Caching

In computer science, a cache is a collection of data **duplicating original values...**, where the **original data is expensive** to fetch... or to compute, **compared to the cost of reading the cache.**

In other words, a cache is a **temporary storage** area where frequently accessed data can be stored **for rapid access.**

Was cachen?

- **Readonly** Daten!
- **Häufig** benutzte Daten (Was heißt häufig?)
- **Teure** Daten (Was heißt teuer?)

Time to Live?

- Wie **hoch** ist die **Aktualisierungsrate** der **original Daten**?
- Wie **teuer** sind die **original Daten**?

hohe *Aktualisierungsrate →* ***niedrige*** *TTL*
teure *original Daten →* ***hohe*** *TTL*

TTL muss immer niedriger als die Aktualisierungsrate sein!

Basics: RD::$Self->CacheTtl()

```
class RDM_Element_Oldschool extends RDM {
    const TTL = 3600; // Sekunden!!!

    public function Start() {
        // Do easy stuff here...
        $this->Assign('Foo', 'Bar');
        $this->Assign('Fubar', $this->FuckedUpBeyondAllRepair());
    }

    protected function FuckedUpBeyondAllRepair() {
        if (!$Data = $this->CacheTtl('RDM_Element_Oldschool')) {
            // Do heavy stuff here...
            $this->CacheTtl('RDM_Element_Oldschool', $Data, self::TTL);
        }

        return $Data;
    }
}
```

APE = APE Patterns of Experience

1. Simple Element Caching
2. Key Binded Element Caching

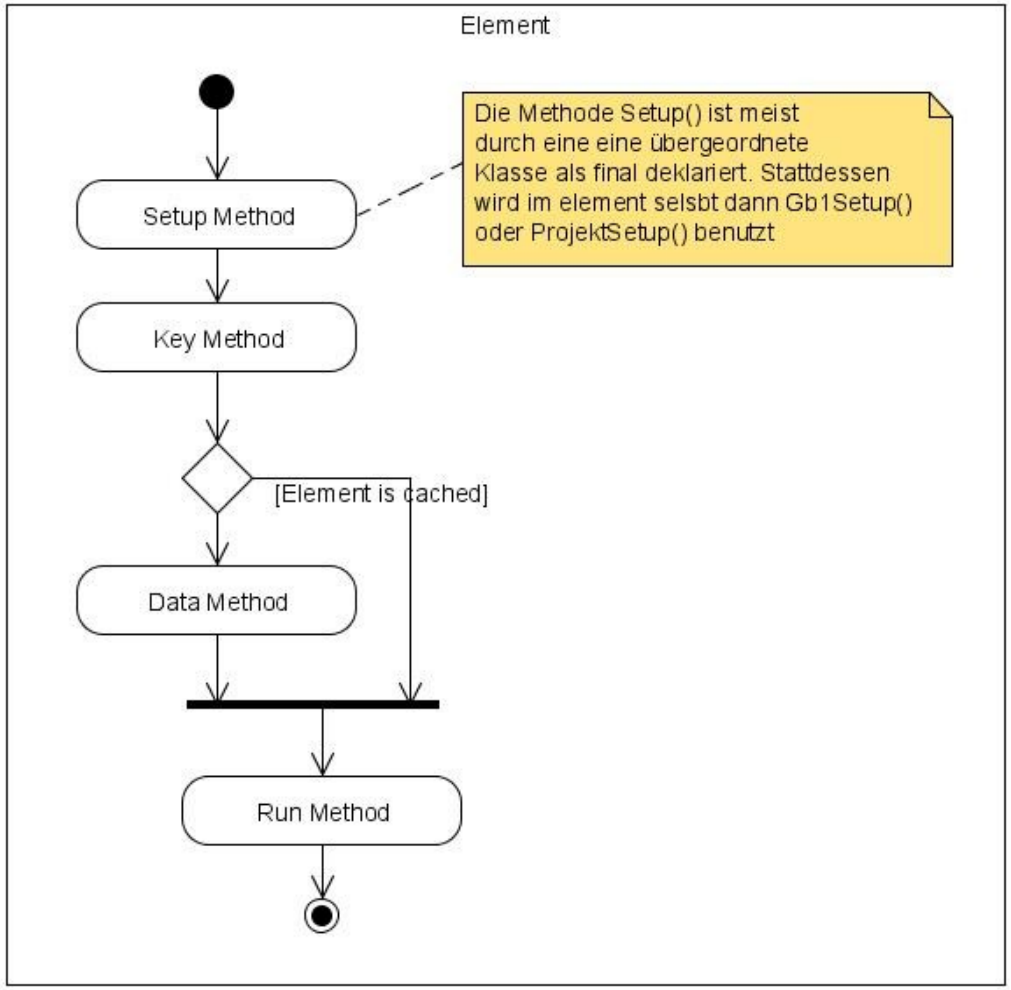
Simple Element Caching

```
class RDM_Element_Bla_Blub extends Nifty_Core {
    public $CacheTTL = 3600 // Sekunden!!!

    public function Run($Assigns) {
        // Do easy stuff here...
        $Assigns['Foo'] = 'Bar';
        return $Assigns;
    }

    public function Data($Assigns) {
        // Do heavy stuff here...
        $Assigns['Fubar'] = $this->FuckedUpBeyondAllRepair();
        return $Assigns;
    }

    protected function FuckedUpBeyondAllRepair() { //… }
}
```

