enum}.
     *
     * @defaultValue `WarningStyle.DialogBox`
     * */
    warningStyle?: WarningStyle;

    /**
     * 警告是否会中断用户当前的活动。
     * @defaultValue
     * 默认未 `true ` 除非 `WarningStyle.StatusMessage` 被指定.
     * */
    cancellable?: boolean;

    /**
     * 警告信息
     * */
    message: string;
}
```

@enum

如果存在于具有字符串或数字字面值的对象上，TypeDoc 会将变量转换为 枚举而不是变量。

例

```
/**
 * This will be displayed as an enumeration.
 * @enum
 */
export const MyEnum = {
        /**
         * Doc comments may be included here.
         */
        A: "a",
        B: "b"
    } as const;

/**
 * This works too, but is more verbose
 * @enum
 */
export const MyEnum2: { A: "a" } = { A: "a" };

/**
 * So does this, for declaration files
 */
export declare const MyEnum3: { A: "a" };
```

@event

该标签用于将倒影标记为属于 “Events” 组。 它等效于在注释中指定。`@event` `@group` `Events`

例

```
export class App extends EventEmitter {
    /**
     * @event
     */
    static ON_REQUEST = "request";
}
```

@eventProperty

该标签用于将倒影标记为属于 “Events” 组。 它等效于在注释中指定。`@eventProperty` `@group` `Events`

例

```
export class App extends EventEmitter {
    /**
     * @eventProperty
     */
    static ON_REQUEST = "request";
}
```

@eventProperty

用法
当应用于类或接口属性时，这表示该属性 返回事件处理程序可以附加到的事件对象。事件处理 API 是实现定义的，但通常属性返回类型为类
替换为 和 等成员。文档工具可以 在 “Events” 标题下显示此类属性，而不是通常的 “Properties” 标题。addHandler()removeHandler()

例

```
class MyClass {
    /**
     * 每当应用程序导航到新页面时，都会触发此事件。
     * @eventProperty
     * */
    public readonly navigatedEvent: FrameworkEvent<NavigatedEventArgs>;
}
```

@example

用法
指示应作为示例显示的文档部分，以说明如何使用 API。 它可能包括代码示例。

应解释与标签出现在同一行的任何后续文本 作为示例的标题。否则，文档工具可以按数字方式为示例编制索引。`@example`

示例 A
对于此代码示例，生成的标题可能是“Example”和“Example 2”：

```
/**
 * 两数相加
 * @example
 * 这是一个简单的示例:
 *
 * // Prints "2":
 * console.log(add(1,1));
 *
 * @example
 * 这是一个负数例子:
 *
 * // Prints "0":
 * console.log(add(1,-1));
 *
 * */
export function add(x: number, y: number): number {
}
```

示例 B
对于此代码示例，生成的标题可能是“示例：解析基本 JSON 文件”：

```
/**
 * 解析Json文件.
 *
 * @param path - 文件绝对路径.
 * @returns 包含Json数据的对象.
 *
 * @example 解析基本 JSON 文件
 *
 * # `file.json`的内容
 * ```json
 * {
 *   "exampleItem": "text"
 * }
 * \```
 *
 * # 用法
 * ```ts
 * const result = parseFile("file.json");
 * \```
 *
 * # 结果
 * ```ts
 * {
 *   exampleItem: 'text',
 * }
 * \```
 * */
```

@experimental

建议的含义
语义与 相同，但由不支持发布阶段的工具使用。@beta@alpha

例

```
/**
 * 表示目录中的一本书。
 * @public
 * */
export class Book {
    /**
     * 书的标题。
     * @experimental
     * */
    public get title(): string;

    /**
     * 书的作者。
     * */
    public get author(): string;
}
```

在此示例中，从包含类 而标记为 `实验性`。 `Book.author` `@public` `Book.title`

@group

当 列在页面的索引中。可以多次指定它以在多个 标题。 `@group`

与 `@category` 标签不同，反射将自动放置在 根据其 kind 的 Header（如果未指定标记）。此标签可用于 模拟自定义杆件类型。 `@group`

