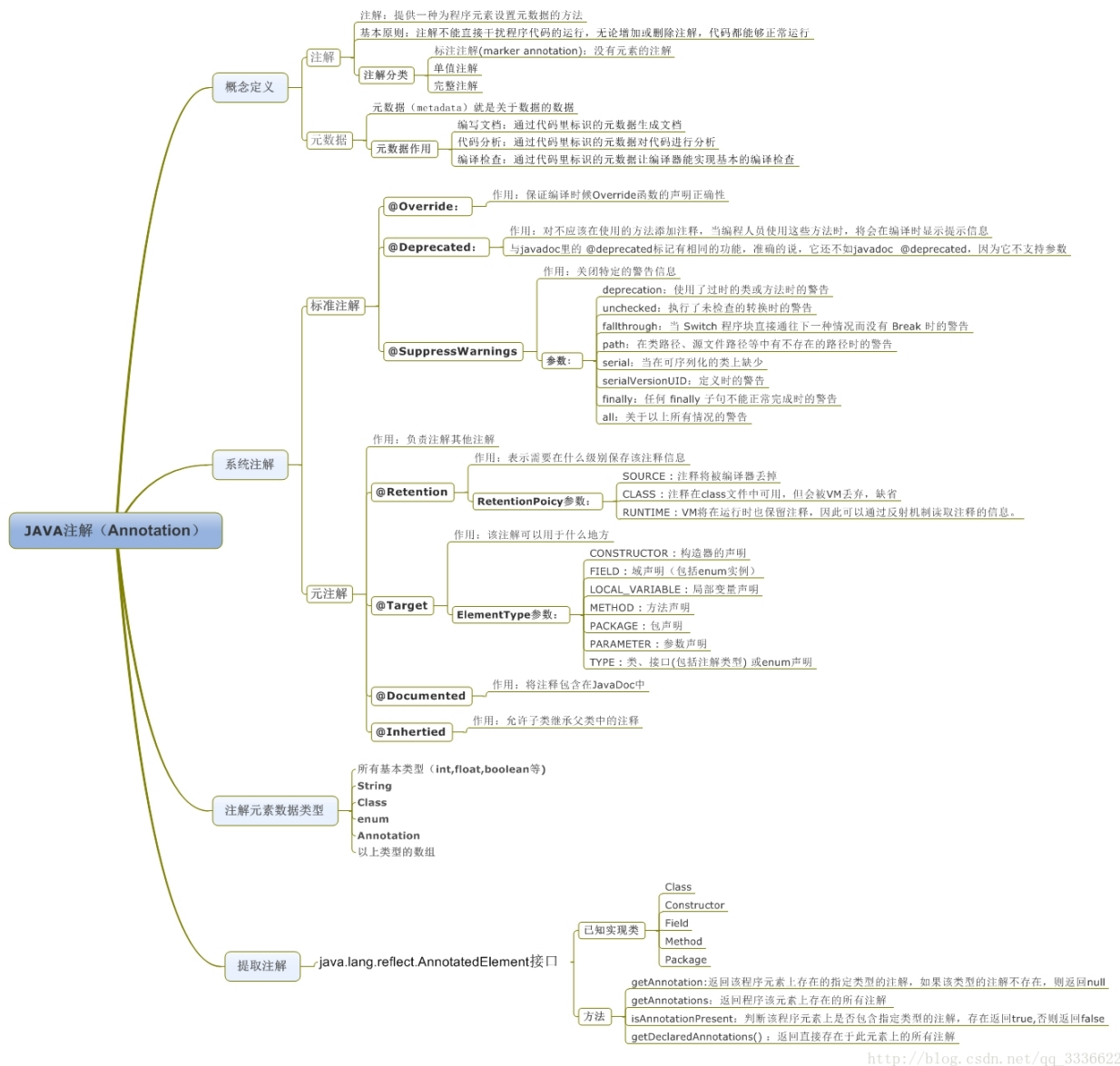


注解：



http://blog.csdn.net/qq_33366229

- * 注解概念：说明程序的。给计算机看的
- * 注释：用文字描述程序的。给程序员看的

* 定义：注解（Annotation），也叫元数据。一种代码级别的说明。注解是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。

* 概念描述：

- * JDK1.5之后的新特性
- * 说明程序的
- * 使用注解：@注解名称

* 作用分类：

①编写文档：通过代码里标识的注解生成文档【生成文档doc文档，即api文档】

注：@param代表参数

IDEA中添加javadoc注解的方法是将光标停在类名或者方法名上，然后alt+enter，出现几个选项，选择添加Javadoc即可

②代码分析：通过代码里标识的注解对代码进行分析【使用反射】

③编译检查：通过代码里标识的注解让编译器能够实现基本的编译检查

【Override】

* JDK中预定义的一些注解

* **@Override**：检测被该注解标注的方法是否是继承自父类(接口)的----（否则会编译不通过，报红色）

* **@Deprecated**：该注解标注的内容，表示已过时----（仍然可以使用，只是不推荐使用，不会爆红；不直接删掉旧方法的原因是有些低版本的程序可能会需要）

注：**Data**类里有很多方法都过时了，推荐使用日历类

注：一般自己写的方法上，若写了**@Deprecated**后，再调用该写的方法后会画上一个横线

* **@SuppressWarnings**：压制警告，一般传递参数all

@SuppressWarnings("all")

注：大多数情况写在类上面，消除代码行中提示的黄线；

只写all参数不用写属性名的原因是**@SuppressWarnings**只有一个属性且名字是value且返回值是String类型

* 自定义注解

* 格式：

元注解

```
public @interface 注解名称{  
    属性列表;  
}
```

* 本质：注解本质上就是一个接口，该接口默认继承Annotation接口

* **public interface MyAnno extends**

java.lang.annotation.Annotation {}

* 属性：在这里指接口中的抽象方法（这里属性是接口里可以定义的内容：常量、成员方法等）（抽象方法：指没有方法体的方法，即在方法声明时没有{}及其中的内容，而是直接在声明时在方法名后加上分号结束）（对于接口来说，里面的方法一定是抽象方法，即使比如在接口里定义方法**String show()**；其实是跟**public abstract String show**

()；一样的，接口会自动为它加上**public abstract**；对于抽象类来说里面的方法不一定必须时抽象方法）

* 要求：

1. 注解里的属性的返回值类型有下列取值

* 基本数据类型（即4类8种）

* **String**

* 枚举

* 注解

* 以上类型的数组

注：不允许void，会爆红

2. 定义了属性，在使用时需要给属性赋值(这就是为什么叫抽象方法为属性的原因；所以在给方法起名字的时候尽量起的像key名)

1. 如果定义属性时，使用default关键字给属性默认初始化值，则使用注解时，可以不进行属性的赋值。

2. 如果只有一个属性需要赋值，并且属性的名称是value，则value可以省略，直接定义值即可。

3. 数组赋值时，值使用{}包裹。如果数组中只有一个值，则{}可以省略

* 元注解：用于描述注解的注解（元注解本质上还是一个注解）（我们在定义注解的时候可以将我们的注解上面加元注解来描述我们自己定义的这个注解）

* @Target：描述注解能够作用的位置

ElementType取值：

* TYPE：可以作用于类上

* METHOD：可以作用于方法上

* FIELD：可以作用于成员变量上

@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})

