Проектирование архитектур программного обеспечения

лекция 6

Зозуля А.В.

КСПТ

2016 г.

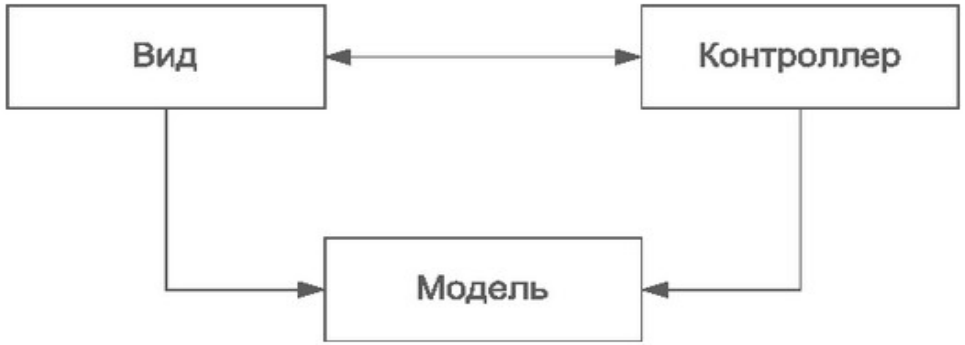
ранее..

Объектно-реляционные типовые решения, предназначенные для моделирования структуры

Типовые решения представления данных в Web

- Модель-представление-контроллер (Model View Controller)
- Контроллер страниц (Page Controller)
- Контроллер запросов (Front Controller)
- Представление по шаблону (Template View)
- Представление с преобразованием (Transform View)
- Двухэтапное представление (Two Step View)
- Контроллер приложения (Application Controller)
- Виджет (Widget), Фильтр (Filter), Модуль (Module)

Модель — представление - контроллер (Model View Controller)

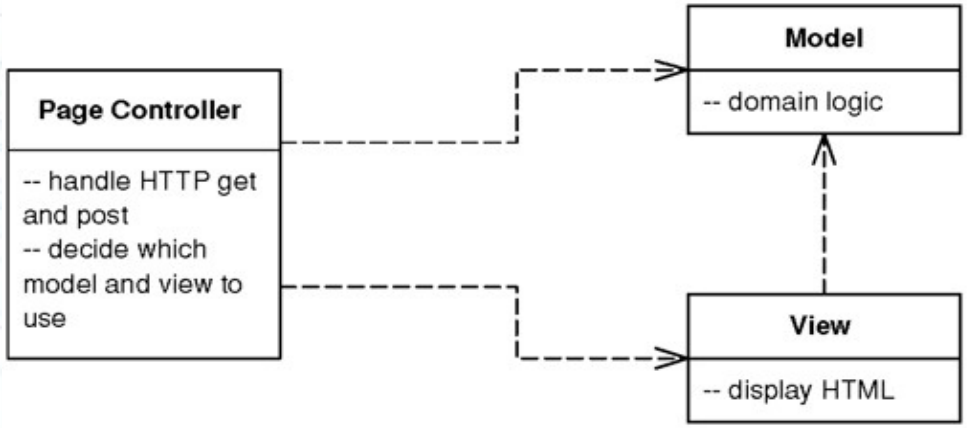


- Распределяет обработку взаимодействия с пользовательским интерфейсом между тремя участниками: модель, представление, контроллер
- Модель: Модель предметной области, сценарий транзакции,...
- Представление: пользовательский интерфейс
- Контроллер: обработка изменений информации, выполняет операции над моделью, управляет обновлением представления

Модель — представление - контроллер (Model View Controller)

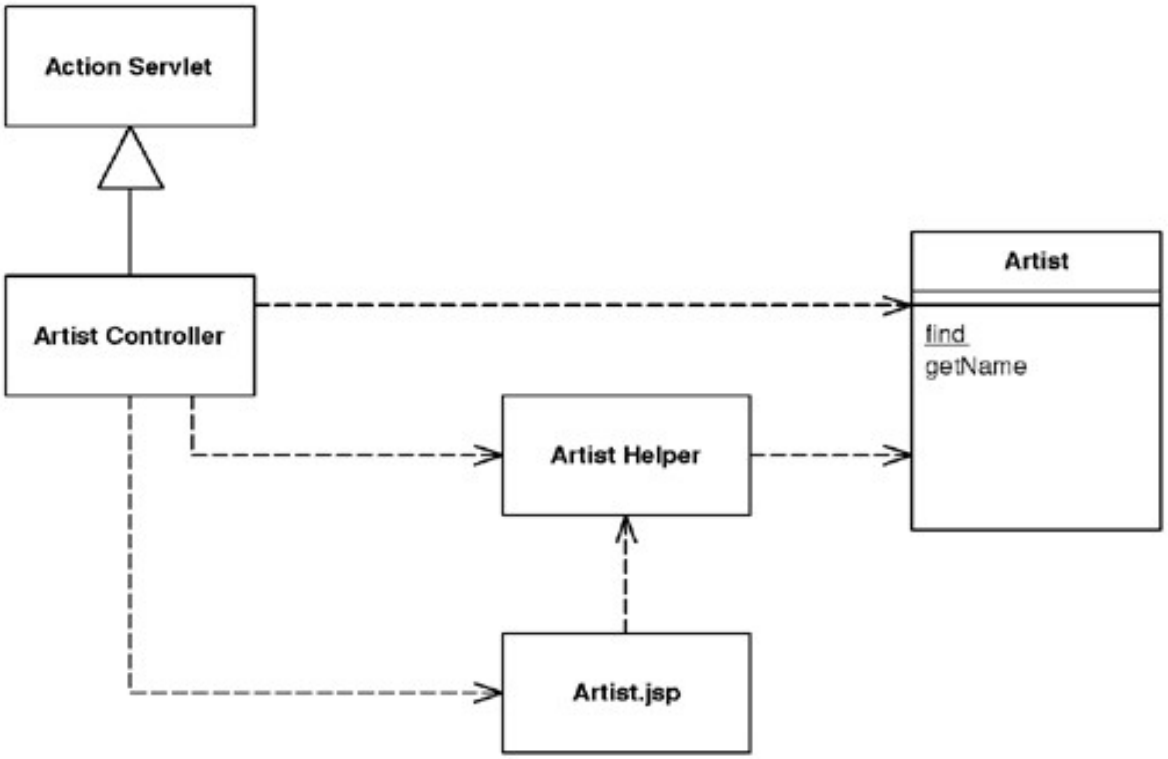
- Отделение модели от представления
  - Разные разработчики
  - Разные библиотеки
  - Несколько представлений
  - Возможность тестирования модели
  - Представление зависит от модели
- Отделение контроллера от представления
  - Особенно актуально для Web
  - Множество данных форм
  - AJAX-запросы

