Adding Custom Assessments

目录

- 基础概念
  - Creating the plugin
  - Registering the plugin with WordPress
  - 测试代码

除了在 Yoast SEO 已有的评估中添加自定义输入字段外，您还可以编写自己的评估来分析文章、页面或术语的各个方面。

注意：本示例将使用 ES6 语法，如果您尚未使用 ES6 或未使用 Babel 或 Webpack 等编译器，请确保调整代码以适应您的语法。

基础概念

编写自定义评估的首要步骤之一是确保 Yoast SEO 能够识别其存在。实现这一目标的方法是编写自定义评估、一个能够接入 Yoast SEO 的 Web Worker，以及一些额外的代码将它们整合起来。以下步骤将引导您了解创建此代码的基础知识。

Creating the plugin

First, create a file in your own plugin's `js/` directory (i.e. `js/MyCustomAssessmentPlugin.js`).

In this file, you'll have to ensure that, at a minimum, the following is present:

```
/* global YoastSEO: true, myCustomAssessmentPluginL10n */

class MyCustomAssessmentPlugin {
    /**
     * The constructor.
     */
□constructor() {
□□// Ensure YoastSEO.js is present and can access the necessary features.
```

```
  if ( typeof YoastSEO === "undefined" || typeof YoastSEO.analysis === "undefined" || typeof
YoastSEO.analysis.worker === "undefined" ) {
    return;
  }

  this.registerWorker();
 }

 /**
  * Registers the custom worker.
  *
  * @returns {void}
  */
 registerWorker() {
  const worker = YoastSEO.analysis.worker;

  worker.loadScript( myCustomAssessmentPluginL10n.script_url ).then(
    () => worker.sendMessage( "initialize", myCustomAssessmentPluginL10n,
"MyCustomAssessmentPlugin" )
  );
 }
}

/**
 * Adds eventListener on page load to load the Yoast Extended.
 */
if ( typeof YoastSEO !== "undefined" && typeof YoastSEO.analysis !== "undefined" ) {
 new MyCustomAssessmentPlugin();
}
else {
 jQuery( window ).on(
  'YoastSEO:ready',
  function() {
    new MyCustomAssessmentPlugin();
  }
 );
}
```

The above code checks whether all necessary Yoast SEO code is available and registers our custom worker and code to be used within Yoast SEO.

Note: The last part in the above JavaScript is used to properly load the JavaScript class *after* YoastSEO.js is done initializing. Please make sure you don't forget to add this.

Next, we'll create the webworker itself. In `js/`, create a file named `CustomWorker.js` and add the following code:

```
/* global analysisWorker */
import CustomAssessment from "./CustomAssessment";

class CustomWorker {
	/**
	 * Constructs a worker to be run inside the analysis web worker.
	 */
	constructor() {
		this._worker = analysisWorker;
	}

    /**
     * Registers the worker.
     */
	register() {
		this._worker.registerMessageHandler( "initialize", this.initialize.bind( this ),
"MyCustomAssessmentPlugin" );
	}

    /**
     * Initializes the worker.
     *
     * @param {Object} settings The settings to pass along to the assessment.
     */
	initialize( settings ) {
		this.customAssessment = new CustomAssessment( settings );

		this._worker.registerAssessment( "customAssessment", this.customAssessment,
"MyCustomAssessmentPlugin" );
	}
}

const customWorker = new CustomWorker();

customWorker.register();
```

What this code does, is hook into the webworkers messaging system and registers itself. Additionally, it takes our custom assessment (which we'll create later on) and registers it as well. Finally, we create a new instance of the class and immediately call the register method as we don't need to do any kind of conditional checks.