Key Binded Elememnt Caching

```
class RDM_Element_Bla_Blub extends Nifty_Core {
    public $CacheTTL = 3600 // Sekunden!!!

    public function NiftySetup($Config) {
        $this->act = $this->GetGet('act', 1);
    }

    public function Run($Assigns) { … }

    public function Data($Assigns) {
        // Do heavy stuff here...
        $Assigns['Fubar'] = $this->FuckedUpBeyondAllRepair($this->act);
        return $Assigns;
    }

    public function Key() {
        return parent::Key().'_'.$this->act;
    }
}
```

RapiDev – Behind the scenes



RapiDev – Behind the scenes

wx_rapidev/trunk/classes/RDM/Webix/Elemet.php:

```
abstract class RDM_Webix_Element extends RDM {
    public function Key() {
        $Key  = 'Webix_Element_';
        $Key .= $this->IRMCC['Container_ID'].'_';
        $Key .= $this->ElementName.'_';
        return $Key;
    }
}
```

wx_rapidev/branches/pre_v_08/classes/AF/RD/Element/InterRed.php:

```
abstract class AF_RD_Element_InterRed extends AF_RD_Element {
    public function Key() {
        $Key  = 'AF/RD/Element/InterRed/'.$this->IRMCC['Container_ID'];
        $Key .= '_'.$this->ElementName.'_';
        return $Key;
    }
}
```

Nicht raten, nachmessen!

Die Kenngröße bei uns ist ein HTTP-Request!

Die Klassiker in der Performance-Analyse:

- Execution **Count**
- Execution **Time** (Delta-T)

Execution Count

Messpunkte (Logging im code):

```
protected function FuckedUpBeyondAllRepair() {
    // …
    RDD::Log('I do FUBAR now', TRACE, 471100);
    //here comes the heavy stuff...
}
```

Auswertung (tail rd.log | egrep "TRACE-471100" | wc -l):

Einfach zählen wie oft der Logeintrag vorkommt.

ACTUNG: Vor jeder Messung rd.log „leeren“, da man sonst nicht die exakte Execution Count pro Request hat!

Zum nachdenken: Wird eine Methode/Funktion zwangsläufig immer gleich oft ausgeführt?

Nifty Execution Count

Messpunkte (Logging im code):

```
protected function FuckedUpBeyondAllRepair() {
    static $ExecutionCounter = 0;
    // …

    $ExecutionCounter++;
    RDD::Log('I do FUBAR now. Exec counter: '.$ExecutionCounter, TR...
    //here comes the heavy stuff...
}
```

Auswertung (tail -f rd.log | egrep "TRACE-471100"):

```
    TRACE-471100 ... I do FUBAR now. Exec counter: 1
    TRACE-471100 ... I do FUBAR now. Exec counter: 2
    TRACE-471100 ... I do FUBAR now. Exec counter: 3
```

Delta-T nicht Delta-Force

Messpunkte (Logging im code):

```
    protected function FuckedUpBeyondAllRepair() {
        // …
        RDD::Log('Starting FUBAR...', TRACE, 471100);
        //here comes the heavy stuff...
        RDD::Log('FUBAR done.', TRACE, 471100);
        // …
    }
```

Auswertung (tail -f rd.log | egrep "TRACE-471100"):

```
    TRACE-471100 [1.72242617607] ... Starting FUBAR...
    TRACE-471100 [4.0895409584] ... FUBAR done.
```

Wie teuer? $\Delta t = t_{Ende} - t_{Start} \approx 4{,}0895\,s - 1{,}7224\,s \approx \underline{2{,}3671\,s}$

Something generell…

- **Variablen** die **Bool** sind sollten das auch im Namen aussagen
  - **$IsEnabled**, **$WasDeleted**, **$HasSomething** etc.
  - **$Paging**, **$i**, **$Karlheinz**, etc.
- **Methoden** die **Bool** zurückgeben sollten das auch im Namen aussagen
  - **$this->IsFubarEnabled()**, **$this->HasFubarSomeErrors()**, etc.
  - **$this->CcdbStatus()**, **$this->GetConnection()**, etc.

Thanks

„Das einzige was sich schneller als Licht bewegen kann sind schlechte Nachrichten.“

(Douglas Adams)