例

```
export class App extends EventEmitter {
    /**
     * @group Events
     */
    static readonly BEGIN = "begin";

    /**
     * The `@event` tag is equivalent to `@group Events`
     * @event
     */
    static readonly PARSE_OPTIONS = "parseOptions";

    /**
     * The `@eventProperty` tag is equivalent to `@group Events`
     * @eventProperty
     */
    static readonly END = "end";
}
```

导航自定义
可以使用 option 将组添加到导航树中。这可以通过指定 和 modifier 标记中的
对父反射的注释。`navigation.includeGroups` `@showGroups` `@hideGroups`

@groupDescription

该标签可用于提供有关一组反射的其他上下文。 TypeDoc 根据反射的 TypeScript 类型自动对反射进行分组，但自定义组可以
使用 `@group` 标记创建。`@groupDescription`

该标签应放置在包含反射的注释中 子反射组。`@groupDescription`

的第一行将作为组名称，接下来的几行将 用于描述。`@groupDescription`

例

```
/**
 * @groupDescription Events
 * Events are for...
 */
export class App extends EventEmitter {
    /**
     * @group Events
     */
    static readonly BEGIN = "begin";

    /**
     * The `@event` tag is equivalent to `@group Events`
     * @event
     */
    static readonly PARSE_OPTIONS = "parseOptions";

    /**
     * The `@eventProperty` tag is equivalent to `@group Events`
     * @eventProperty
     */
    static readonly END = "end";
}
```

@hidden

标有该标签的反射将从文档中删除。 它相当于 JSDoc 标签，也被 TypeDoc 识别。`@hidden` `@ignore`

例

```
export class Visibility {
    /** @hidden */
    newBehavior(): void;
}
```

@hideconstructor

此标签只能用于解决 TypeScript#58653。 请改用 `@hidden` 或 `@ignore` 标签。

标记为 的类将隐藏其构造函数，它也可以放置在构造函数上以 从文档中删除它们`@hideconstructor`

例

```
/** @hideconstructor */
export class Visibility {
    /** Will not be present in the generated documentation */
    constructor() {
    }
}
```

@ignore

标有该标签的反射将从文档中删除。 它等效于 JSDoc 标记。`@hidden@ignore`

例

```
export class Visibility {
    /** @ignore */
    newBehavior(): void;
}
```

@interface

如果存在在类型别名上，将导致它被转换为接口。这将导致所有 “dynamic” 属性 扩展到不动产。

例

```
/**
 * This will be displayed as an interface
 * @property a comment for a
 * @prop b comment for b
 * @interface
 */
export type Resolved = Record<"a" | "b" | "c", string>;


// will be documented as if you wrote

/** This will be displayed as an interface */
export interface Resolved {
    /** comment for a */
    a: string;
    /** comment for b */
    b: string;
    c: string;
}
```

@inheritDoc

用法
此内联标签用于通过从另一个 API 项目复制 API 项目的文档来自动生成 API 项目的文档 API 项。inline tag
参数包含对另一个项目的引用，该项目可能是一个不相关的类 甚至是从单独的 NPM 包导入。

> **Note**
> 声明引用的符号尚未最终确定。请参阅 GitHub 问题 #9

复制的内容
该标签不会复制整个评论正文。仅复制以下组件：`@inheritDoc`

- 摘要部分
- @remarks块
- @params块
- @typeParam块
- @returns块
  其他标记（如 or 未复制），并且需要在
  标签。指定标记后，注释中不能指定摘要部分或部分。`@defaultValue` `@example` `@inheritDoc` `@inheritDoc` `@remarks`

例

```
import { Serializer } from "example-library";


/**
 * 描述页面的接口
 * @public
 */
export interface IWidget {
    /**
     * 通过显示接口绘制页面
     * @param x - 页面X坐标
     * @param y - 页面Y坐标
     */
    public

    draw(x: number, y: number): void;
}


/** @public */
export class Button implements IWidget {
    /** {@inheritDoc IWidget.draw} */
    public draw(x: number, y: number): void {
    }

    /**
     * {@inheritDoc example-library#Serializer.writeFile}
     * @deprecated 使用 {@link example-library#Serializer.writeFile} 代替.
     */
```

```
    public save(): void {
    }
}
```