注： @Target就一个属性，名字叫做value，这个属性的返回值类型是ElementType[]，ElementType[]是一个枚举，这个枚举里面有三个取值

* @Retention：描述注解被保留的阶段（JAVA代码有三个阶段，源码阶段、class类对象阶段、Runtime运行时阶段）

* @Retention(RetentionPolicy.RUNTIME)：它描述的注解会保留到@Documented描述的注解所注解的代码的class字节码文件中，并被JVM读取到（注：我们自己定义的注解的元注解通常都取RUNTIME这个值）

* @Retention(RetentionPolicy.CLASS)：也会保留到class字节码文件中，但并不会被JVM读取到

* @Retention(RetentionPolicy.SOURCE)：不会保留在class字节码文件里

注：@Retention就一个属性，名字叫做value，这个属性的返回值类型是RetentionPolicy，RetentionPolicy是一个枚举，这个枚举里面有三个取值

* @Documented：描述注解是否被抽取到api文档中，即@Documented描述的注解所注解的代码会被抽取到我们javadoc的文档中

* @Inherited：@Inherited描述的注解所注解的代码的子类会继承@Inherited描述的注解

* 在程序使用(解析)注解：获取注解中定义的属性值

1. 获取注解定义的位置的对象（Class, Method, Field）

2. 获取指定的注解

* getAnnotation(Class)

//其实就是在内存中生成了一个该注解接口的子类实现对象

```
public class ProImpl implements Pro{
```

```

        public String className(){
            return "cn.itcast.annotation.Demo1";
        }
        public String methodName(){
            return "show";
        }
    }

```

3. 调用注解中的抽象方法获取配置的属性值

* 案例：简单的测试框架

* 小结：

1. 以后大多数时候，我们会使用注解，而不是自定义注解
2. 注解给谁用？
 1. 编译器
 2. 给解析程序用
3. 注解不是程序的一部分，可以理解为注解就是一个标签

- toString上面的Override也算是注解，它是用来编译检查的，它会去检查被注解标识的这个方法是否是复写的父类的方法，如果不是则编译失败；Object类里右toString方法

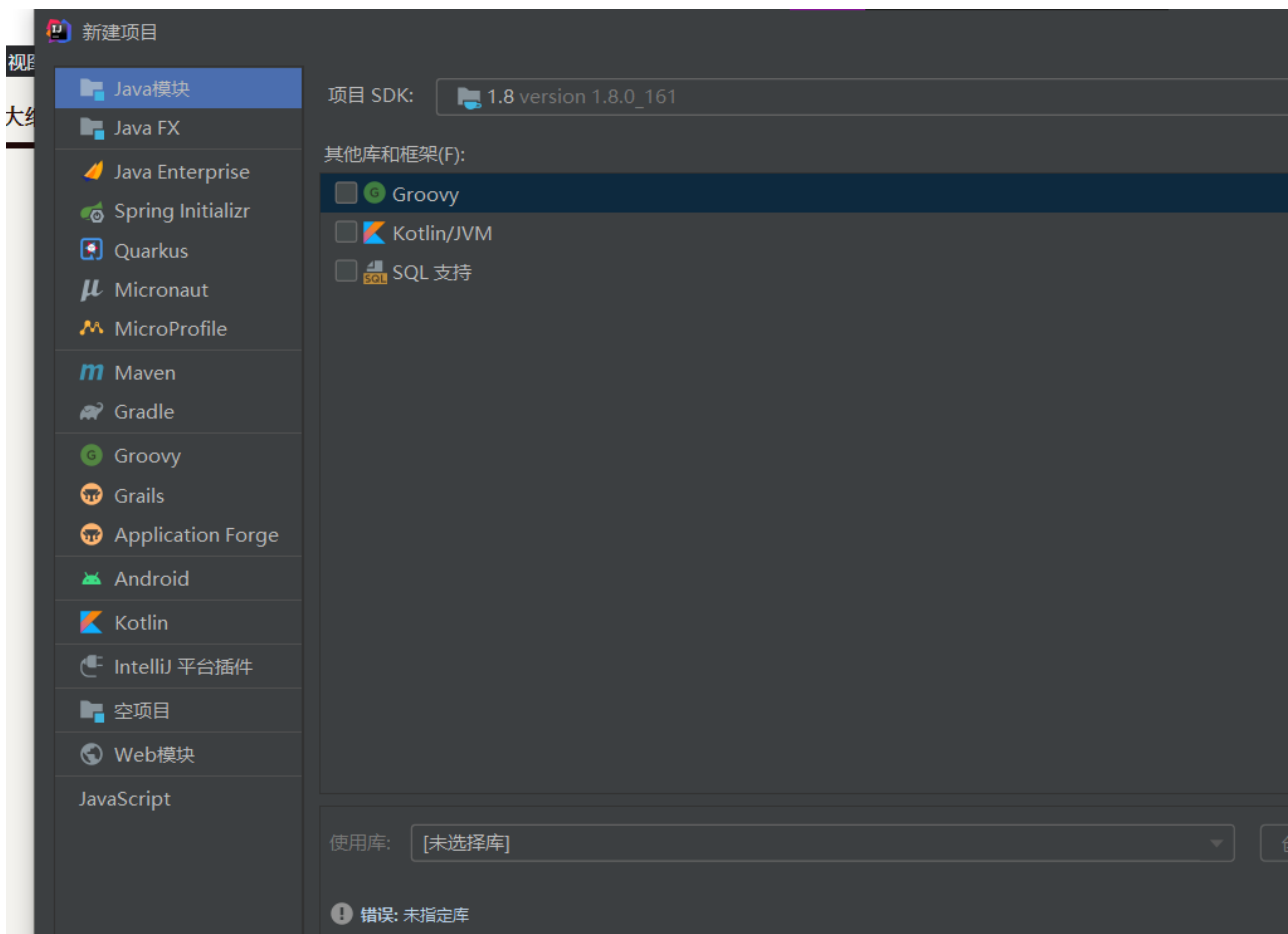
```

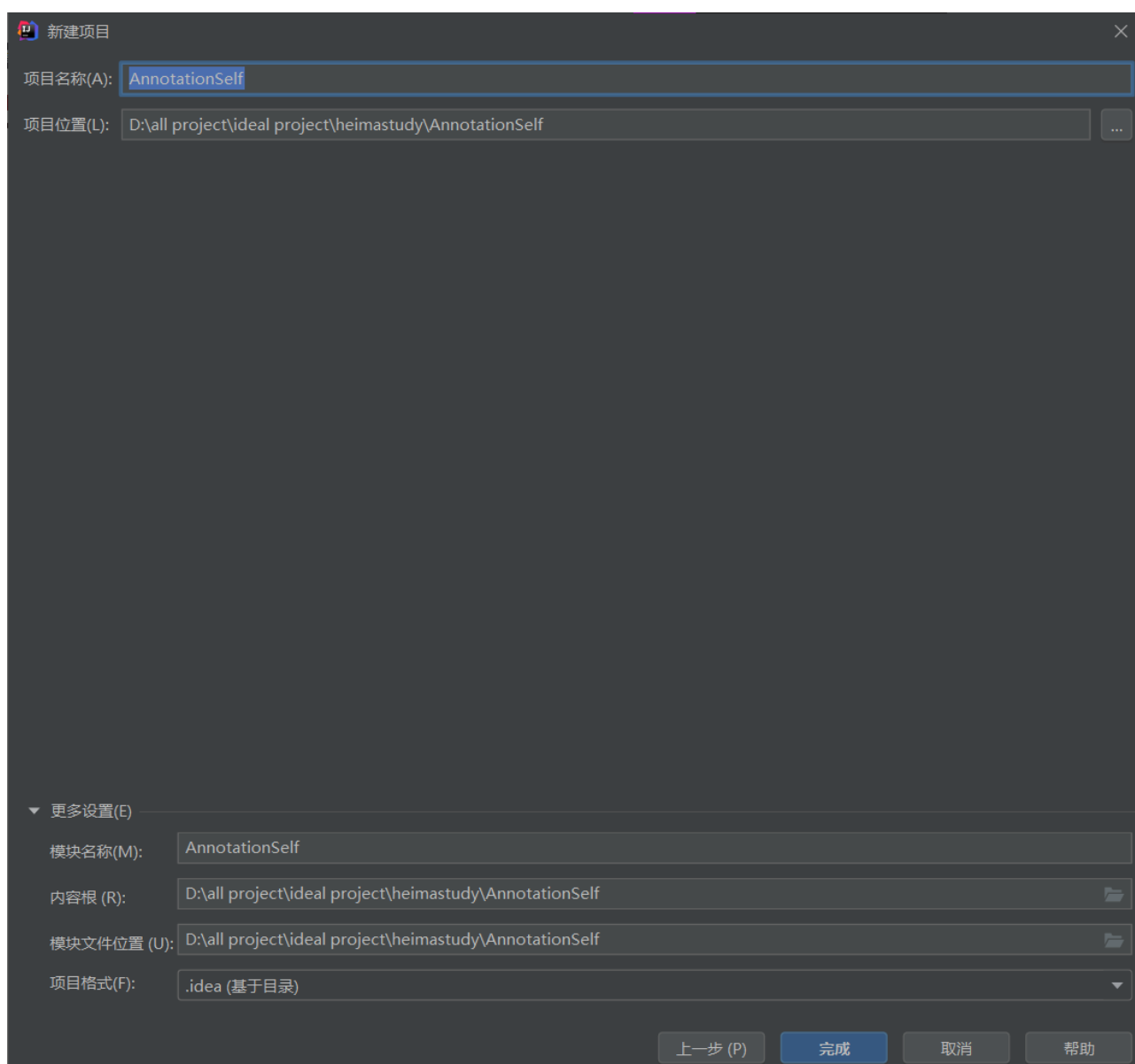
@Override
public String toString() {
    return "Person{" +
        "name='" + name + '\'' +
        ", age=" + age +
        ", a='" + a + '\'' +
        ", b='" + b + '\'' +
        ", c='" + c + '\'' +
        ", d='" + d + '\'' +
        '}';
}