Контроллер страниц (Page Controller)



- Объект, обрабатывающий запрос к конкретной Web-странице или выполнение конкретного действия на Web-сайте
- Контроллер нужен для каждого действия
- Извлечение данных POST и GET
- Создание объектов модели, вызов методов обработки
- Определение представления для действия и передача данных модели

Контроллер страниц (Page Controller)



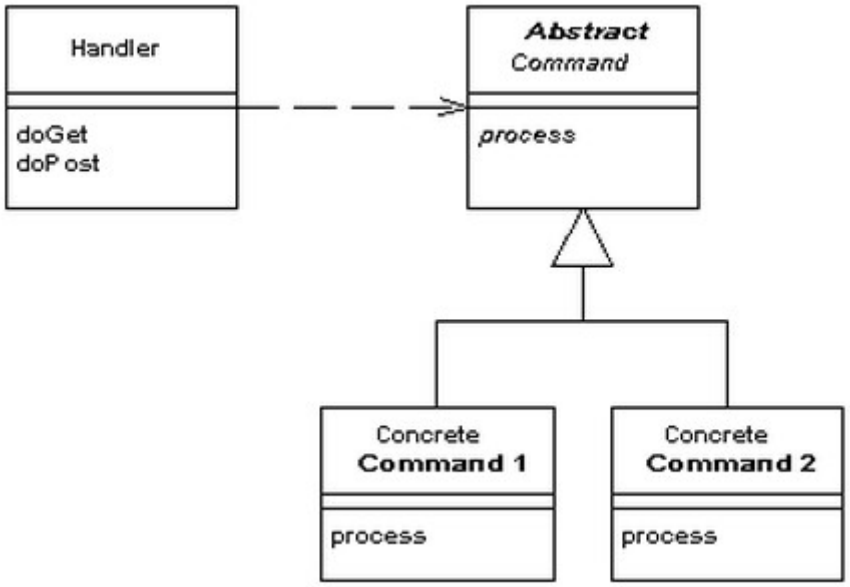
- Контроллер реализует метод обработки запроса
- Вспомогательный объект осуществляет обработку всей логики контроллера
- Страница сервера играет роль представления по шаблону

Контроллер страниц (Page Controller)

```
<!-- Tomcat web.xml →
<servlet>
      <servlet-name>artist</servlet-name>
      <servlet-class>actionController.ArtistController</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>artist</servlet-name>
      <url-pattern>/artist</url-pattern>
</servlet-mapping>
```

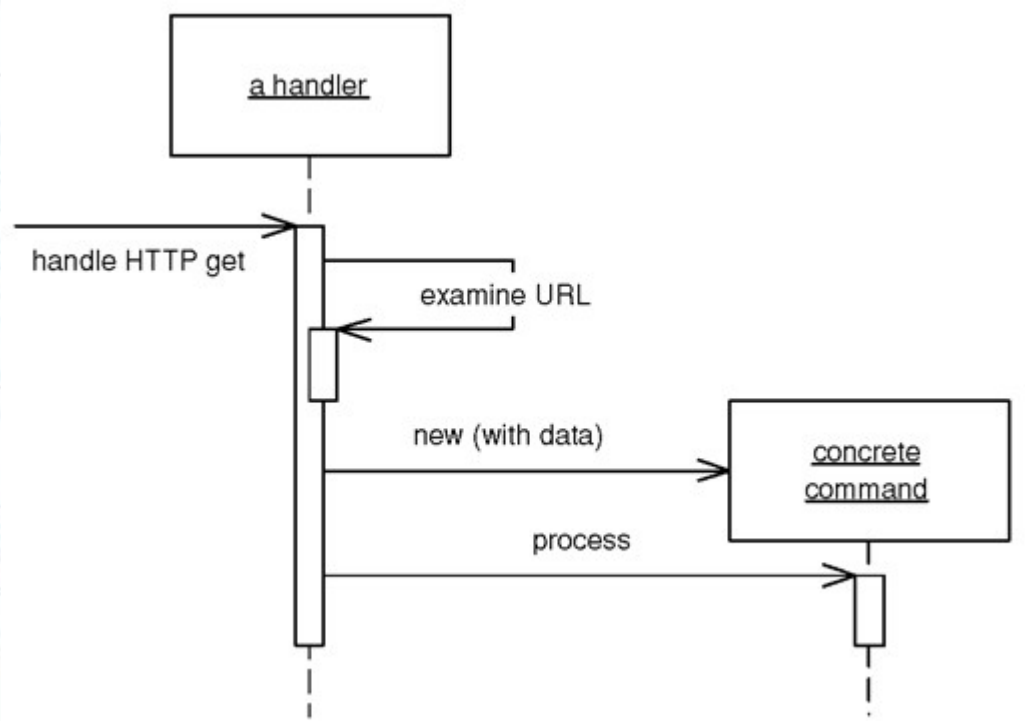
```
class ArtistController extends ActionServlet
            throws IOException, ServletException {

      public void doGet(HttpServletRequest request, HttpServletResponse response) {
            Artist artist = Artist.findNamed(request. getParameter ("name"));
            if (artist == null)
                  forward("/MissingArtistError.jsp", request, response);
            else {
                  request.setAttribute("helper", new ArtistHelper(artist));
                  if (artist instanceof ClassicalArtist)
                        forward("/classicalartist.jsp", request, response);
                  else
                        forward("/artist.jsp", request, response);
            }
      }
}
```