@internal

建议的含义
指定 API 项不计划由第三方开发人员使用。该工具可能会修剪 来自公开发布的声明。在某些 implementations 中，某些指定的软件包可能被允许
使用内部 API 项，例如，因为软件包是同一产品的组件。

例

```
/**
 * 表示目录中的一本书。
 * @public
 */
export class Book {
    /**
     * 书的标题.
     * @internal
     */
    public get _title(): string;

    /**
     * 书的作者.
     */
    public get author(): string;
}
```

在此示例中，从包含类 而标记为 `internal`。`Book.author` `@public` `Book._title`

@label

用法
inline 标签用于标记声明，以便可以使用 TSDoc 声明引用表示法。`{@label}`

注意：符号尚未最终确定。请参阅 GitHub 问题 #9 `{@label}`

例

```
export interface Interface {
    /**
     * 缩写:  {@link InterfaceL1.(:STRING_INDEXER)}
     * 全名:  {@link (InterfaceL1:interface).(:STRING_INDEXER)}
     *
     * {@label STRING_INDEXER}
     */
    [key: string]: number;

    /**
     * 缩写:  {@link InterfaceL1.(:NUMBER_INDEXER)}
     * 全名:  {@link (InterfaceL1:interface).(:NUMBER_INDEXER)}
     *
     * {@label NUMBER_INDEXER}
```

```
     */
    [key: number]: number;

    /**
     * 缩写:  {@link InterfaceL1.(:FUNCTOR)}
     * 全名:  {@link (InterfaceL1:interface).(:FUNCTOR)}
     *
     * {@label FUNCTOR}
     */
    (source: string, subString: string): boolean;

    /**
     * 缩写:  {@link InterfaceL1.(:CONSTRUCTOR)}
     * 全名:  {@link (InterfaceL1:interface).(:CONSTRUCTOR)}
     *
     * {@label CONSTRUCTOR}
     */
    new(hour: number, minute: number);
}
```

@link

用法
内联标签用于创建指向 文档系统或通用 Internet URL。特别是，它支持 用于引用 API 项的表达式。
`{@link}`

> **Note**
> 声明引用的符号尚未最终确定。请参阅 GitHub 问题 #9

例

```
/**
 * 学习如何使用 `{@link}` 标签.
 *
 * @remarks
 *
 * 链接可以指向URL: {@link https://github.com/microsoft/tsdoc}
 *
 * 链接可以指向API项: {@link Button}
 *
 * 您可以选择包含自定义链接文本: {@link Button | the Button class}
 *
 * 假设 `Button` 类是外部包的一部分。在这种情况下，我们可以在引用包时包含包名
 *
 * {@link my-control-library#Button | the Button class}
 *
 * 包名称可以包含NPM作用域和导入路径:
 *
 * {@link @microsoft/my-control-library/lib/Button#Button | the Button class}
 *
 * TSDoc标准将这种表示法称为“声明引用”。该符号支持对许多不同类型的TypeScript声明的引用。
 * 此符号最初是为在`{@link}`和`{@hericDoc}`标签中使用而设计的，但您也可以在自己的自定义标签中使用它。
 *
 * 例如，`Button`可以是TypeScript命名空间的一部分:
```

```
 *
 * {@link my-control-library#controls.Button | the Button class}
 *
 * 我们可以从类中引用一位成员:
 *
 * {@link controls.Button.render | the render() method}
 *
 * 如果静态成员和实例成员同名，我们可以使用选择器来区分它们:
 *
 * {@link controls.Button.(render:instance) | the render() method}
 *
 * {@link controls.Button.(render:static) | the render() static member}
 *
 * 这也是我们如何引用类的构造函数:
 *
 * {@link controls.(Button:constructor) | the class constructor}
 *
 * 有时，一个名称包含的特殊字符不是合法的TypeScript标识符:
 *
 * {@link restProtocol.IServerResponse."first-name" | the first name property}
 *
 * 这里有一个相当复杂的例子，其中函数名是ECMAScript 6符号，它是一个重载函数，使用标签选择器（使用`{@label}`TSDoc标签定义）:
 *
 * {@link my-control-library#Button.([UISymbols.toNumberPrimitive]:OVERLOAD_1)
 * | the toNumberPrimitive() static member}
 *
 * 有关“声明引用”符号的更多详细信息，请参阅TSDoc规范。
 * */
```