```

```
@Override
public String toString1() {
    return "Person{" +
        "name='" + name + '\'' +
        ", age=" + age +
        ", a='" + a + '\'' +
        ", b='" + b + '\'' +
        ", c='" + c + '\'' +
        ", d='" + d + '\'' +
        '}';
}
```

IDEA新建





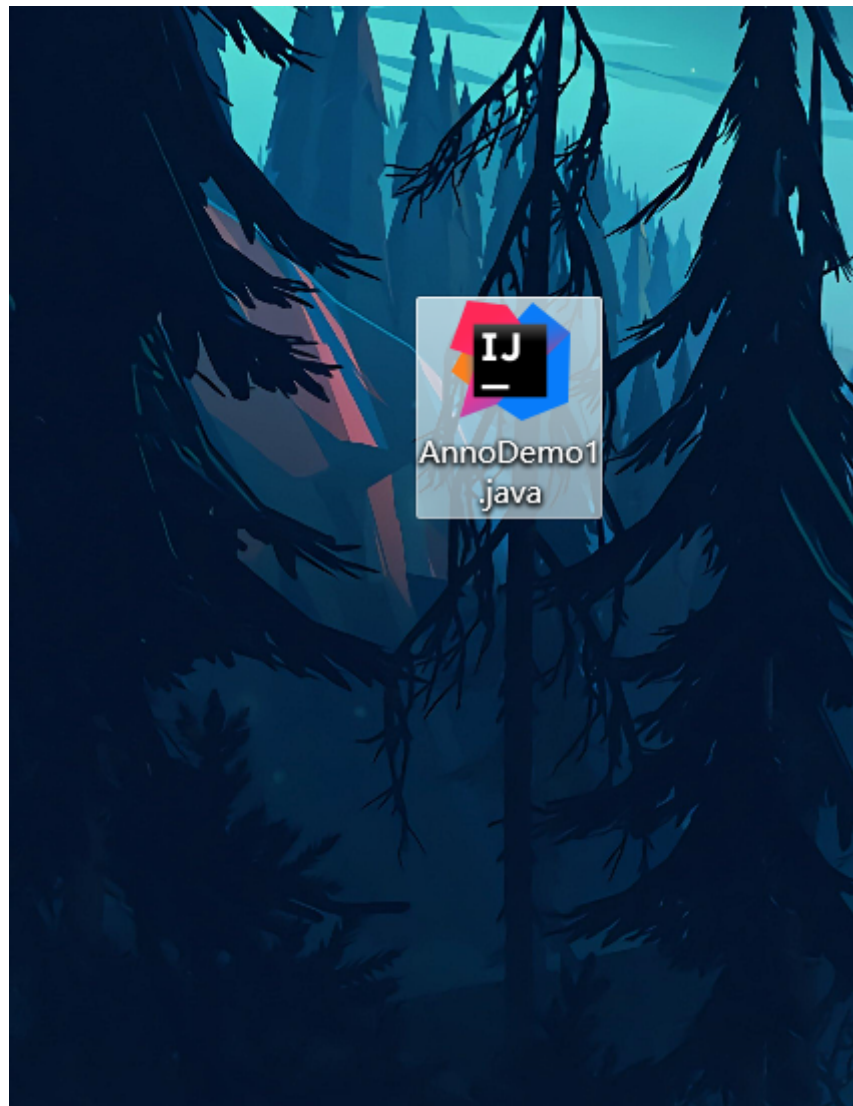
javadoc

- 写上如下代码

```
1 package cn.itcast.annotation;  
2 /**  
3  * 注解javadoc演示  
4  * 这下面的@也是注解  
5  * @author itcat  
6  * @version 1.0  
7  * @since 1.5  
8  */  
9 public class AnnoDemo1 {  
10     /**  
11      * 计算两数的和  
12      * @param a 整数  
13      * @param b 整数  
14      * @return 两数的和  
15      */  
16     /**  
17      * @Description: add  
18      * * @param a  
19      * * @param b  
20      * @return: int  
21      * @Author: Mr.Cao  
22      * @Date: 2018/6/18  
23      */  
24     public int add(int a, int b) { return a + b; }  
27     /**  
28      * @Description: summ  
29      * @Param: [a, b]  
30      * @return: int  
31      * @Author: Mr.Cao  
32      * @Date: 2018/6/18    tab+a+/  
33      */
```

```
34     /**  
35      * @Author: Mr.Cao  
36      * @Date: 2018/6/18    tab+a+/  
37      */  
38     public int summ(int a,int b) { return a+b; }  
39 }
```

- 复制这个类到桌面



- 用记事本打开删除包信息

使用ANSI编码（即自动使用跟本地操作系统一样的编码方式，即GBK）-----此操作的原因是防止生成的api乱码

C:\Users\15516535379\Desktop\AnnoDemo1.java - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?