Контроллер запросов (Front Controller)



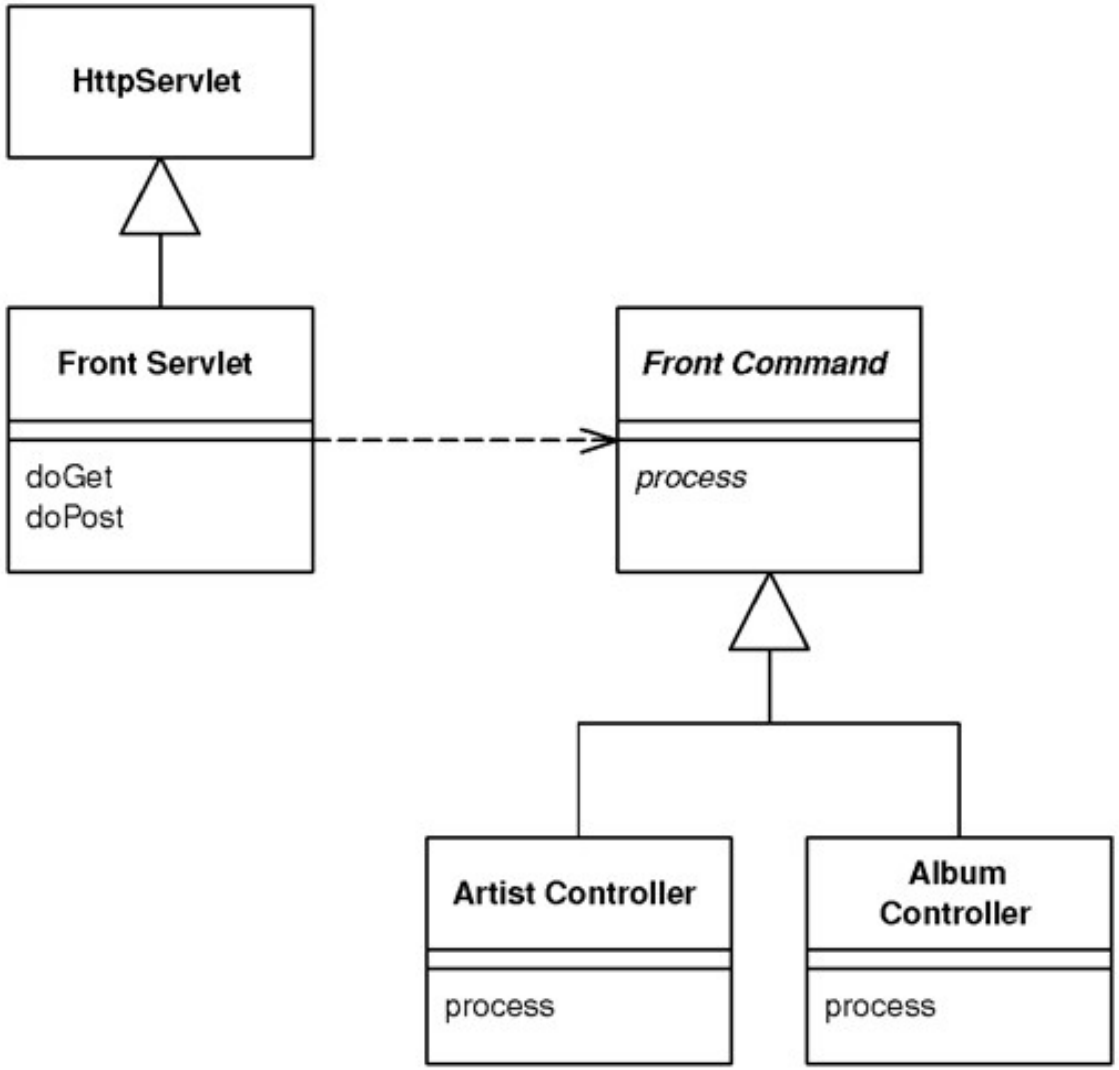
- Контроллер, который обрабатывает все запросы к Web-сайту
- Контроллер = веб-обработчик + иерархия команд
- Считывает URL, GET и POST-данные, делегирует их команде
- Статический и динамический выбор класса команды
- Команды выбирают представление

Контроллер запросов (Front Controller)



- Более сложный аналог Контроллера страниц
- Единая «точка входа» веб-сервера
- Возможность контроля доступа к ресурсам: проверка безопасности, поддержка интернационализации, маршрутизация запросов в зависимости от типа контента