@module

该标签用于将注释标记为引用文件，而不是其后面的声明。 它可以选择性地用于重命名名称 TypeDoc 猜错的模块。`@module`

TSDoc 指定的 `@packageDocumentation` 标签也可以用来标记 引用文件的注释，但不能用于重命名模块。

注意：使用标记的注释块必须是文件中的第一个注释。 因此，建议将其放在文件顶部的任何 import 语句之前。`@module`

例

```
// file1.ts
/**
 * This is the doc comment for file1.ts
 *
 * Specify this is a module comment and rename it to my-module:
 * @module my-module
 */
import * as lib from "lib";

// file2.ts
/**
 * Specify this is a module comment without renaming it:
```

```
 * @module
 */
import * as lib from "lib";

// file3.ts
/**
 * This is *not* a doc comment for the file, it is a doc comment for the import.
 * Include the `@module` or `@packageDocumentation` tag to mark it as a file
comment.
 */
import * as lib from "lib";
```

@namespace

该标签可用于告诉 TypeDoc 将变量转换为命名空间。这将导致 要解析并记录为导出的变量/函数的任何属性。`@namespace`

例

```
const a = 1;
const b = () => 2;
const c = { a, b, c: 3 };
/** @namespace */
export const d = { ...c, d: 4 };

// will be documented as if you wrote

export namespace d {
    export const a = 1;
    export const b = () => 2;
    export const c = 3;
    export const d = 4;
}
```

@override

用法
此修饰符与 C# 或 Java 中的关键字具有相似的语义。对于成员函数或属性， 显式表示此定义正在覆盖（即重新定义）从
基类。基类定义通常标记为 .`override` `virtual`

文档工具可以强制,, 和/或 修饰符始终是 应用，但 TSDoc 标准不要求这样做。`@virtual` `@override` `@sealed`

例
在下面的代码示例中，覆盖 virtual member： `Child.render()` `Base.render()`

```
class Base {
    /** @virtual */
    public render(): void {
    }

    /** @sealed */
    public initialize(): void {
```

```
    }
}


class Child extends Base {
    /** @override */
    public render(): void;
}
```

@overload

该标记被识别用于 JavaScript 项目，自 TypeScript 5.0 以来，这些项目可以使用它来声明重载。它会自动从渲染的 带有 --excludeTags
选项的文档`@overload`

例

```
/**
 * @overload
 * @param {string} value first signature
 * @return {void}
 */

/**
 * @overload
 * @param {number} value second signature
 * @param {number} [maximumFractionDigits]
 * @return {void}
 */

/**
 * @param {string | number} value
 * @param {number} [maximumFractionDigits]
 */
function printValue(value, maximumFractionDigits) {
}
```

@packageDocumentation

用法
用于表示描述整个 NPM 包的文档注释（而不是属于 添加到该包中）。注释位于 *.d.ts 文件中，该文件充当 包，它应该是该文件中遇到的第一个注释。包含
标签绝不能用于描述单个 API 项目。`@packageDocumentation` `/**` `@packageDocumentation`

例

```
// Copyright (c) Example Company. All rights reserved. Licensed under the MIT
license.

/**
 * 用于创建页面的库.
 *
 * @remarks
```

```
 * The `widget-lib` defines the {@link IWidget} interface and {@link Widget}
class,
 * which are used to build widgets.
 *
 * @packageDocumentation
 */

/**
 * Interface implemented by all widgets.
 * @public
 */
export interface IWidget {
    /**
     * Draws the widget on the screen.
     */
    render(): void;
}
```

@param

用法
用于记录函数参数。标签后跟参数名称，后跟连字符 后跟描述。`@param`

例

```
/**
 * Returns the average of two numbers.
 *
 * @remarks
 * This method is part of the {@link core-library#Statistics | Statistics
subsystem}.
 *
 * @param x - The first input number
 * @param y - The second input number
 * @returns The arithmetic mean of `x` and `y`
 *
 * @beta
 */
function getAverage(x: number, y: number): number {
    return (x + y) / 2.0;
}
```