AnnoDemo1.java

```
1 package cn.itcast.annotation;
2 /**
3  * 注解javadoc演示
4  * 这下面的@也是注解
5  * @author itcat
6  * @version 1.0
7  * @since 1.5
8  */
9 public class AnnoDemo1 {
10     /**
11      * 计算两数的和
12      * @param a 整数
13      * @param b 整数
14      * @return 两数的和
15      */
16     /**
17      * @Description: add
18      *      * @param a
19      *      * @param b
20      * @return: int
21      * @Author: Mr.Cao
22      * @Date: 2018/6/18
23      */
24     public int add(int a, int b ){
25         return a + b;
26     }
27     /**
28      * @Description: summ
29      * @Param: [a, b]
30      * @return: int
31      * @Author: Mr.Cao
32      * @Date: 2018/6/18      tab+a+
33      */
34     public int summ(int a,int b){
35         return a+b;
36     }}
37 }
```

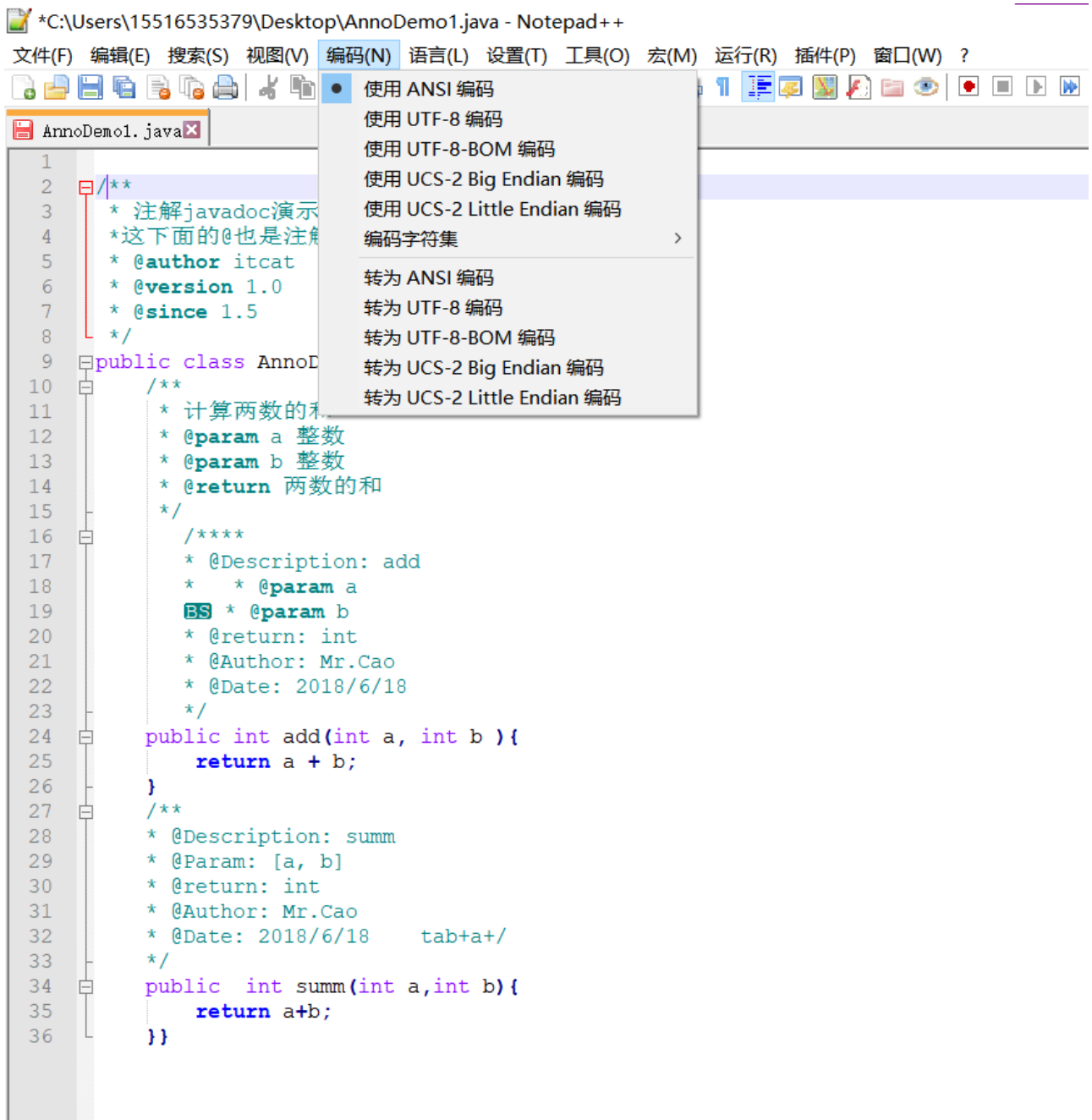
*C:\Users\15516535379\Desktop\AnnoDemo1.java - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?



AnnoDemo1.java

```
1
2  /**
3   * 注解javadoc演示
4   * 这下面的@也是注解
5   * @author itcat
6   * @version 1.0
7   * @since 1.5
8   */
9  public class AnnoDemo1 {
10     /**
11      * 计算两数的和
12      * @param a 整数
13      * @param b 整数
14      * @return 两数的和
15     */
16     /**
17      * @Description: add
18      *      * @param a
19      *      * @param b
20      * @return: int
21      * @Author: Mr.Cao
22      * @Date: 2018/6/18
23     */
24     public int add(int a, int b ){
25         return a + b;
26     }
27     /**
28      * @Description: summ
29      * @Param: [a, b]
30      * @return: int
31      * @Author: Mr.Cao
32      * @Date: 2018/6/18      tab+a+
33     */
34     public int summ(int a,int b){
35         return a+b;
36     }
37 }
```



- shift+右键打开命令行，输入javadoc AnnoDemo1.java即可自动下载文档

C:\Windows\System32\cmd.exe

C:\Users\15516535379\Desktop>javadoc AnnoDemol.java

正在加载源文件AnnoDemol.java...

正在构造 Javadoc 信息...

标准 Doclet 版本 1.8.0_161

正在构建所有程序包和类的树...

正在生成. \AnnoDemol.html...