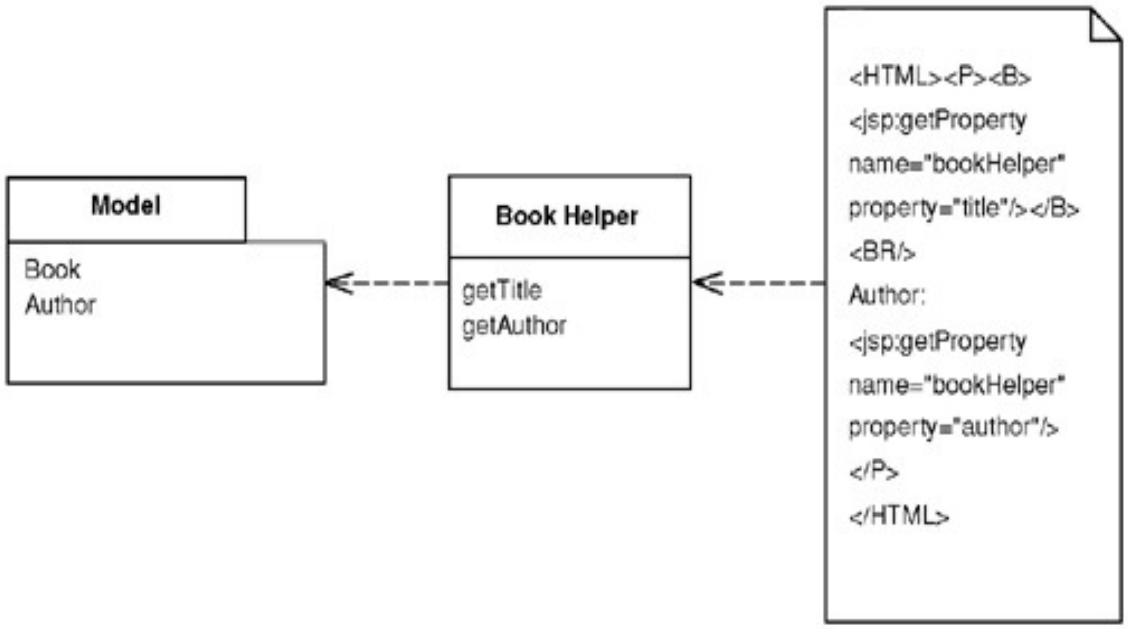
Контроллер запросов (Front Controller)



Контроллер запросов (Front Controller)

```
class FrontServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws IOException, ServletException {
		FrontCommand command = getCommand(request);
		command.init(getServletContext(), request, response);
		command.process() ;
	}
	private FrontCommand getCommand(HttpServletRequest request) {
		try {
			return (FrontCommand) getCommandClass(request).newInstance();
		} catch (Exception e) {
			throw new ApplicationException (e);
		}
	}
	private Class getCommandClass(HttpServletRequest request) {
		Class result;
		final String commandClassName = "frontController." + (String)
request.getParameter("command") + "Command";
		try {
			result = Class.forName(commandClassName);
		} catch (ClassNotFoundException e) {
			result = UnknownCommand.class;
		}
		return result;
	}
}
```

Представление по шаблону (Template View)



- Роль представления в MVC
- Преобразует результаты выполнения запроса в формат HTML путем внедрения маркеров в HTML-страницу
- Маркеры могут быть HTML-подобными или текстовыми
- Шаблон — результат творчества верстальщика

Представление по шаблону (Template View)

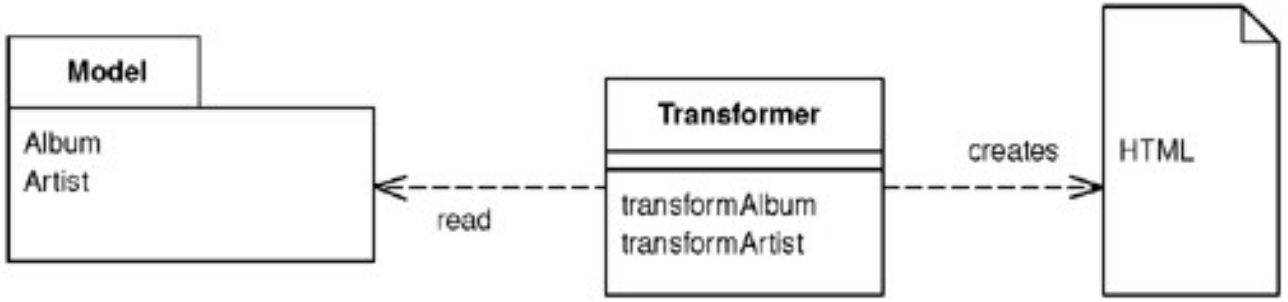
- Вспомогательный объект — способ ухода от скриптлетов
- Условное отображение и итерация
    - Перенос в вспомогательный объект
    - Вместе с минимальным HTML-кодом
- Возможность кеширования
- Исключения должны обрабатываться во вспомогательном классе
- «-»
    - Опасность переполнения логикой
    - Невозможность тестирования

Представление по шаблону (Template View)

```
<jsp:useBean id="helper" type="actionController.ArtistHelper" scope="request"/>
<h1><jsp:getProperty name="helper" property="name"/></h1>
<div><%=helper.getAlbumList()%></div>
```

```
class ArtistHelper {
	private Artist artist;

	public ArtistHelper(Artist artist) { this.artist = artist; }

	public String getName() { return artist.getName(); }

	public List getAlbums() { return artist.getAlbums();}

	public String getAlbumList() {
		StringBuffer result = new StringBuffer();
		result.append("<UL>");
		for (Iterator it = getAlbums().iterator(); it.hasNext();) {
			Album album = (Album)it.next();
			result.append("<LI>");
			result.append(album.getTitle());
			result.append("</LI>") ;
		}
		result.append("</UL>");
		return result.toString();
	}
}
```

