当我们访问项目的/test路径时，浏览器会向我们的服务器端发送一个请求。我们打开浏览器的开发者工具，打开network选项卡。

在地址栏中敲击回车，浏览器向我们的服务器端发送了一个请求。请求的地址是127.0.0.1:8000/test路径，请求的方法是GET方法，请求的头部headers中，浏览器自动添加了一些参数。

当Symfony接收到这个请求时，会将请求和请求参数封装成一个Request对象。再交由路由系统，根据请求的路径来调用对应的controller的action方法。

对于/test路径会调用index方法，在index方法中我们可以通过一些方法来获取请求的参数，然后根据参数来进行对应的处理。 比如说存取数据库、一些计算等等，最后我们需要返回一个Response对象。

查看Response类的源码，按着command键，鼠标点击Response。Response对象包含了请求返回的内容，返回的状态码，还有返回的headers，这样就完成了一个HTTP请求的处理。

我们来跟踪一遍代码的执行过程，来加深理解。

我们已经安装了PHP的xdebug扩展，我们修改一下xdebug的配置。打开控制台查看一下PHP配置文件的路径，我们修改一下PHP配置文件。

在xdebug配置段下，我们修改xdebug的模式为debug模式，我们修改idekey为PHPSTORM，这样我们在我们的编辑器中就可以直接使用xdebug了，其他三项我们按照这个配置直接写就行了。

```
[xdebug]
zend_extension="xdebug.so"
xdebug.mode = debug
xdebug.idekey = PHPSTORM
xdebug.cli_color = 0
xdebug.start_with_request = yes
xdebug.log_level = 0
```

修改完成之后记得保存，重启我们的项目，在控制台中按control + c停止服务器，再重启服务器。

Symfony项目的入口文件是public目录中的index.php文件，我们在index.php文件中下断点，点击这个按钮监听debug请求。

刷新/test路径，断点已经停到这了，我们一步一步跟着走。我们现在学习的Symfony5.3版本，它的index.php文件和5.2之前的版本不一样，我们按照最新的版本来学习。

回到浏览器，我们搜索symfony 5.3 index.php，我们看下5.3版本的更改，打开这个页面。 Symfony 5.3增加了一个Runtime组件，它是为了让Symfony在不修改代码的情况下可以兼容更多的运行时，比如说FPM、React-PHP或者Swoole等运行时，所以说他和之前版本的index.php有了大的更改。

回到项目我们断在了第5行，在这一行很重要，我们进入第5行的代码，它进入autoload_runtime.php这个文件。
我们下一步，再下一步，在这里SCRIPT_FILENAME，这个变量是我们的index.php。它会再次调用我们的index.php。
我们下一步，在再次调用index.php时，它会返回第7行代码，返回一个闭包函数。下一步，这时我们看$app变量，它是一个闭包函数。

我们继续，在第21行，我们并没有配置APP_RUNTIME这个变量，所以在这一行$runtime变量它是SymfonyRuntime的全类名。 下一步，这一行$runtime会进行实例化。
继续，在这一步他会解析和处理我们的闭包方法。
继续，在第30行会执行我们的闭包方法。
再下一步，这时$app变量就会生成我们的Kernel实例，最重要一步就在这里。 运行时($runtime)实例会根据我们的$app的类型获取对应的Runner对象，然后再调取Runner对象的run方法来处理请求，点击步入来进入代码。 在上一步我们$app参数它是一个Kernel对象，Kernel对象是HttpKernelInterface的实例，所以这一步他会返回HttpKernelRunner对象。 下一步，然后，它会调用Runner对象的run方法。 下一步，在run方法中最重要的一步就是使用Kernel对象来处理请求。

在继续下一步之前，我们打开浏览器搜索使用symfony kernel，我们看一下HttpKernel文档，文档给我们提供了整个Kernel处理请求的流程。

首先Symfony会将我们的请求参数封装成一个Request对象，然后根据Request对象中的参数解析到对应的controller方法。再将controller action方法中的各个参数进行处理，然后调用controller的action方法，如果action方法返回的是Response对象，那就直接可以结束。如果controller方法返回的是view对象，我们最终要转化为Response对象，然后结束这段请求。

回到项目，我们继续跟踪代码，在第37行，我们进入handle方法，点击步入，在handle方法中我们继续往下看，在第199行他会调用HttpKernel对象的handle方法。
我们直接下一步到199行，首先它会获取HttpKernel对象，进入，通过容器来获取HttpKernel对象。 这里我们后面会讲，然后再下一步。

现在就到了HttpKernel的handle方法中，在第79行，重要的一步就是handleRaw，它会处理请求和类型。
我们下一步，再进入handleRaw方法。
我们一步一步跟踪，首先它会将我们的请求进行压栈。下一步，这是一个事件，我们后面课程会讲到Symfony的事件处理。
再下一步下一步，一直到140行，Symfony会根据我们的请求，获取对应的controller方法。 继续下一步，到149行，Symfony会根据我们的请求和controller方法来解析controller方法的参数。
我们继续下一步，在157行，会执行controller的action方法。
继续，在160行，如果$response它不是Response对象的话，它要进行事件处理，最终要返回一个Response对象。
我们回看handleRaw方法，handleRaw方法它的返回值一定是个Response对象，当handleRaw方法最后执行完毕后，我们的请求就处理结束了。
我点击下一步，在Kernel类的201行，进行请求的出栈操作。继续，在Runner对象中，对Response对象进行发送。在第41行，Kernel对象结束这个请求。

到这一步，整个Symfony处理一个请求的流程就结束了。

如果你是Symfony的初学者，看完本节课之后你会一脸懵。没关系，随着我们的深入学习，我们可能会再次的跟踪整个处理请求的流程。你也可以自己追踪这段流程来加深理解。

HttpKernel组件是Symfony的核心。其他的开源项目大多也使用了这套组件，在课程的后期，如果你有兴趣，你可以使用HttpKernel组件来开发出自己的框架，本节课先到这里。

在下一节，我们将讲解如何在action方法中来获取到请求的一些参数。