AnnoDemol.java:20: 错误: 未知标记: Description

* @Description: add

AnnoDemol.java:24: 错误: 未知标记: Author

* @Author: Mr.Cao

AnnoDemol.java:25: 错误: 未知标记: Date

* @Date: 2018/6/18

AnnoDemol.java:27: 警告: a没有 @param

public int add(int a, int b){

AnnoDemol.java:27: 警告: b没有 @param

public int add(int a, int b){

AnnoDemol.java:31: 错误: 未知标记: Description

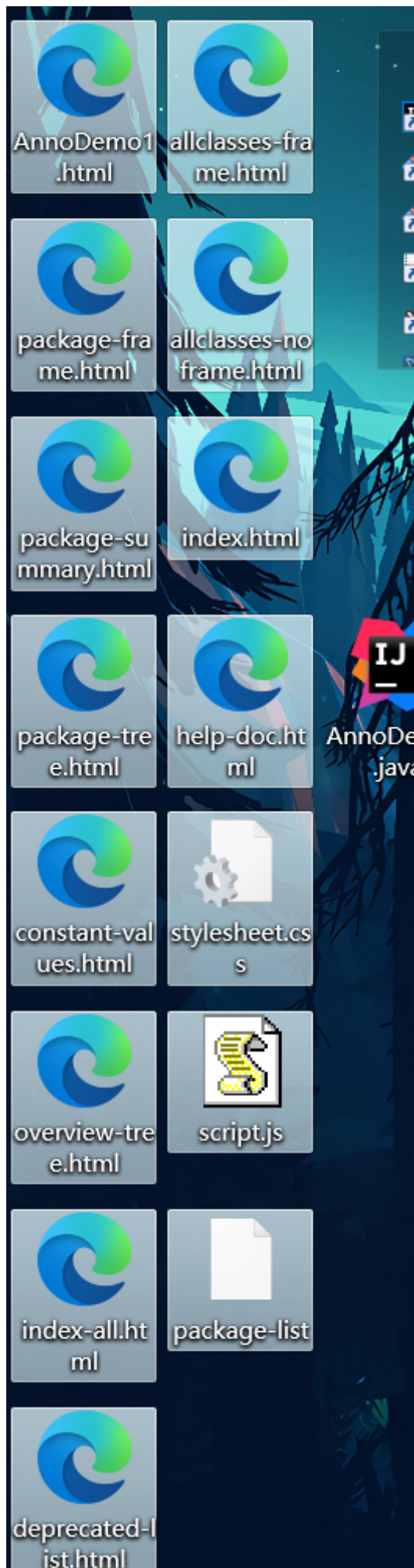
* @Description: summ

AnnoDemol.java:32: 错误: 未知标记: Param

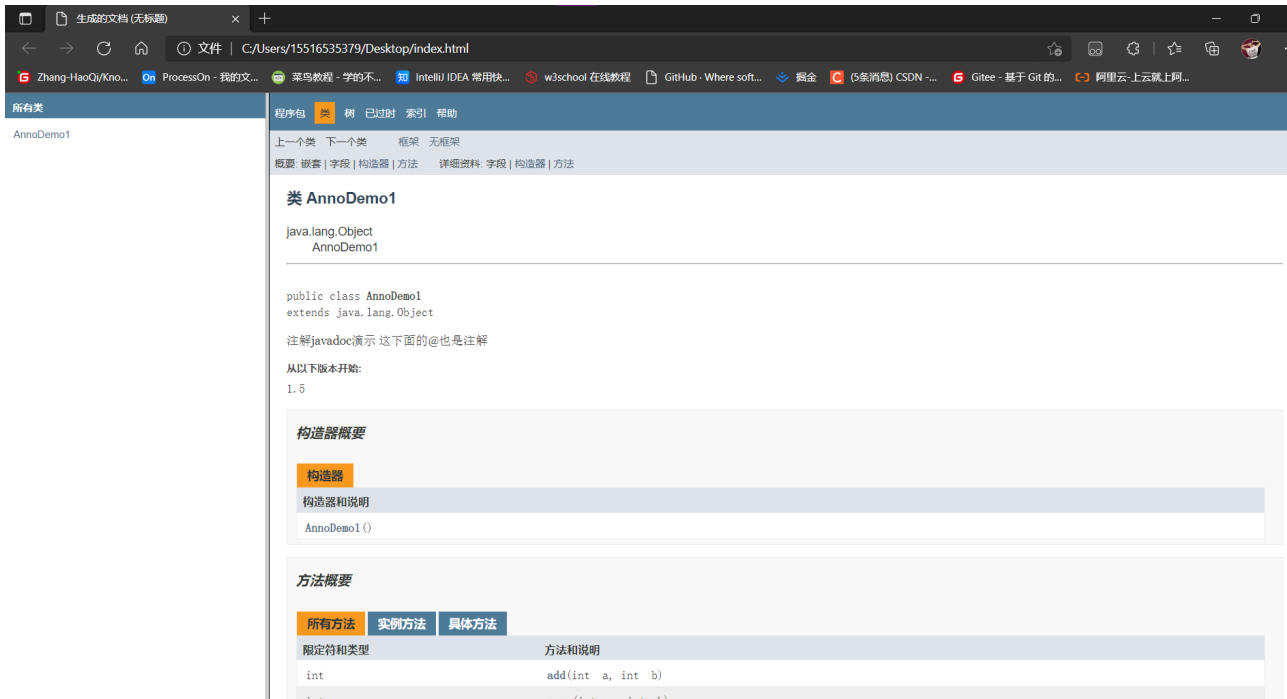
* @Param: [a, b]