Представление с преобразованием (Transform View)



- Представление, которое поочередно обрабатывает элементы данных домена и преобразует их в код HTML
- XSLT (eXtensible Stylesheet Language Transformations) - наиболее популярный язык для написания преобразований
- Данные домена извлекаются в формате XML, далее XSLT преобразует их в HTML
- «+»: простота тестирования, отсутствие логики в представлении
- «-»: сложность функционального языка XSLT, невозможность привлечения верстальщика HTML

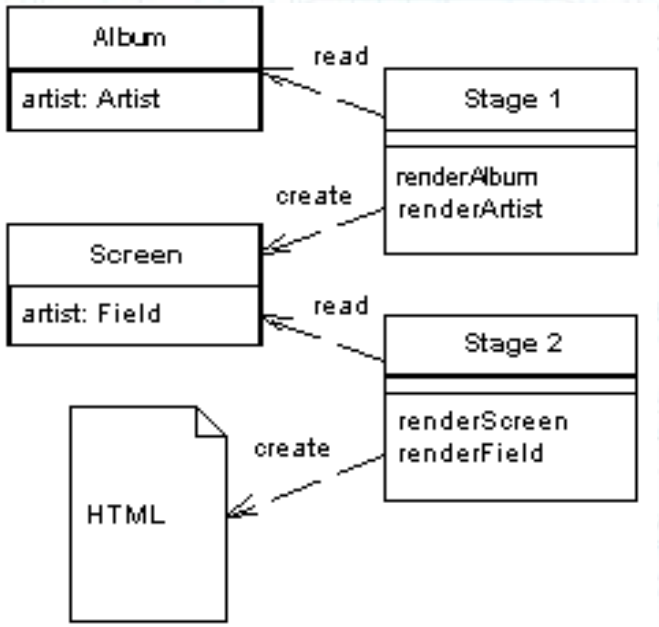
Представление с преобразованием (Transform View)

```
class AlbumCommand {
      public void process () {
            try {
                  Album album = Album.findNamed(request.getParameter("name"));
                  PrintWriter out = response.getWriter();
                  XsltProcessor processor = new SingleStepXsltProcessor("album.xsl");
                  out.print(processor.getTransformation(album.toXmlDocument())) ;
            } catch (Exception e) { throw new ApplicationException(e) ; }
      }
}
```

```
<album>
      <title>Stormcock</title>
      <artist>Roy Harper</artist>
</album>
```

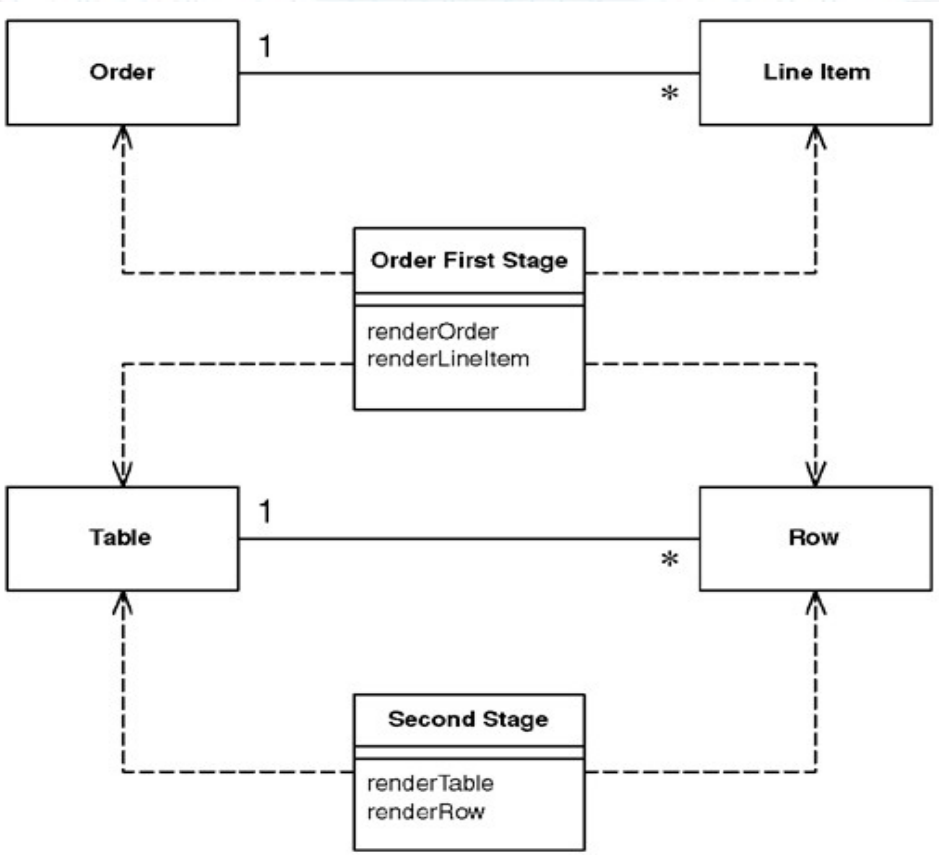
```
<xsl:template match="album">
      <HTML><BODY bgcolor="white"><xsl:apply-templates/></BODY></HTML>
</xsl:template>
<xsl:template  match="album/title">
      <hl><xsl:apply-templates/></hl>
</xsl:template>
<xsl:template match="artist">
      <P><B>Artist: </B><xsl: apply-templates/></P>
</xsl:template>
```

