TNG TECHNOLOGY CONSULTING

TYPEHINTS IN PHP5

AN INTRODUCTION TO TYPEHINTS, ERROR HANDLERS, CLASSLOADING AND SOURCE TO SOURCE TRANSFORMATION

Created by Mathias Burger, @MathiasBurger

TYPE HINTS IN PHP7

- Make intentions transparent
- Code can be better analyzed
- Future performance improvements

USAGE OF TYPE HINTS

In functions for

- arguments
- the return value

Type	PHP Version
Name of class/interface	5.0.0
self	5.0.0
array	5.1.0
callable	5.4.0
bool	7.0.0
float	7.0.0
int	7.0.0
string	7.0.0
void (return type)	7.1
iterable	7.1
?<type> (nullable type)	7.1
numeric (float|int)	proposed as an RFC

Scalar type hints

```php
<?php

declare(strict_types=1);

function fun(string $s, int $i, float $f, bool $b)
{
}

function timesTwoPointFive(float $f) : float
{
    return $f*2.5;
}
```

Nullability

```php
<?php

function oldstyleNullable(string $s=null)
{
}

function newstyleNullable(?string $s) : ?string
{
    return $s;
}
```

Void and iterable

```php
<?php

function returnNothing() : void
{
}

function generator() : iterable
{
    yield 1;
    yield 2;
    yield 3;
}
```

IMPLEMENTING SCALAR ARGUMENT TYPES IN PHP5

IDEA

Argument types exist for classes and interfaces.

Missing types → recoverable error with message:

> *Argument «n» passed to «namespace»\«class»::«function»() must be an instance of «namespace»\«scalar-type», «given-type» given, called in «file» on line «line-number» and defined*

Handle with *set_error_handler()*

SCALAR-TYPE !== GIVEN-TYPE FOR SAME TYPE

scalar-type	given-type
string	string
bool	boolean
int	integer
float	double

SET_ERROR_HANDLER()

- callback signature:

```
function(
        $errorLevel,
        $errorMessage,
        $file,
        $line
    ) { …; return $suppressPhpErrorHandling; }
```

- generate custom message with *trigger_error()*

SET_ERROR_HANDLER() CTD.

- can be set for specific error types:
  E_RECOVERABLE, …
- cannot handle
  E_ERROR, E_PARSE, E_CORE_ERROR,
  E_CORE_WARNING, E_COMPILE_ERROR,
  E_COMPILE_WARNING, most E_STRICT
- return *false* if php error handling is desired

GETTING THE ARGUMENT VALUE

- Look at call stack: *debug_backtrace()*
- Algorithm:
  - Iterate over entries
  - Match *$file*, *$line* from error handler
  - Return argument from position given in *$errorMessage*

IMPLEMENTING SCALAR RETURN TYPES IN PHP5

IDEA

- *function() {} : type* is invalid php5
- *E_PARSE* cannot be handled
- Patch source code before loading
  → *spl_autoload_register()*
- Check return value before returning

SPL_AUTOLOAD_REGISTER()

- callback signature:

```
function($className) { /* register or not */ … }
```

- Callback invoked if class not yet registered
- Multiple autoloaders may be chained
- Prepending autoloader is possible

USE COMPOSER TO ...

- load classes before custom handler
- load custom handler's dependencies
- find files in custom handler
- load code from *vendor/*

```
$composerLoader = require __DIR__ . '/../vendor/autoload.php';
\TypeHints\TypeHints::init($composerLoader);
```

LOAD AND PATCH ALGORITHM

- Do not load if not found or in *vendor/*
- Transform php7 to php5 and type check manually
- *eval()* patched code → registers class

SOURCE TO SOURCE TRANSFORMATION

WHY IT'S COOL

- Fix code style
- Find bad programming patterns
- Convert old patterns to new patterns

  → Power refactoring!
- Code using new language features, backport

  → Babel already does this for JS

GET A PHP PARSER

```
$ composer install nikic/php-parser
```

- Supports php 5.2 to 7.1
- Provides:
  - lexer, parser
  - ast traversal
  - code generator

PARSE

```
$parser = (new ParserFactory())
    ->create(ParserFactory::PREFER_PHP7);
$ast = $parser->parse($code, $this->errorHandler);
```

TRAVERSE

```
class MyNodeVisitor extends NodeVisitorAbstract
{
    public function enterNode(Node $node) { ... }
    public function leaveNode(Node $node) { ... }
    public function beforeTraverse(array $nodes) { ... }
    public function afterTraverse(array $nodes) { ... }
}
$traverser = new \PhpParser\NodeTraverser();
$nodeVisitor = new MyNodeVisitor();
$traverser->addVisitor($nodeVisitor);
$traverser->traverse($ast);
```

MODIFY AST

- Do whatever you like to *$ast*
- Create Nodes of type *\PhpParser\Node\…*
- Don't modify AST while traversing