AnnoDemol.java:34: 错误: 未知标记: Author

* @Author: Mr.Cao

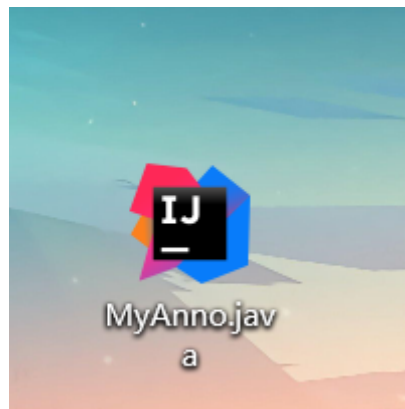


- 打开index.html

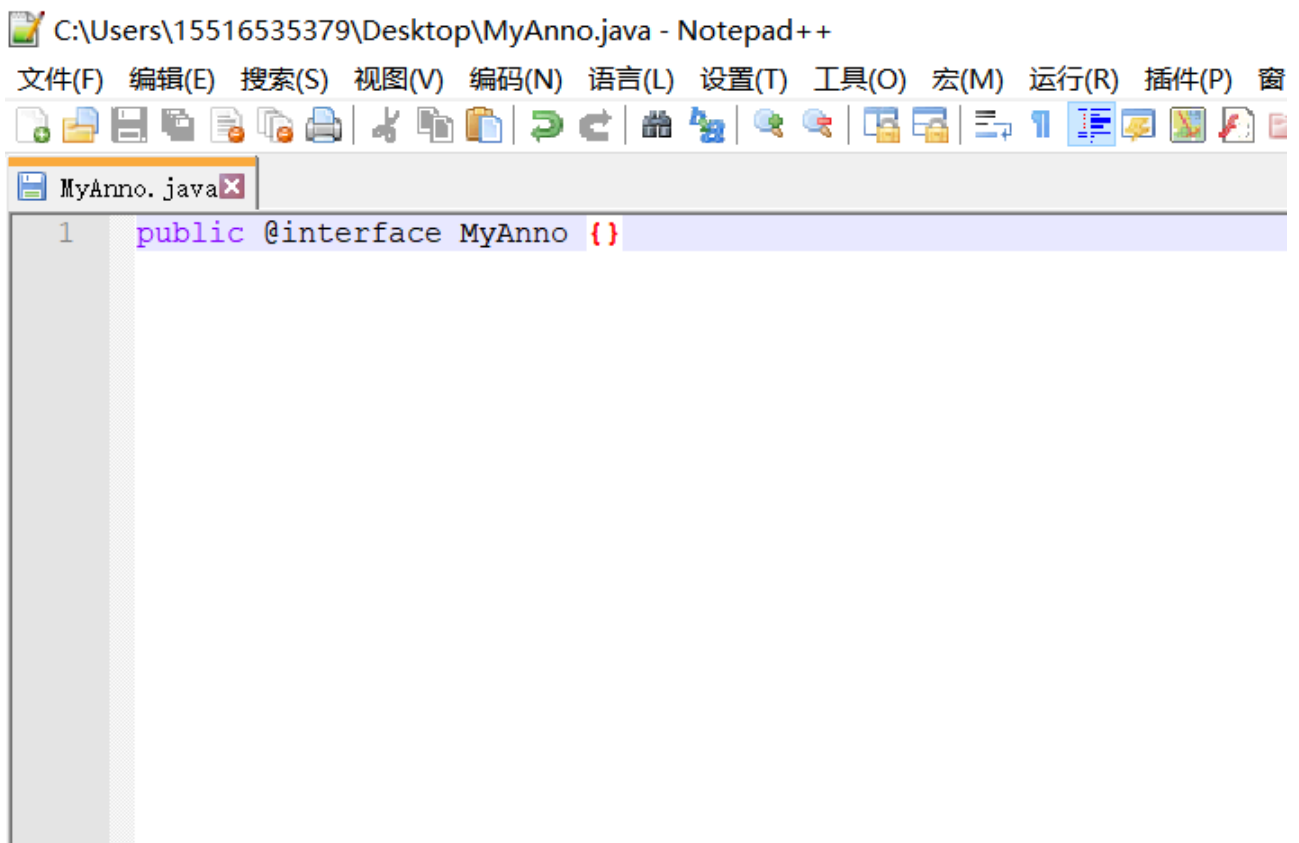


反编译自定义注解

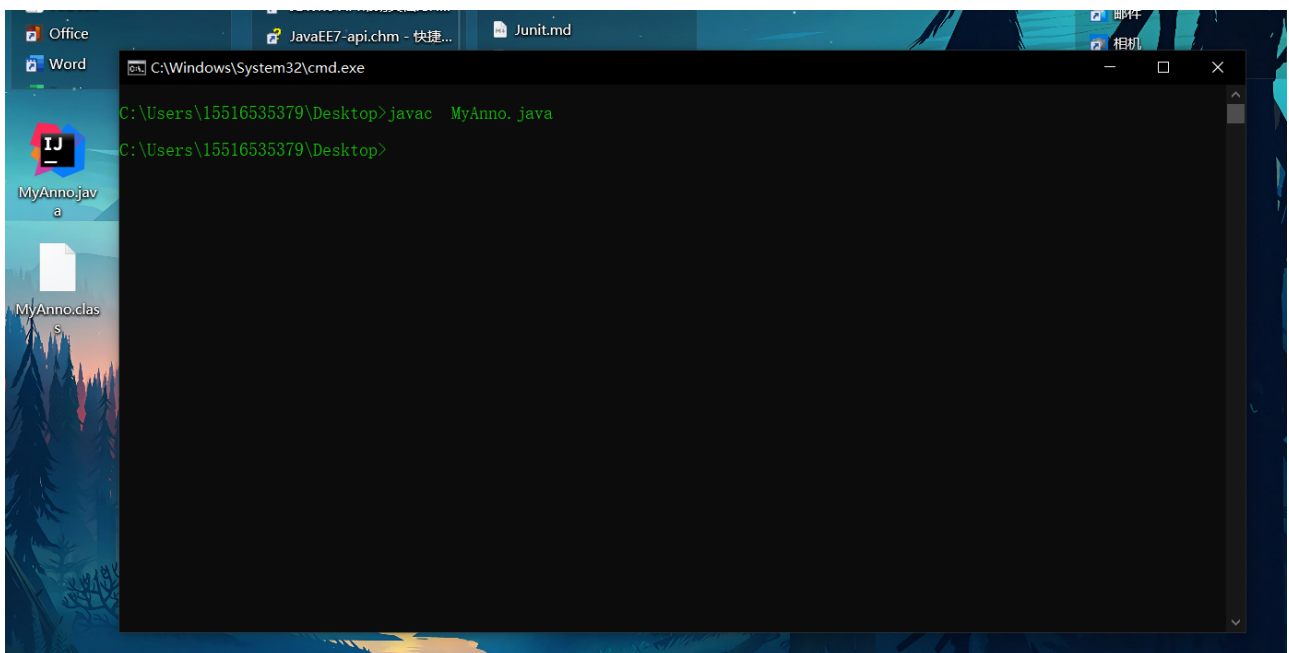
- 新建MyAnno.txt文件,再改名字为MyAnno.java(由于新建的文件默认是GBK,所以不需要后期调整编码方式)