@privateRemarks

用法
开始一个不面向公众受众的其他文档内容部分。 工具必须从 API 参考网站中省略整个部分，生成的 *.d.ts 文件， 以及包含内容的任何其他输出。

例

```
/**
 * 摘要部分应该简短。在文档网站上，它将显示在一个页面上，该页面列出了许多不同API项目的摘要。在单个项目的详细信息页面上，将显示摘要，然后是备注部分（如果有的话）。
 *
 * @remarks
 *
 * API项的主要文档被分为一个简短的“摘要”部分，后面可选地加上包含其他详细信息的`@remarks`块。
 *
 * @privateRemarks
 *
 * `@privateRemarks`标签开始了一段不面向外部受众的附加评论。文档工具必须从API参考网站中省略此内容。在生成规范化的.d.ts文件时，也应该省略它。
 */
```

@public

建议的含义
指定 API 项的发布阶段为 “public”。它已正式发布给第三方开发者， 并且其签名保证稳定（例如，遵循语义版本控制规则）。

例

```
/**
 * 表示目录中的一本书。
 * @public
 */
export class Book {
    /**
     * 书的标题。
     * @internal
     */
    public get _title(): string;

    /**
     * 书的作者。
     */
    public get author(): string;
}
```

在此示例中，从包含类 而标记为 “internal”。 `Book.author` `@publicBook._title`

@property

标签可用于向当前反射的子对象添加注释。 它旨在与 `@namespace` 和 `@interface` 标签一起使用可能没有方便的位置来包含每个成员的评论。 `@property` `@prop`

例

```
/**
 * This will be displayed as an interface
 * @property a comment for a
 * @prop b comment for b
 * @interface
 */
```

```
export type Resolved = Record<"a" | "b" | "c", string>;


// will be documented as if you wrote

/** This will be displayed as an interface */
export interface Resolved {
    /** comment for a */
    a: string;
    /** comment for b */
    b: string;
    c: string;
}
```

@private

通常不应使用此标记，并且可能会在将来的发行版中删除此标记。 该标签将反射的可见性覆盖为私有。 `@private`

例

```
export class Visibility {
    /** @private */
    member = 123;
}


// Will be documented as:
export class Visibility {
    private member = 123;
}
```

@protected

通常不应使用此标记，并且可能会在将来的发行版中删除此标记。 该标记将覆盖要保护的反射的可见性。@protected

例

```
export class Visibility {
    /** @protected */
    member = 123;
}


// Will be documented as:
export class Visibility {
    protected member = 123;
}
```

@readonly

用法
此修饰符标记指示 API 项应记录为只读，即使 TypeScript type system 可能另有指示。例如，假设一个类属性有一个 setter 函数，该函数始终
引发一个异常，说明无法分配该属性;在这种情况下，修饰符 ，以便该属性在文档中显示为只读。
`@readonly`

例

```
export class Book {
    /**
     * 从技术上讲，属性有一个设置器，但出于文档目的，它应该以只读形式呈现。
     *
     * @readonly
     */
    public get title(): string {
        return this._title;
    }

    public set title(value: string) {
        throw new Error("This property is read-only!");
    }
}
```

@remarks

参见 @privateRemarks

@returns

用法
用于记录函数的返回值。

例

```
/**
 * Returns the average of two numbers.
 *
 * @remarks
 * This method is part of the {@link core-library#Statistics | Statistics
subsystem}.
 *
 * @param x - The first input number
 * @param y - The second input number
 * @returns The arithmetic mean of `x` and `y`
 *
 * @beta
 */
function getAverage(x: number, y: number): number {
    return (x + y) / 2.0;
}
```

@sealed

用法
此修饰符与 C# 或 Java 中的关键字具有相似的语义。对于类，指示 子类不得从类继承。对于成员函数或属性，指示子类
不得覆盖（即重新定义）成员。`sealed`