Двухэтапное представление (Two Step View)



- Выполняет визуализацию данных домена в два этапа: вначале формирует некое подобие логической страницы, после чего преобразует логическую страницу в формат HTML
- Позволяет быстро кардинально изменять вид множества страниц благодаря промежуточному логическому представлению

Двухэтапное представление (Two Step View)



- «-»: сохраняет недостатки Представления с преобразованием, вносит в дополнительную сложность в разработку преобразователя

Представление по шаблону на стороне браузера

doT.js

Шаблон

```
{{? it.name }}
<div>Oh, I love your name,
{{=it.name}}!</div>
{{?? it.age === 0}}
<div>Guess nobody named you yet!
</div>
{{??}}
You are {{=it.age}} and still don't have
a name?
{{?}}
```

Результат компиляции шаблона

```
function anonymous(it) { var
out='';if(it.name){out+='<div>Oh, I love
your name, '+(it.name)+'!</div>';}else
if(it.age === 0){out+='<div>Guess
nobody named you yet!
</div>';}else{out+='You are '+(it.age)
+' and still don\'t have a name?';}return
out; }
```

Данные

```
{"name":"Jake","age":31}
```

Результат

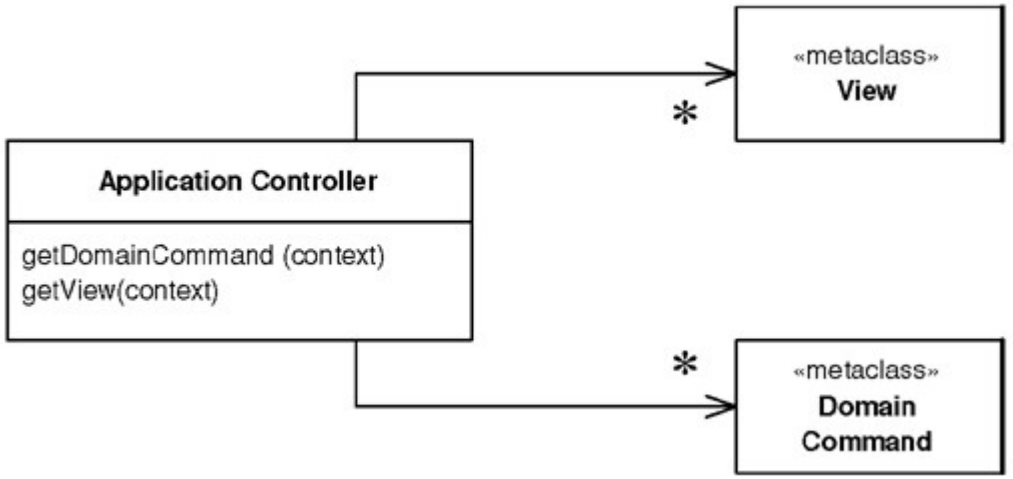
```
<div>Oh, I love your name, Jake!</div>
```

Использование

```
<script type="text/javascript" src="doT.js"></script>

var tempFn = doT.template("<h1>Here is a sample template {{=it.foo}}</h1>");

var resultText = tempFn({foo: 'with doT'});
```

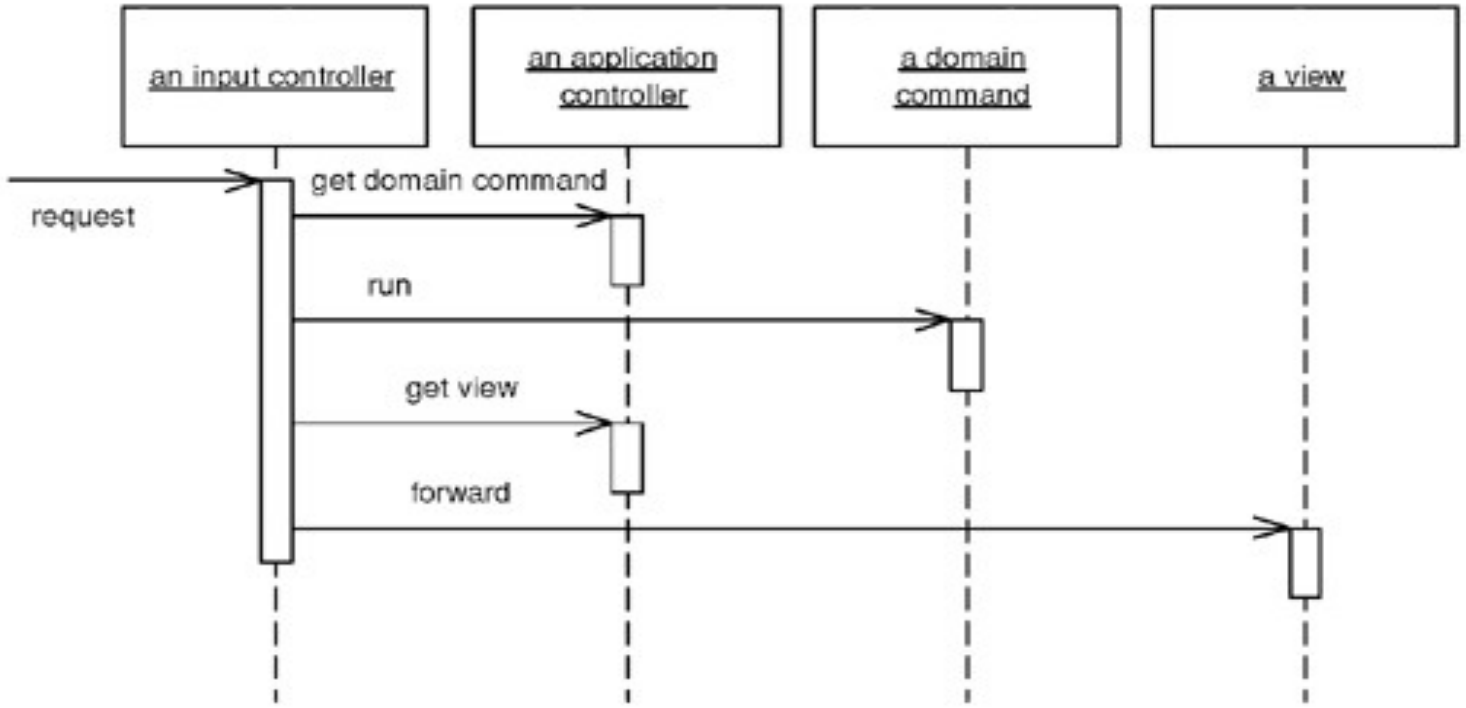
Контроллер приложения (Application Controller)