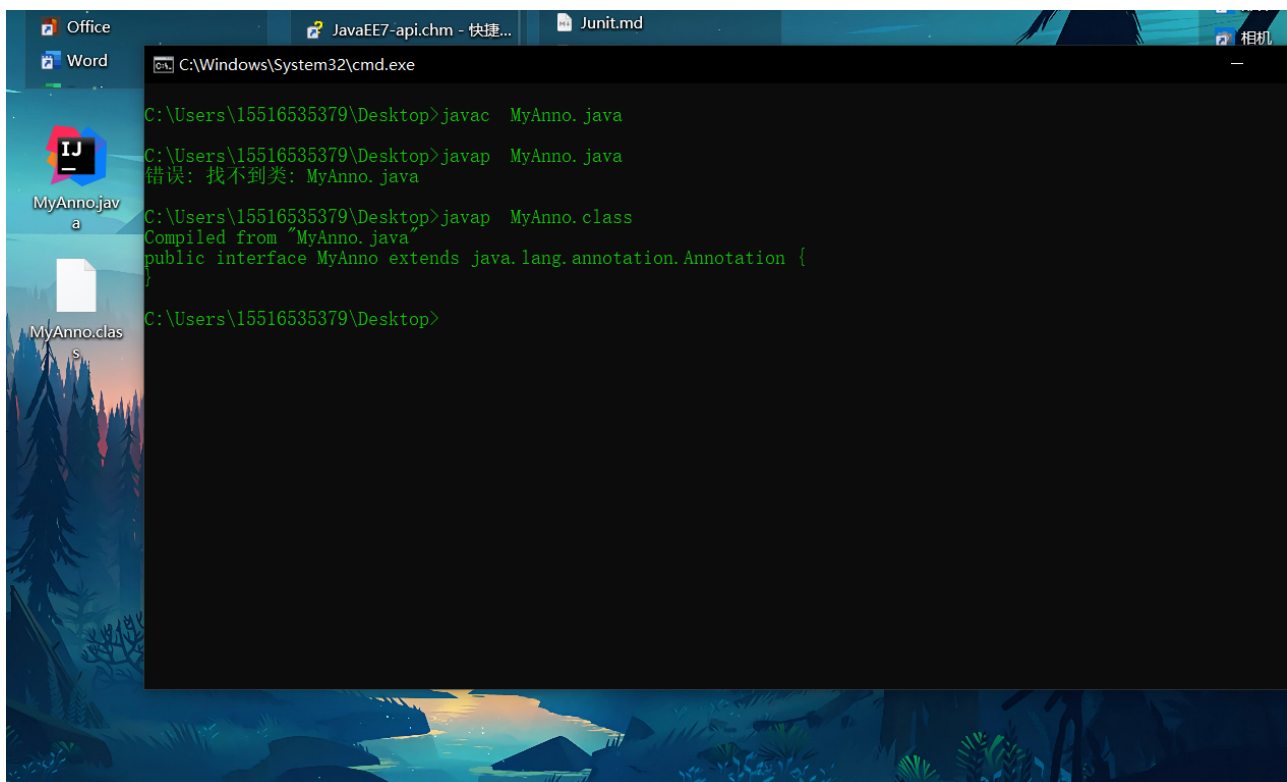
- 写入以下代码



- `javac MyAnno.java` 编译生成字节码文件

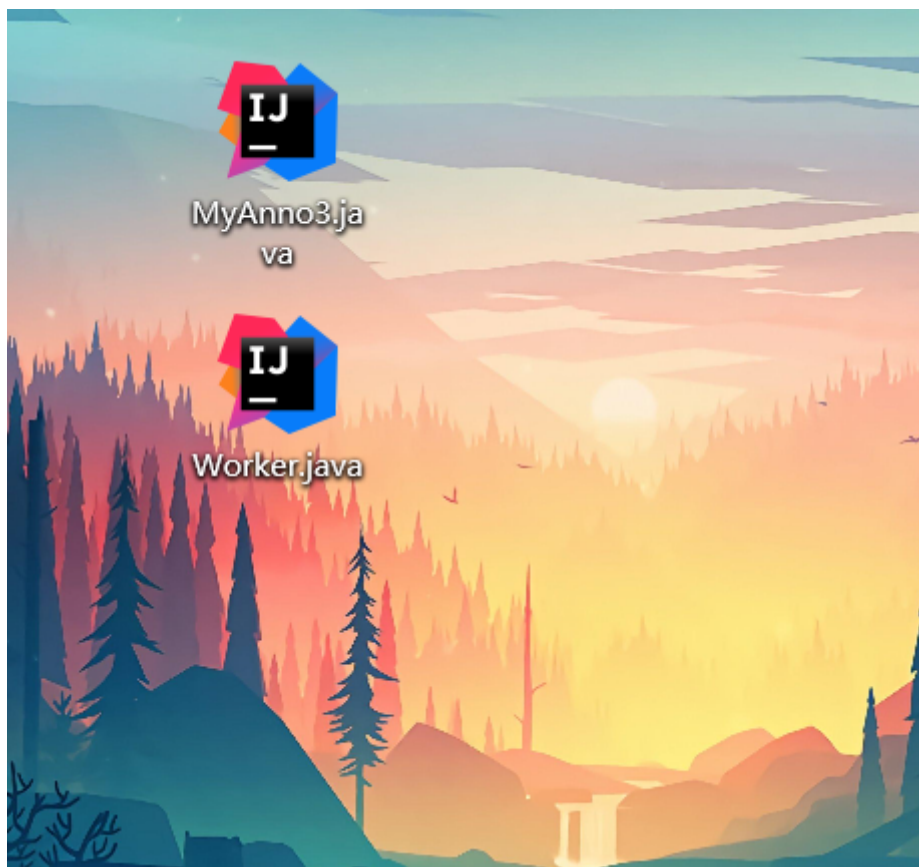


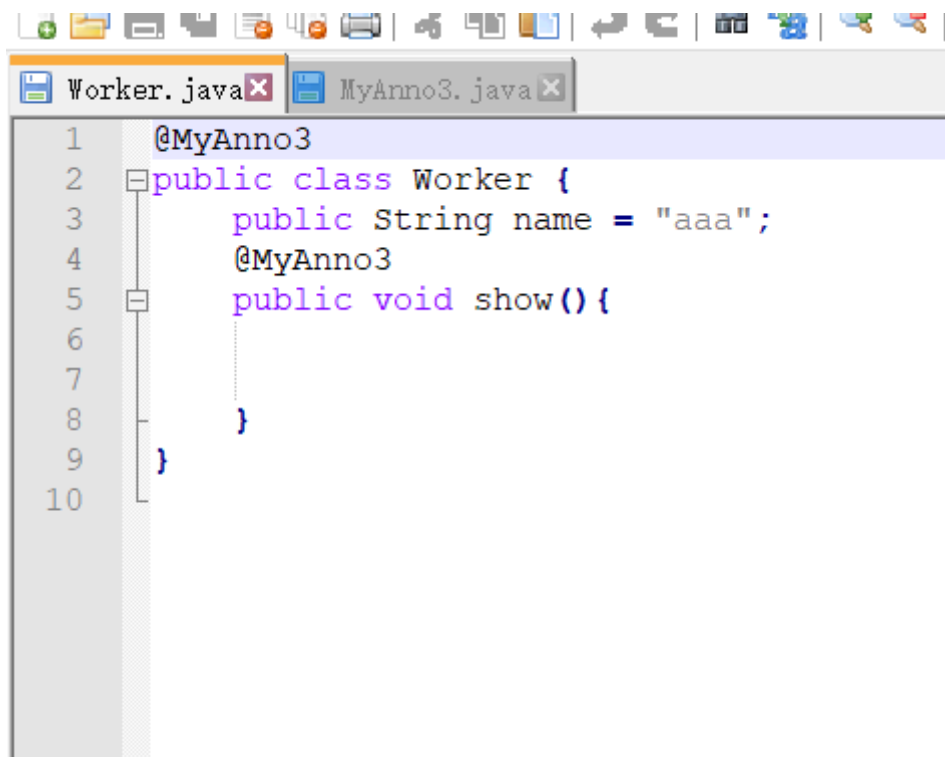
- `javap MyAnno.class` 反编译，会重新在命令行里生成一个java文件代码（注：只会在命令行里生成，不会产生新文件），发现代码有所变化：