Finally, we'll create the assessment itself. In `js/`, create a file named `CustomAssessment.js` and add the following code:

```
import { AssessmentResult, Assessment } from 'yoastseo';

export default class CustomAssessment extends Assessment {
	/**
	 * The constructor.
	 *
	 * @param {Object} settings The settings to pass along.
	 */
	constructor( settings ) {
		super();
		this.settings = settings;
	}

	/**
	 * Executes the assessment and return its result.
	 *
	 * @param {Paper}      paper      The paper to run this assessment on.
	 * @param {Researcher} researcher The researcher used for the assessment.
	 * @param {Object}     i18n       The i18n-object used for parsing translations.
	 *
	 * @returns {AssessmentResult} The result of the assessment.
	 */
	getResult( paper, researcher, i18n ) {

		// Check for the occurance of the word Yoast in the title.
		const title = paper.getTitle();
		const matches = (title.match(/Yoast/gi) || []).length;

		const assessmentResult = new AssessmentResult();
		const { score, text } = this.score( matches );

		assessmentResult.setScore( score ) ;
		assessmentResult.setText( text );
```

```
  return assessmentResult;
 }

 /**
  * Determines the score based on the amount of matches.
  *
  * @param {number} matches The amount of matches.
  *
  * @returns {Object} The score object.
  */
 score( matches ) {

  if ( matches === 0 ) {
   return{
    score: 0,
    text: "Yoast isn't present in your title",
   };
  }

  if ( matches >= 5 ) {
   return{
    score: -10,
    text: "Yoast occurs too many times in your title",
   };
  }

  return{
   score: 50,
   text: "Yoast is present in your title",
  };
 }
}
```

The above code is the actual assessment itself. We take an instance of the `Paper` class, which then gets passed through the webworker that analyzes the Paper's title property. Within this title, we look for the word 'Yoast' and count the amount of instances that we can find. Based on this, we return a score (which determines whether a bullet is grey, red, orange or green) and a feedback message so the user knows why the score is what it is.

Registering the plugin with WordPress

Next, we'll have to make sure our custom JavaScript gets loaded. To achieve this, we need to make sure that our custom plugin is registered and loaded by WordPress. The main importance is that the above JavaScript file loads at the right time. Therefor, it's necessary to implement a chain of hooks that'll run once WordPress loads your plugin.

Assuming you have a plugin file present (i.e. `MyCustomPlugin.php`), your file might look a little something like this:

```
/**
 * Plugin Name: My Custom Plugin
 */

class MyCustomPlugin {

	/**
	 * MyCustomPlugin constructor.
	 */
	public function __construct() {
		// ...
		add_action( 'admin_enqueue_scripts', [ $this, 'enqueue_scripts' ] );
	}

	/**
	 * Enqueues the plugin file.
     *
     * @return void
	 */
	public function enqueue_scripts() {
		global $pagenow;

		if ( ( $pagenow == 'post.php' ) || ( get_post_type() == 'post' ) ) {
			wp_enqueue_script( 'my-custom-assessment-plugin', plugins_url( 'dist/my-custom-assessment.min.js', __FILE__ ), [], '1.0', true );
			wp_localize_script( 'my-custom-assessment-plugin', 'myCustomAssessmentPluginL10n', $this->localize_extended_script() );
		}
	}
```

```
	/**
	 * Localizes a set of data to be used in JavaScript.
	 *
	 * @return array The data to localize.
	 */
	public function localize_extended_script() {
		return [
			'script_url'  => plugins_url( 'dist/my-custom-assessment-worker.min.js', __FILE__ ),
		];
	}
}

/**
 * Loads the plugin.
 */
function loadMyCustomPlugin() {
	new MyCustomPlugin();
}

if ( ! wp_installing() ) {
	add_action( 'plugins_loaded', 'loadMyCustomPlugin', 20 );
}
```

测试代码

最后一步是测试代码。请确保您的插件已安装并启用，然后创建一篇新文章。打开 SEO 分析，查看分析结果中是否显示“标题中未包含 Yoast”的提示信息。现在将 Yoast 添加到文章标题中，观察分析结果的变化。

版本号 #2
由 壳壳虫 创建于 2026-03-27 01:08:40 UTC
由 壳壳虫 更新于 2026-03-27 01:08:40 UTC