跟踪symfony3记录

1. 自动加载

自动加载完成后，ClassLoader：：loadClass 还有 swift里也有一个自动加载得函数。

1. Debug

调用Debug::enable();

ExceptionHandler::register();

set\_exception\_handler(array($handler, 'handle'))

ErrorHandler::register(new ErrorHandler(new BufferingLogger()));

$this->loggedErrors;是记录得错误，$this->loggers;是记录得方式

register\_shutdown\_function(\_\_CLASS\_\_.'::handleFatalError');

set\_error\_handler(array($handler, 'handleError'), $handler->thrownErrors | $handler->loggedErrors);

set\_exception\_handler(array($handler, 'handleException'))

public function setExceptionHandler(callable $handler = null)

{

$prev = $this->exceptionHandler;

$this->exceptionHandler = $handler;

return $prev;

}

DebugClassLoader::enable();

总结：注册错误处理函数，开启调试，把自动加载接入调试里。

3、

class AppKernel extends Kernel

abstract class Kernel implements KernelInterface, RebootableInterface, TerminableInterface

interface KernelInterface extends HttpKernelInterface, \Serializable

interface HttpKernelInterface

$this->checkClass($class, $file);

$deprecations = $this->checkAnnotations($refl, $name);

$parentAndOwnInterfaces = $this->getOwnInterfaces($class, $parent);

1. $kernel = new AppKernel('dev', true);

public static function createFromGlobals()

$request = self::createRequestFromFactory($\_GET, $\_POST, array(), $\_COOKIE, $\_FILES, $server);

$response = $kernel->handle($request);

$this->initializeBundles();

public function registerBundles()

实例化bundle经过的类：

new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),

class FrameworkBundle extends Bundle

abstract class Bundle implements BundleInterface

use Symfony\Component\DependencyInjection\ContainerAwareTrait;

trait ContainerAwareTrait

interface BundleInterface extends ContainerAwareInterface

interface ContainerAwareInterface

new Symfony\Bundle\SecurityBundle\SecurityBundle(),

class SecurityBundle extends Bundle

new Symfony\Bundle\TwigBundle\TwigBundle(),

$this->initializeContainer();

$cache = new ConfigCache($cacheDir.'/'.$class.'.php', $this->debug);

{

$checkers = array(new SelfCheckingResourceChecker());

parent::\_\_construct($file, $checkers);

}

class ConfigCache extends ResourceCheckerConfigCache

{

$this->resourceCheckers = $resourceCheckers;

}

class ResourceCheckerConfigCache implements ConfigCacheInterface

interface ConfigCacheInterface

加载meta文件？？？

class ComposerResource implements SelfCheckingResourceInterface, \Serializable

class SelfCheckingResourceChecker implements ResourceCheckerInterface

interface ResourceCheckerInterface

class appDevDebugProjectContainer extends Container

class Container implements ResettableContainerInterface

interface ResettableContainerInterface extends ContainerInterface

interface ContainerInterface extends PsrContainerInterface

$container = $this->buildContainer();

$container = $this->getContainerBuilder();

class ContainerBuilder extends Container implements TaggedContainerInterface

interface TaggedContainerInterface extends ContainerInterface

class ParameterBag implements ParameterBagInterface

（将bundle放到 $this->parameters[$this->normalizeName($name)] = $value;）

$container->addObjectResource($this);

$this->prepareContainer($container);（实例化bundle里面的extension）

$this->resources[(string) $resource] = $resource（E:\wnmp\nginx\html\symfony3\app\AppKernel.php）;

$this->extensions[$extension->getAlias()] = $extension;

到配置文件了：

解析import：

$this->parseImports($content, $path);

$this->registerClasses($definition, $namespace, $service['resource'], $exclude);

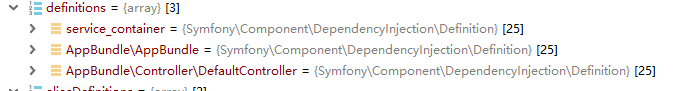
service配置文件的时候，加载了src/controller下面的文件

并加到container的resource里面

现在看看container里面有什么？



分析出servic里面的，放在definition里面



class AddAnnotatedClassesToCachePass implements CompilerPassInterface

public function \_\_construct(Kernel $kernel)

{

$this->kernel = $kernel;

}

接下来到这里了

$container->compile();

过程：

$compiler = $this->getCompiler();

$compiler->compile($this);

###############

foreach ($this->passConfig->getPasses() as $pass) {

$pass->process($container);

}

疑问一：

Container里面属性的作用。和如何初始化（怎么来的）。

疑问二：

Container的extensions是怎么来的？

这段代码太繁杂了，先放着先。看得老夫一口老血。。。。

Kernel文件的如下函数

$this->dumpContainer($cache, $container, $class, $this->getContainerBaseClass());

生成了一系列的自动生成文件，并且替换了kernel里面的container属性。

重头戏：

**return** $this->getHttpKernel()->handle($request, $type, $catch);

Httpkernel.php:

return $this->handleRaw($request, $type);

class GetResponseEvent extends KernelEvent

class KernelEvent extends Event

注意到一件事情：listener，事件监听这个东西。

$this->preProcess($eventName);

以下：开启section？？？

$this->preDispatch($eventName, $event);

执行每个事件监听。

public function dispatch($eventName, Event $event = null)

Symfony\Component\HttpKernel\EventListener\FragmentListener

这里是url和baserul 设置到request里面

每个listenner共用一个stopwatch

$this->**context**->fromRequest($request);

留意一下RouterListener.php下面的context，这里面会有各种route信息（RequestContext实例）

Matcher是router实例也留意一下。

引入了E:\wnmp\nginx\html\symfony3\var\cache\dev\appDevDebugProjectContainerUrlMatcher.php

文件，（问题：什么时候产生的？这个问题非常重要）我改了一下url可以发现文件的修改时间改变了，可以追踪一下。

$matcher = $this->getMatcher();

这个就是返回上述文件产生的类。