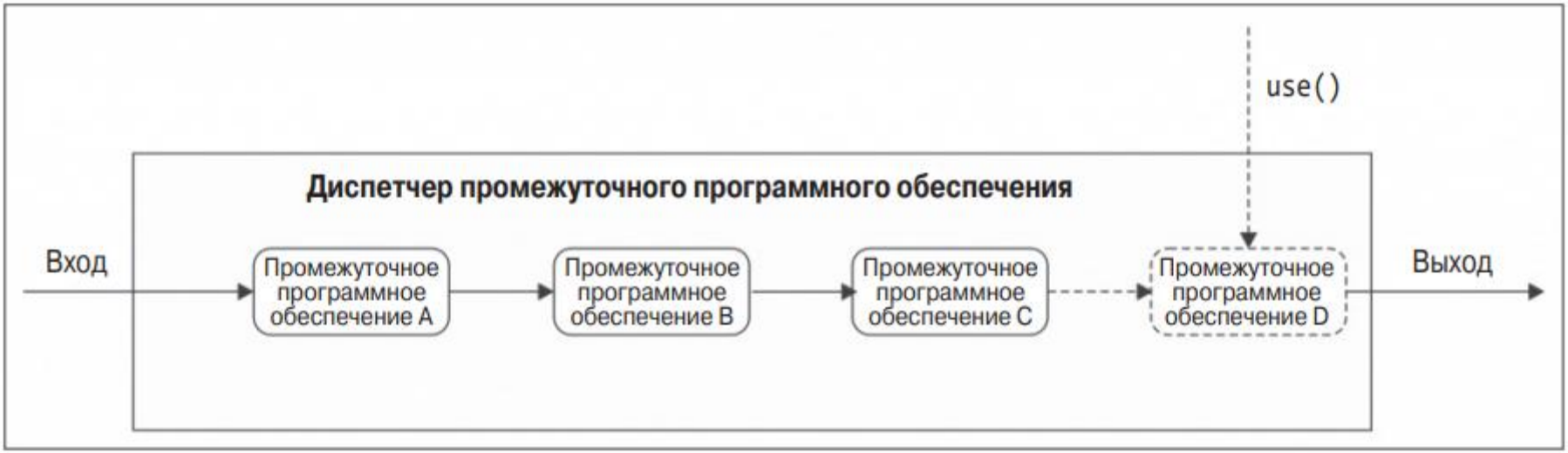

express
node
JS®

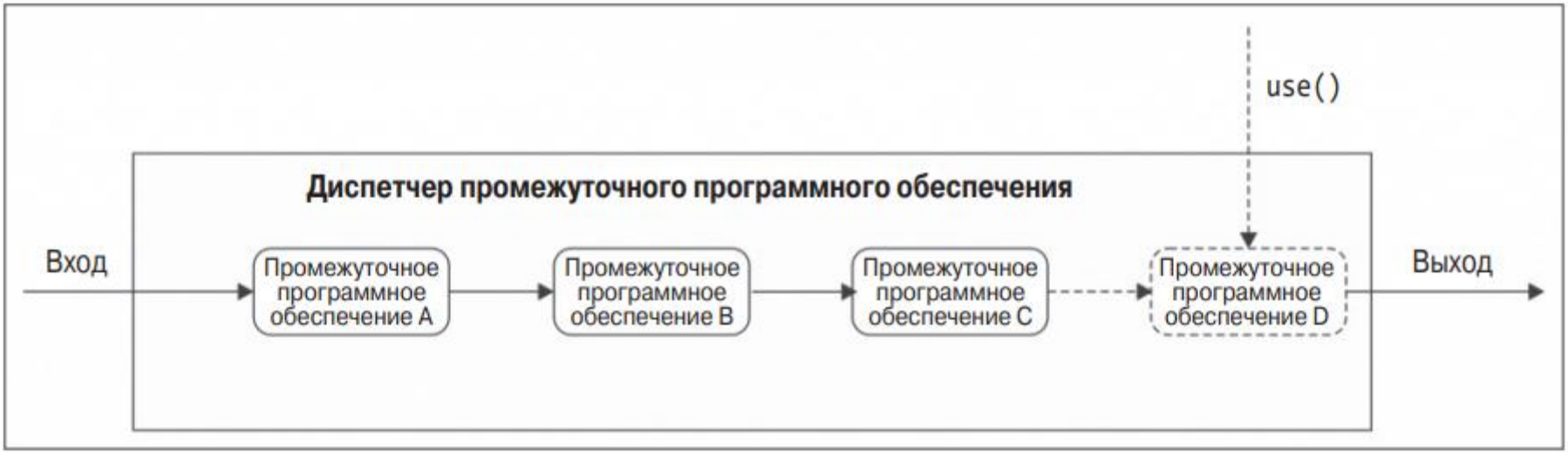
ШАБЛОН ПРОМЕЖУТОЧНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ



Шаблон позволяет создать инфраструктуру дополнительных встраиваемых модулей с невероятно малыми усилиями, обеспечивая ненавязчивый способ расширения системы новыми обработчиками.