GENERATE CODE

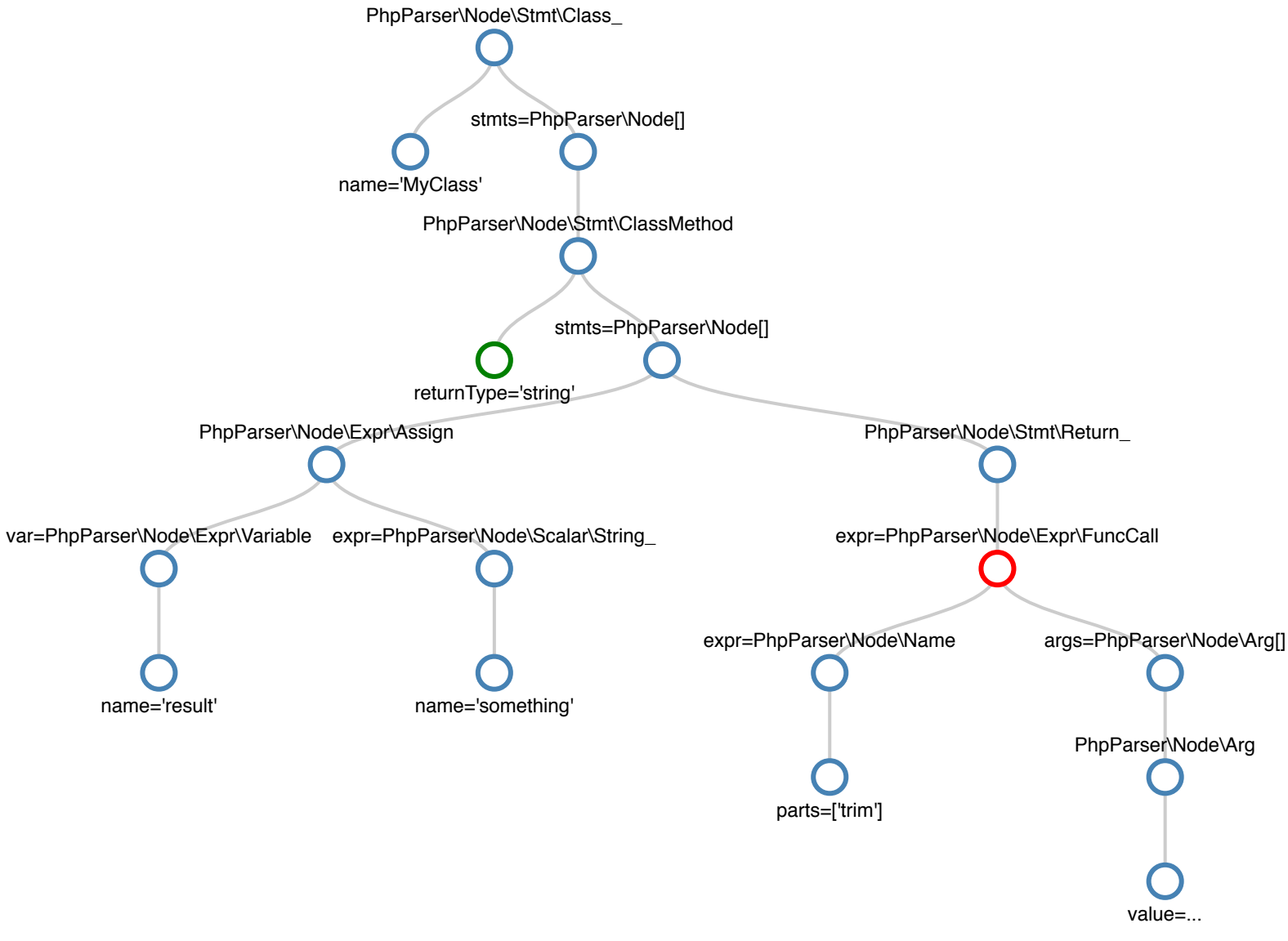
```
$generator = new \PhpParser\PrettyPrinter\Standard();
$generator->prettyPrintFile($ast);
```