- Точка централизованного управления порядком отображения интерфейсных экранов и потоком функций приложения
- При усложнении Web-приложения логика из контроллеров страниц может быть вынесена
- Контроллер приложения является поставщиком команд (логики домена) и представлений для входных контроллеров



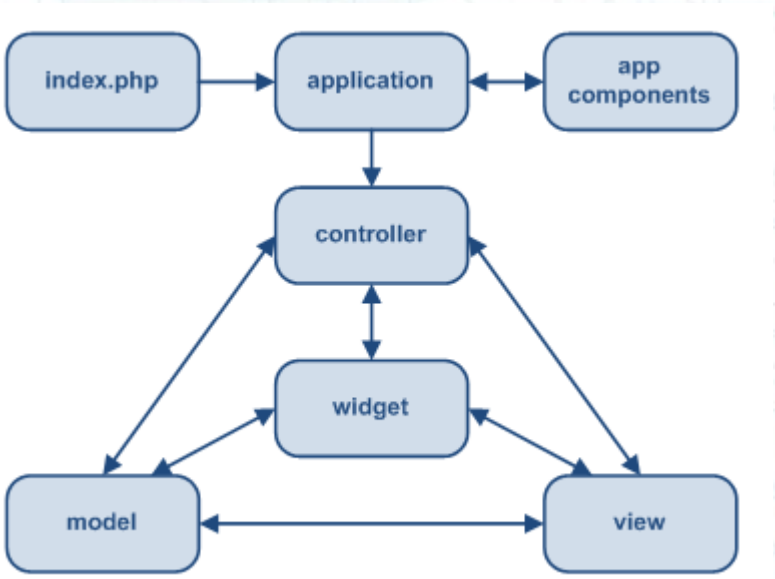
Контроллер приложения (Application Controller)

- Командами могут быть как команды Контроллера приложения, так и ссылки на Сценарий транзакции или методы объектов Домена предметной области



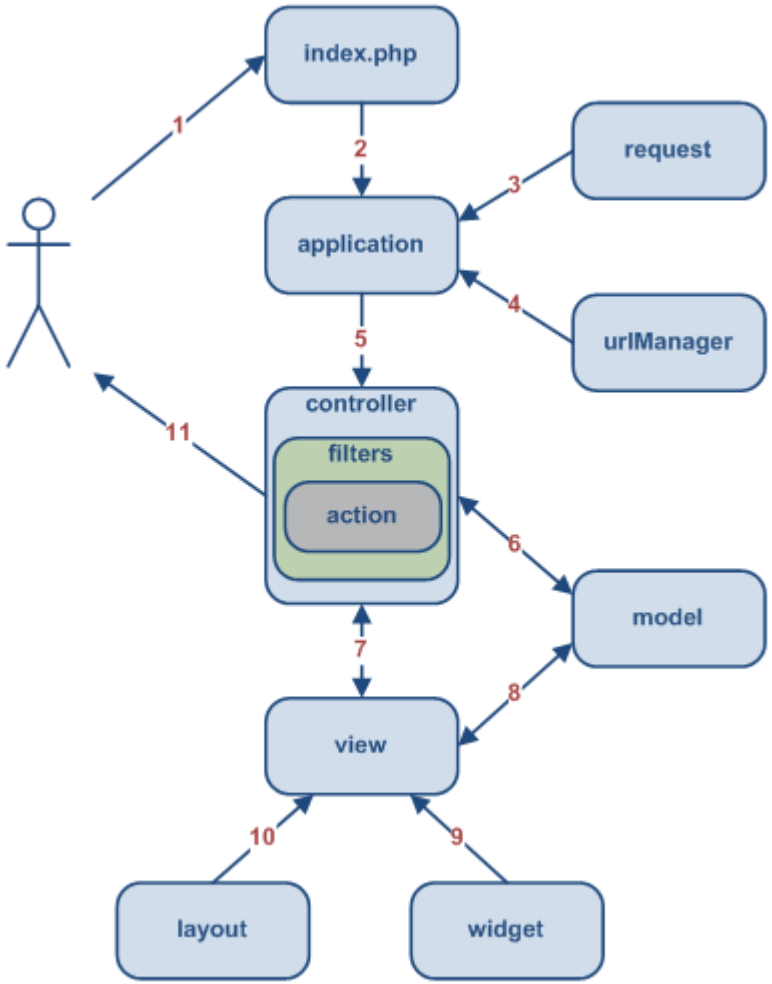
MVC фреймворк Yii

Статическая структура



MVC фреймворк Yii

Обработка запроса



1. http://www.example.com/index.php?r=post/show&id=1

2. Скрипт загрузки index.php создает экземпляр приложения CWebApplication и запускает его.