Важным компонентом этого шаблона является **диспетчер промежуточного программного обеспечения**, отвечающий за организацию и выполнение заданий (промежуточного программного обеспечения).

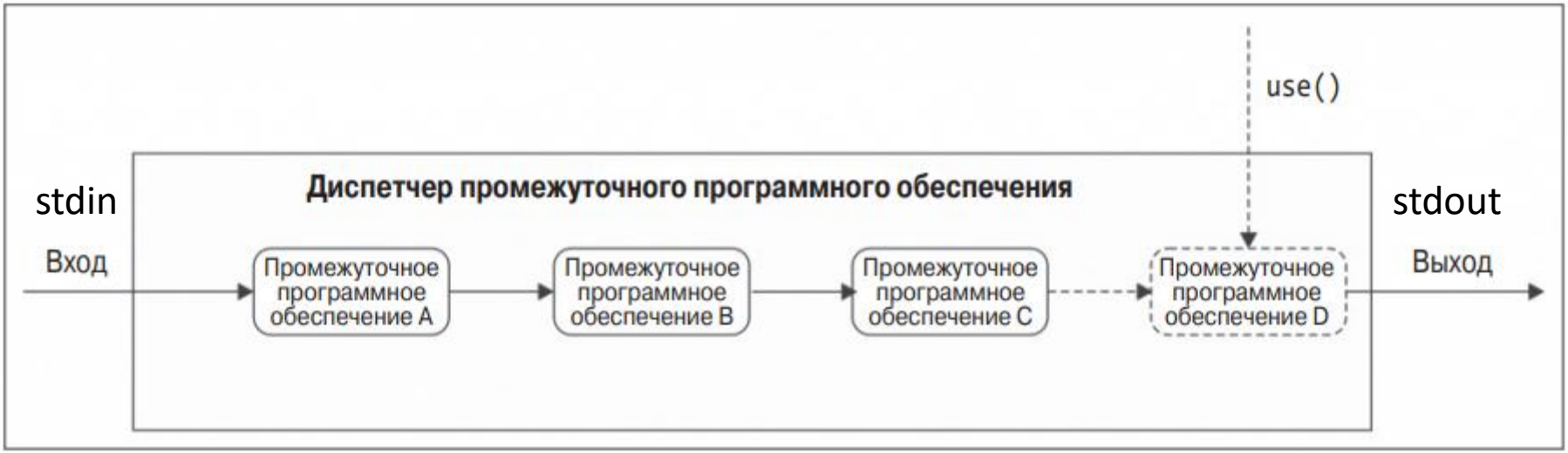
ШАБЛОН ПРОМЕЖУТОЧНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ



Детали реализации шаблона:

- ✓ Новое промежуточное программное обеспечение (ПО) можно зарегистрировать вызовом функции use() (как правило, может называться и по другому);
- ✓ Новое промежуточное ПО можно добавлять только в конец конвейера;
- ✓ Для обработки новых данных запускается асинхронный поток последовательного выполнения зарегистрированных заданий (промежуточного ПО);
- ✓ Любое задание (промежуточного ПО) может принять решение об остановке дальнейшей обработки, просто не вызывая указанную функцию или передав ей признак ошибки. Обычно ошибки вызывают выполнение другой последовательности заданий, специально предназначенной для обработки ошибок.

ЗАДАЧА



Преобразовать данные поступающие из stdin в stdout с использованием шаблона промежуточное программное обеспечение.

EXPRESS

В Express промежуточное программное обеспечение представляет набор служб, которые обычно являются функциями, собранными в конвейер и отвечающими за обработку входящих HTTP-запросов и создание надлежащих ответов.

Промежуточное программное обеспечение Express имеет следующую сигнатуру:
function(req, res, next) { ... }
где req представляет входящий HTTP-запрос, res – ответ, а next – функцию обратного вызова, вызываемую промежуточным программным обеспечением после выполнения им своих заданий, что, в свою очередь, приводит к вызову следующего промежуточного программного обеспечения в конвейере.

EXPRESS

Функции промежуточной обработки могут выполнять следующие задачи:
- Выполнение любого кода.
- Внесение изменений в объекты запросов и ответов.
- Завершение цикла “запрос-ответ”.
- Вызов следующей функции промежуточной обработки из стека.