TRANSFORMATION EXAMPLE

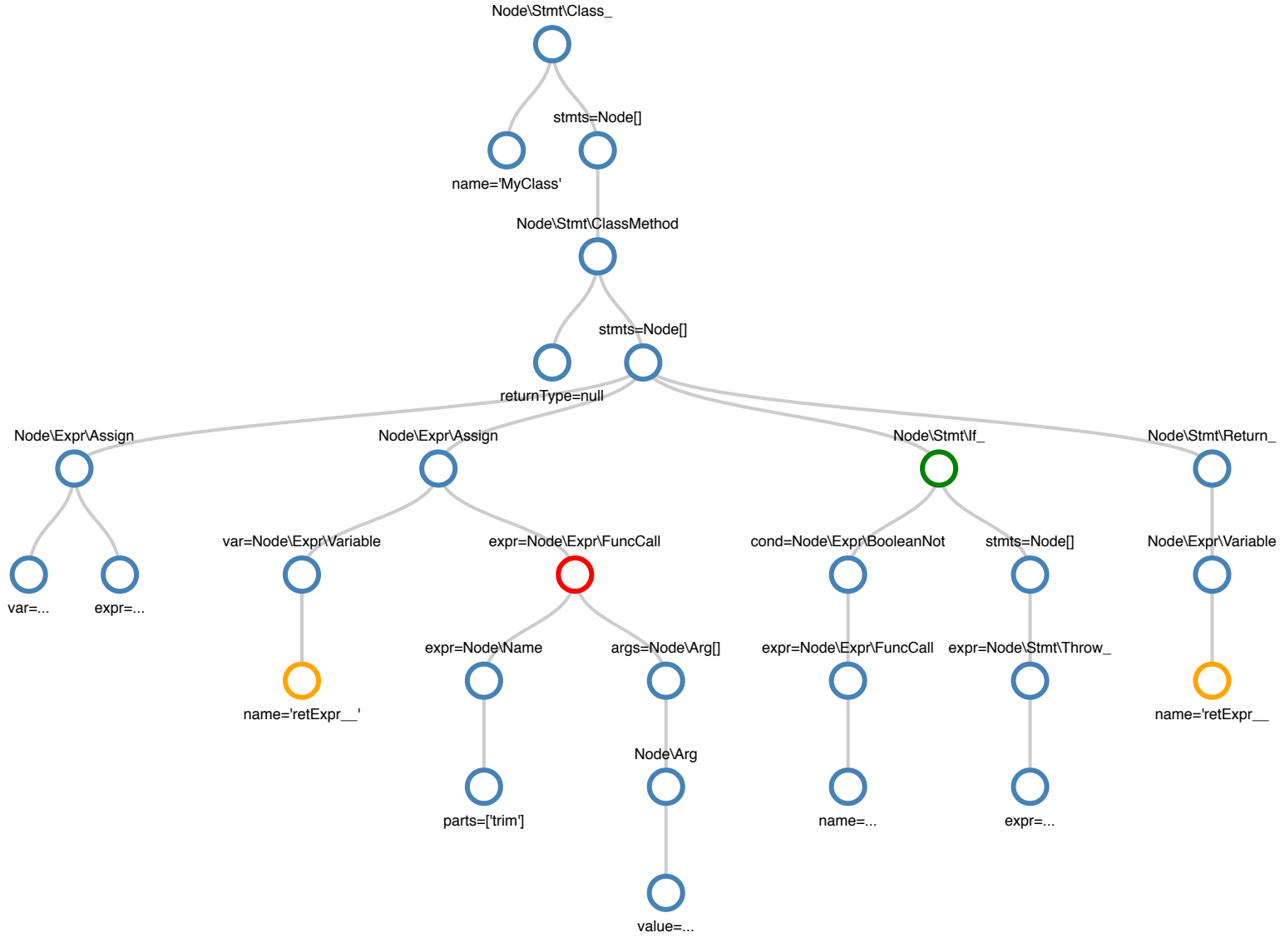
```php
public function fun() : string
{
    $result = "something";
    return trim($result + 'abc ');
}

class MyClass
{
    public function fun()
    {
        $result = "something";
        $retExpr__ = trim($result + 'abc ');
        if (!is_string($retExpr__)) {
            throw new \TypeHints\TypeHintsException(
              'Return value does not match return type string'
            );
        }
        return $retExpr__;
    }
}
```

AST BEFORE


PhpParser\Node\Stmt\Class_
name='MyClass'
stmts=PhpParser\Node[]
PhpParser\Node\Stmt\ClassMethod
returnType='string'
stmts=PhpParser\Node[]
PhpParser\Node\Expr\Assign
var=PhpParser\Node\Expr\Variable
name='result'
expr=PhpParser\Node\Scalar\String_
name='something'
PhpParser\Node\Stmt\Return_
expr=PhpParser\Node\Expr\FuncCall
expr=PhpParser\Node\Name
parts=['trim']
args=PhpParser\Node\Arg[]
PhpParser\Node\Arg
value=...

AST AFTER TRANSFORMATION



IMPLEMENTING THE TRANSFORMATION

```
$retExpr = new Variable('retExpr__');
$assignment = new Assign(
    $retExpr,
    $returnStatementNode->expr;
);
$typeCheck = new If_(
    new BooleanNot(new FuncCall(new Name('is_' . $this->returnType), [new Arg($retExpr)])),
    ['stmts' => [
        new Throw_(
            new New_(
                new FullyQualified(['TypeHints', 'TypeHintsException']),
                [new Arg(new String_('Return value does not match return type '.$this->returnType))]
            )
        )
    ]]
);
$this->insertBefore($returnStatementNode, [$assignment, $typeCheck]);
$returnStatementNode->expr = $retExpr;
```

SHOW ME THE CODE INCLUDING TESTS!

QUESTIONS?

Contact:

mathias.burger@tngtech.com,
@MathiasBurger