3. Приложение получает информацию из запроса при помощи объекта CHttpRequest.

4. Приложение определяет необходимый контроллер PostController и действие с помощью CUrlManager.

5. Приложение создает контроллер. Контроллер определяет, что действие show относится к методу actionShow в классе контроллера. Затем создаются и выполняются фильтры, связанные с этим действием. Действие выполняется, если это разрешено фильтрами.

6. Действие считывает модель Post с ID 1 из базы данных.

7. Действие назначает представлению show данные модели Post.

8. Представление считывает и отображает атрибуты модели Post.

9. Представление инициирует выполнение виджетов.

10. Результат рендеринга вида встроен в макет.

11. Действие завершает рендеринг и выводит результат пользователю.

Filter (Фильтр)

- Фильтр — это часть кода, которая может выполняться до или после выполнения действия контроллера
- Действие может иметь множество фильтров
- Фильтры запускаются в том порядке, в котором они указаны в списке фильтров, при этом фильтр может предотвратить выполнение действия и следующих за ним фильтров
- Фильтр может быть определён как метод класса контроллера
- **Примеры**:
  - фильтр контроля доступа может проверять, аутентифицирован ли пользователь перед тем, как будет выполнено запрошенное действие
  - Фильтр, контролирующий производительность, может быть использован для определения времени, затраченного на выполнение действия

Filter (Фильтр)

```
class PerformanceFilter extends CFilter {
    protected function preFilter($filterChain) {
        // код, выполняемый до выполнения действия
        return true; // false — для случая, когда действие не должно быть выполнено
    }

    protected function postFilter($filterChain) {
        // код, выполняемый после выполнения действия
    }
}
```

```
class PostController extends CController
{
    …
    // Для того чтобы применить фильтр к действию, необходимо переопределить метод
    // CController::filters(), возвращающий массив конфигураций фильтров
    public function filters() {
        return array(
            'postOnly + edit, create',
            array(
                // Используя операторы '+' и '-' можно указать, к каким действиям должен
                // и не  должен быть применён фильтр
                'application.filters.PerformanceFilter - edit, create',
                'unit'=>'second',  // начальные значения свойств фильтра
            ),
        );
    }
}
```

Widget (Виджет)

- Автономный компонент, который может генерировать представление, основанное на модели данных
- Микро-контроллер, который вкладывается в представления, управляемые контроллерами
- По сравнению с контроллером, виджет не имеет ни действий, ни фильтров

ActiveForm
ListView
Pager
Breadcrumbs
Captcha
FilterWidget
Menu
TabView
TreeView

Widget (Виджет)

```php
<?php $form = $this->beginWidget('CActiveForm', array(
    'id'=>'user-form',
    'enableAjaxValidation'=>true,
    'enableClientValidation'=>true,
    'focus'=>array($model,'firstName'),
)); ?>
```

```php
class CActiveForm extends CWidget {
        public function init() {
                echo CHtml::beginForm($this->action, $this->method, $this->htmlOptions);
        }
        public function run() {
                echo CHtml::endForm();
                $options['attributes']=array_values($this->attributes);
        }
        public function error($model,$attribute,$htmlOptions=array(),
                        $enableAjaxValidation=true,$enableClientValidation=true) {
                foreach($model->getValidators($attributeName) as $validator) {
                        if(($js = $validator->clientValidateAttribute($model,$attributeName)) != '')
                                $validators[] = $js;
                }
        }
        public function emailField($model,$attribute,$htmlOptions=array()) {
                return CHtml::activeEmailField($model,$attribute,$htmlOptions);
        }
        public function dateField($model,$attribute,$htmlOptions=array()) {
                return CHtml::activeDateField($model,$attribute,$htmlOptions);
        }
        public function hiddenField($model,$attribute,$htmlOptions=array()) {
                return Chtml::activeHiddenField($model,$attribute,$htmlOptions);
        }
}
```

Module (Модуль)

- Самодостаточная программная единица, состоящая из моделей, представлений, контроллеров и иных компонентов
- Модуль в отличие от Приложения не может использоваться сам по себе
- Пользователи могут обращаться к контроллерам внутри модуля
- Декомпозиция Приложения на несколько подсистем, разрабатываемых и поддерживаемых по отдельности.
- Повторное использование кода
- Организован как директория, имя которой выступает в качестве уникального идентификатора модуля
- Модули могут быть вложенными друг в друга сколько угодно раз

Module (Модуль)

```
forum/
   ForumModule.php          файл класса модуля
   components/              содержит пользовательские компоненты
      views/                содержит файлы представлений для виджетов
   controllers/             содержит файлы классов контроллеров
      DefaultController.php файл класса контроллера по умолчанию
   extensions/              содержит сторонние расширения
   models/                  содержит файлы классов моделей
   views/                   содержит файлы представлений контроллера и макетов
      layouts/              содержит файлы макетов
         index.php          файл представления 'index'
```

```
// Конфигурация модуля
return array(
    ...
    'modules'=>array(
        'forum'=>array(
            'postPerPage'=>20,
        ),
    ),
    ...
);
```

```
// К экземпляру модуля можно обращаться посредством свойства module
// активного в настоящий момент контроллера
$postPerPage=Yii::app()->controller->module->postPerPage;
```

далее..

Архитектура нагруженного веб-приложения