Приложение Express может использовать следующие типы промежуточных обработчиков:
- Промежуточный обработчик уровня приложения
- Промежуточный обработчик уровня маршрутизатора
- Промежуточный обработчик для обработки ошибок
- Встроенные промежуточные обработчики
- Промежуточные обработчики сторонних поставщиков ПО

ПРОМЕЖУТОЧНЫЙ ОБРАБОТЧИК УРОВНЯ ПРИЛОЖЕНИЯ

1) без указания маршрута (сработают для всех HTTP запросов):

```
var express = require('express‘);
var app = express();

app.use(function(req, res, next) {
        console.log('Time:', Date.now());
        next(); //переход к следующему обработчику
});

app.use(function(req, res, next) {
        res.send('User');
});
```

ПРОМЕЖУТОЧНЫЙ ОБРАБОТЧИК УРОВНЯ ПРИЛОЖЕНИЯ

2) с указанием маршрута (сработает для подходящих HTTP запросов по URL):

```
var express = require('express‘);
var app = express();

app.use('/user', function(req, res, next) {
        res.send('User');
});

app.use('/',function(req, res, next) {
        res.send('Hello');
});
```

ПРОМЕЖУТОЧНЫЙ ОБРАБОТЧИК УРОВНЯ ПРИЛОЖЕНИЯ

3) с указанием метода и маршрута (сработает для подходящих HTTP запросов по методу и по URL запроса):

```
var app = require('express‘)();

app.get('/user', function(req, res, next) {
        console.log('Time:', Date.now());
        next('route'); //переход к другому app.METHOD() или router.METHOD()
});

app.use(function(req, res, next) {
        res.send('Hellou');
});
```

ПРОМЕЖУТОЧНЫЙ ОБРАБОТЧИК УРОВНЯ МАРШРУТИЗАТОРА

Промежуточный обработчик уровня маршрутизатора работает так же, как и промежуточный обработчик уровня приложения, но он привязан к конкретному маршруту:

файл user.js:

```
var express = require('express');
var router = express.Router();
router.get('/hello', function(req, res, next) {
        res.send('Hello user');
});
module.exports = router;
```

файл app.js:

```
…
var userRout = require('./routes/user.js');
…
app.use('/user', userRout);
```

ПРОМЕЖУТОЧНЫЙ ОБРАБОТЧИК ДЛЯ ОБРАБОТКИ ОШИБОК

Функции промежуточного обработчика для обработки ошибок определяются так же, как и другие функции промежуточной обработки, но с указанием для функции обработки ошибок не трех, а четырех аргументов: (err, req, res, next).

…

```
app.use('/admin/:id', function(req, res, next) {
        if(req.params.id === '0')
                return next('ID user error'); //Генерация ошибки
        res.send('User');
});

app.use(function(req, res, next) {
   res.send('Hello');
});

app.use(function(err, req, res, next) {
        console.error(err.stack);
        next(err); //передача в следующий обработчик ошибок или в стандартный обработчик ошибок (при отладке, если не установлен NODE_ENV значение production)
});
```

ВСТРОЕННЫЕ ПРОМЕЖУТОЧНЫЕ ОБРАБОТЧИКИ

Единственной встроенной функцией промежуточной обработки в Express является express.static:

```
var options = { //не обязательный аргумент
      dotfiles: 'ignore', //запрет на предоставление файлов
    //и директорий начинающихся с точки
      etag: false, //установка версий ресурса
      extensions: ['htm', 'html'], //установка альтернативных
    //версий файлов для случая отсутствия искомого
      maxAge: '1d', //время жизни в кэше
}

app.use(express.static('public', options));
```

ПРОМЕЖУТОЧНЫЕ ОБРАБОТЧИКИ СТОРОННИХ ПОСТАВЩИКОВ ПО

Установите модуль Node.js для соответствующей функциональной возможности, затем загрузите его в приложение на уровне приложения или на уровне маршрутизатора.

```
var express = require('express');
var app = express();
var logger = require('morgan');

app.use(logger('tiny')); //Логирование запросов
```

НАСТРОЙКА СЕРВЕРА / ХРАНИЛИЩЕ КЛЮЧ-ЗНАЧЕНИЕ

На уровне приложения можно использовать хранилище типа ключ-значение:

```
app.set('title', 'My Site');
app.get('title'); //вернет 'My Site'
```

Используя специальные ключи можно настраивать работу сервера Express, например:

```
app.set('views', './views'); //установка каталога, в котором находятся файлы шаблонов
app.set('view engine', 'pug'); //установка используемого шаблонизатора
app.set('x-powered-by', false); //убрать из заголовка ответа "X-Powered-By: Express"
```