@Documented

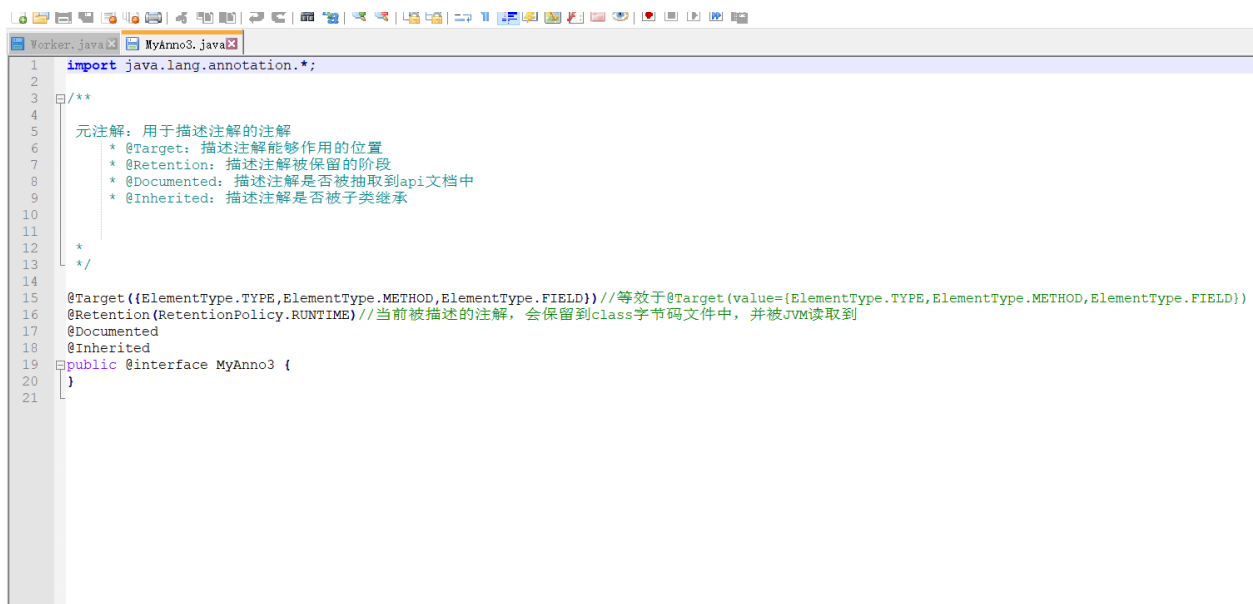
- 桌面新建如下两个文件，并设置使用ANSI编码（即自动使用跟本地操作系统一样的编码方式，即GBK）-----此操作的原因是防止生成的api乱码





The screenshot shows an IDE with two tabs: Worker.java and MyAnno3.java. The Worker.java tab is active, displaying the following code:

```
1 @MyAnno3
2 public class Worker {
3     public String name = "aaa";
4     @MyAnno3
5     public void show() {
6
7     }
8 }
9
10
```



The screenshot shows an IDE with two tabs: Worker.java and MyAnno3.java. The MyAnno3.java tab is active, displaying the following code:

```
1 import java.lang.annotation.*;
2
3 /**
4  * 元注解：用于描述注解的注解
5  * @Target: 描述注解能够作用的位置
6  * @Retention: 描述注解被保留的阶段
7  * @Documented: 描述注解是否被抽取到api文档中
8  * @Inherited: 描述注解是否被子类继承
9  */
10
11 *
12 */
13
14 @Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})//等效于@Target(value={ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
15 @Retention(RetentionPolicy.RUNTIME)//当前被描述的注解，会保留到class字节码文件中，并被JVM读取到
16 @Documented
17 @Inherited
18 public @interface MyAnno3 {
19 }
20
21
```

- 进入命令行并可看到桌面生成一大堆文档

```
C:\Users\15516535379\Desktop>javadoc worker.java
```

```
正在加载源文件worker.java...
```

```
正在构造 Javadoc 信息...
```

```
标准 Doclet 版本 1.8.0_161
```

```
正在构建所有程序包和类的树...
```

```
正在生成.\Worker.html...
```

```
正在生成.\package-frame.html...
```

```
正在生成.\package-summary.html...
```

```
正在生成.\package-tree.html...
```

```
正在生成.\constant-values.html...
```

```
正在构建所有程序包和类的索引...
```

```
正在生成.\overview-tree.html...
```

```
正在生成.\index-all.html...
```

```
正在生成.\deprecated-list.html...
```

```
正在构建所有类的索引...
```

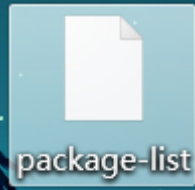
```
正在生成.\allclasses-frame.html...
```

```
正在生成.\allclasses-noframe.html...
```

```
正在生成.\index.html...
```

```
正在生成.\help-doc.html...
```

```
C:\Users\15516535379\Desktop>
```



package-list



Worker.html



deprecated-list.html



package-frame.html



allclasses-frame.html



package-summary.html



allclasses-no-frame.html



package-tree.html



index.html



constant-values.html



help-doc.html



overview-tree.html



stylesheet.css



index-all.html



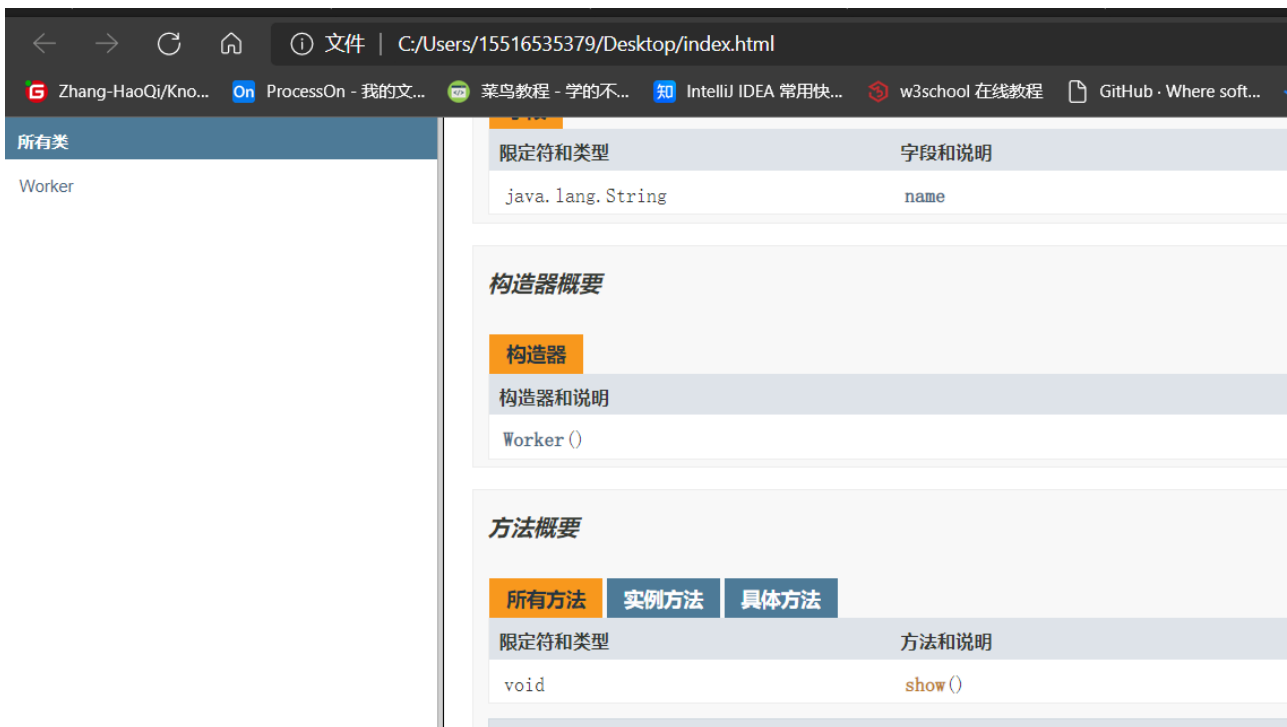
script.js



- 进入index.html可发现注解被保留了



- 点击“方法概要”下的show () 后进行跳转，发现上面注解也被保留了





- 点击“字段概要”下的name后进行跳转，发现上面注解没有被保留

