我们可以创建xml配置文件，然后用配置文件的方式把服务类添加到容器里。还有另一种通过代码的方式添加：修改Extension类把Storage、Namer、Handler添加到容器里。这里学习使用代码添加服务类和自动将接口下的类批量注册为服务类。

查看TeebbUploadExtension，我们之前通过$container->getDefinition()获取到了一个服务类的定义，然后设置了服务类的构造参数。容器中的服务类在使用时，会转换为一个对象，Definition类定义了服务类转换为对象时，构造参数用哪个？还可以让类对象在初始化完成时执行什么操作。

现在我们手动定义一个服务类的Definition，

```
class TeebbUploadExtension extends Extension
{

    public function load(array $configs, ContainerBuilder $container)
    {
        $processor = new Processor();
        $configuration = new Configuration();
        $config = $processor->processConfiguration($configuration,
$configs);

        $loader = new XmlFileLoader($container, new
FileLocator(__DIR__.'/../Resources/config'));
        $loader->load('form.xml');

        $container->setParameter('teebb.upload_dir',
$config['upload_dir']);

        $fileManagedTypeDefinition = $container-
>getDefinition('teebb.form.file_managed_type');
        $fileManagedTypeDefinition->setArgument(0, $config['upload_dir']);

        $this->registerStorages($container);
    }


    private function registerStorages(ContainerBuilder $container)
    {
//        $storageDefinition = new Definition(FileSystemStorage::class);
//        $container->setDefinition(FileSystemStorage::class,
$storageDefinition);
        //因为FileSystemStorage不需要设置构造参数，上面两行可以简化为：
        $container->register(FileSystemStorage::class,
FileSystemStorage::class);
    }
}
```

同样的添加Namer，最后添加Handler，最后的代码：

```
class TeebbUploadExtension extends Extension
{

    public function load(array $configs, ContainerBuilder $container)
    {
        $processor = new Processor();
        $configuration = new Configuration();
        $config = $processor->processConfiguration($configuration,
$configs);

        $loader = new XmlFileLoader($container, new
FileLocator(__DIR__.'/../Resources/config'));
        $loader->load('form.xml');

        $container->setParameter('teebb.upload_dir',
$config['upload_dir']);

        $fileManagedTypeDefinition = $container-
>getDefinition('teebb.form.file_managed_type');
        $fileManagedTypeDefinition->setArgument(0, $config['upload_dir']);

        $this->registerStorages($container);
        $this->registerNamers($container);

        //最后把Handler添加到容器，Handler有构造参数，所以先定义再添加到容器
        $handlerDefinition = new Definition(UploadHandler::class);
        $handlerDefinition->setArgument(0, new
Reference(FileSystemStorage::class));
        $handlerDefinition->setArgument(1, $config['upload_dir']);
        $handlerDefinition->setArgument(2, new
Reference(PhpNamer::class));

        $container->setDefinition(UploadHandler::class,
$handlerDefinition);
    }


    private function registerStorages(ContainerBuilder $container)
    {
//        $storageDefinition = new Definition(FileSystemStorage::class);
//        $container->setDefinition(FileSystemStorage::class,
$storageDefinition);
        //因为FileSystemStorage不需要设置构造参数，上面两行可以简化为：
        $container->register(FileSystemStorage::class,
FileSystemStorage::class);
    }

    private function registerNamers(ContainerBuilder $container)
    {
        $container->register(PhpNamer::class, PhpNamer::class);
    }

}
```

在表单FileManagedType类中，注入Handler，并修改form.xml配置文件，测试一下：

```
class FileManagedType extends AbstractType
{
    public function __construct(string $uploadDir,
                                EventDispatcherInterface $eventDispatcher,
                                UploadHandler $uploadHandler)
    {
        $this->uploadDir = $uploadDir;
        $this->eventDispatcher = $eventDispatcher;
        $this->uploadHandler = $uploadHandler;
    }


    private function fillFileObject(UploadedFile $file, File $fileObject)
    {
        $originName = $file->getClientOriginalName();

//        $fileName = pathinfo(htmlspecialchars($originName),
PATHINFO_FILENAME) . '-' . $file->getFilename() . '.' . $file-
>getClientOriginalExtension();

        $mimeType = $file->getMimeType();
        $filesize = $file->getSize();

        //这里需要修改，文件名是在Handler类里自动生成的，我们需要在Handler类里添加一
个方法，这里的代码不够优雅，我们后面会重构出更优雅的代码。
        $this->uploadHandler->upload($file);
        $fileName = $this->uploadHandler->getFileName();
//        $file->move($this->uploadDir, $fileName);

        $fileObject->setName($fileName);
        $fileObject->setMimeType($mimeType);
        $fileObject->setOriginName($originName);
        $fileObject->setSize($filesize);
    }
}

//修改Handler类添加fileName属性
class UploadHandler
{
    /**
     * @var NamerInterface
     */
    private $namer;
    /**
     * @var StorageInterface
     */
    private $storage;

    /**
```

```
         * @var string
         */
        private $distDir;

        /**
         * 这里添加一个属性用于保存文件名，很不优雅，后面我们会重构出优雅的代码
         * @var string
         */
        private $fileName;

        /**
         * @param StorageInterface $storage 用于操作文件
         * @param string $distDir 文件的目标目录
         * @param NamerInterface $namer 文件的命名器
         */
        public function __construct(StorageInterface $storage,
                                    string $distDir,
                                    NamerInterface $namer)
        {
            $this->namer = $namer;
            $this->storage = $storage;
            $this->distDir = $distDir;
        }

        public function upload(UploadedFile $uploadedFile)
        {
            $this->fileName = $this->namer->rename($uploadedFile);

            $this->storage->upload($uploadedFile, $this->distDir, $this-
    >fileName);
        }

        public function getFileName(): string
        {
            return $this->fileName;
        }
    }
```

虽然不够优雅，但是能用了。

将来文件的命名可能有很多种方式，可能会有很多个命名器类，我们需要把所有的Namer类，添加到容器中。 下节课，我们再多定义一个命名器类。