文档工具可以强制 ,, 和/或 修饰符始终是 应用，但 TSDoc 标准不要求这样做。`@virtual` `@override` `@sealed`

例
在下面的代码示例中，覆盖虚拟成员 ， 但不能被覆盖，因为它被标记为 “sealed”。`Child.render()` `Base.render()` `Base.initialize()`

```
class Base {
    /** @virtual */
    public render(): void {
    }

    /** @sealed */
    public initialize(): void {
    }
}


class Child extends Base {
    /** @override */
    public render(): void;
}
```

@see

用法
用于构建对 API 项或其他资源的引用列表，这些资源可能与 当前项目。

> **Note**
> JSDoc 会尝试在 .因为这是模棱两可的 对于纯文本，TSDoc 需要一个显式的标签来制作超链接。`@see` `{@link}`

例

```
/**
 * Parses a string containing a Uniform Resource Locator (URL).
 * @see {@link ParsedUrl} for the returned data structure
 * @see {@link https://tools.ietf.org/html/rfc1738|RFC 1738}
 * for syntax
 * @see your developer SDK for code samples
 * @param url - the string to be parsed
 * @returns the parsed result
 */
function parseURL(url: string): ParsedUrl;
```

`@see`是一个块标签。每个块都将成为引用列表中的一个项目。例如，文档 系统可能会按如下方式呈现上述块：

```
`function parseURL(url: string): ParsedUrl;`

Parses a string containing a Uniform Resource Locator (URL).

## See Also

- ParsedUrl for the returned data structure
- RFC 1738 for syntax
- your developer SDK for code samples
```

@throws

用法
用于记录可能由函数或属性引发的异常类型。

应使用单独的块来记录每个异常类型。此标签用于信息 目的，并且不限制抛出其他类型的这是建议的，但不是必需的，
使块以仅包含异常名称的行开头。@throws@throws

例如：

```
/**
 * Retrieves metadata about a book from the catalog.
 *
 * @param isbnCode - the ISBN number for the book
 * @returns the retrieved book object
 *
 * @throws {@link IsbnSyntaxError}
 * This exception is thrown if the input is not a valid ISBN number.
 *
 * @throws {@link book-lib#BookNotFoundError}
 * Thrown if the ISBN number is valid, but no such book exists in the catalog.
 *
 * @public
 */
function fetchBookByIsbn(isbnCode: string): Book;
```

@typeParam

用法
用于记录泛型参数。标签后跟一个参数 name 开头，后跟一个连字符，后跟一个 description。TSDoc 解析器识别 this 语法，并将其提取到
DocParamBlock 节点中。`@typeParam`

例

```
/**
 * Alias for array
 *
 * @typeParam T - Type of objects the list contains
 */
type List<T> = Array<T>;
```

```
/**
 * Wrapper for an HTTP Response
 * @typeParam B - Response body
 * @param <H> - Headers
 */
interface HttpResponse<B, H> {
    body: B;
    headers: H;
    statusCode: number;
}
```

@template

该标签用于记录函数、方法、类、接口或类型别名的类型参数。`@template`

TypeDoc 将标签识别为与 JavaScript 兼容的别名 通过文档注释使用 TypeScript 的项目。对于 TypeScript 项目，应首选 TSDoc 标准 `@typeParam` 标记。`@template` `@typeParam`

例

```
/**
 * @template {string} T - the identity type
 */
export function identity<T>(x) {
  return x;
}
```

@virtual

用法
此修饰符与 C# 或 Java 中的关键字具有相似的语义。对于成员函数或属性， 显式指示子类可以覆盖（即重新定义）成员。`virtual`

文档工具可以强制 、 、 和/或 修饰符始终是 应用，但 TSDoc 标准不要求这样做。`@virtual` `@override` `@sealed`

例
在下面的代码示例中，覆盖 virtual member： `Child.render()` `Base.render()`

```
class Base {
    /** @virtual */
    public render(): void {
    }

    /** @sealed */
    public initialize(): void {
    }
}


class Child extends Base {
    /** @override */
    public render(): void;
}
```

@satisfies

此标签与 JSDoc 中的 TypeScript 5.0 的 @satisfies 支持相同。

默认情况下，它由 --excludeTags 选项隐藏。

例

```
/**
 * @satisfies {ConfigSettings}
 */
export const myConfigSettings = { ... };
```